Renaissance Robotics:
embedding multithreaded real-time feedback into mobile robots and cyber-physical systems

Thomas R Bewley

2024-03-19

Dedicated to Zachary & Nadia,
and in loving memory of
Morticia, Pareese, Morena, & Checkers, sunpups extraördinaire.

Contents

Preface

Leveraging emerging technologies for advanced cellphones and computer graphics, a vast assortment of small powerful single-board computers (SBCs), operating at very low power, are readily available today for coordinating small robotic systems, with remarkable new SBCs being announced often. At the same time, an increasing number of important applications demand creative small-scale robotic solutions, including:

- security & patrol,
- remote inspection & repair,
- mobility enhancement,
- food preparation & delivery,
- elder care & monitoring,
- nursing & feeding assistance,
- biomedical devices & prosthetics,
- minimally invasive surgery,
- pharmaceutical testing & development,
- concierge service & shopping assistance,
- precision agriculture:
  - water/pesticide/fertilizer application,
  - weed removal,
  - fruit & vegetable harvesting,
- package delivery,
- personal transportation,
- HVAC for smart grids,
- floor cleaning & laundry,
- scientific exploration,
- environmental monitoring,
- STEM education & toys.

Additionally, real-world deployments of robust AI/ML algorithms, capable of complex contextual decision making in highly consequential settings, are becoming increasingly capable for smart cars, autonomous flying taxis, and other large-scale safety-critical applications, while advanced wifi, cellular, and satellite-based internet access are becoming faster and more readily available, respectively, within buildings, across both urban and rural areas, and over the entire planet.

An important missing link for the rapid development and deployment of small-scale robotic systems leveraging such existing and emerging components, and across this growing set of needs, is the availability of easy-to-use and easy-to-extend solutions for **motor control** and the attendant high-current **voltage regulation**. In this text, we thus endeavor to fill this void by introducing a new ecosystem of cross-platform, open-design (open hardware + open-source software), self-contained, ARM-based carrier boards, dubbed **Berets**, that readily attach to (a) the 5V power, and 5V TTL logic, 40-pin header on Raspberry Pi (RPi) and compatible motherboards (MBs), including both a large number of RPi clones as well as GPU compute platforms from NVIDIA, (b) the 12V power, and 1.8V CMOS logic, 40-pin header on MBs in the 96boards CE format, including the Qualcomm Robotics RB5 platform, or (c) other small MBs via standard SPI connections, or for standalone operation.

Chapter 5 of this text provides a detailed datasheet for this new **Beret** ecosystem, including an extensible Arduino-style family of small daughterboards for further expansion options, dubbed **Beret Shields**. Concomitant with the presentation of this ecosystem, Chapter 4 introduces a uniquely extensible connector standard dubbed **Recon** that coordinates the (substantial) portfolio of connectivity options on the Berets. The rest of Part I puts these new developments in context, with surveys of some of the current and emerging technologies that enable the development of advanced mobile robots and cyberphysical systems, including discussions of:

- how today's powerful and remarkably efficient SBCs work (Chapter 1),
- essential modern programming environments and languages for embedded applications (Chapter 2),
- the dominant short-range and long-range (wired and wireless) communication protocols (Chapter 3), and
- the sensors, actuators, and interfaces available that enable new game-changing applications (Chapter 6).

Part I does not include any differential equations or advanced mathematics, and should be accessible to all "makers" (in high school and beyond) who want to significantly upgrade their technological portfolios.

Part II then provides brief introductions to some of the essential theory used in modern robotics, including:

- Robot Kinematics & Dynamics (Chapter 7),
- Numerical Methods 101 (Chapter 8),
- Signals & Systems (Chapter 9),
- Circuits (Chapter 10),
- Classical Control (Chapter 11), and
- Motion Planning (Chapter 12).
- Linear Error-Correcting Codes (Chapter 13),

A key skill that separates curious "makers" from professional "roboticists" is the *analysis* that facilitates minimalist, power-efficient, cost-effective, safe, and responsive cyber-physical design. The core material in Part II (at the level of university undergraduate courses on each of the respective subjects, often taught in the traditional engineering fields of ME, AE, EE, or CS), though by no means exhaustive, form the essential theoretical foundations for performing such analysis-based design of robotic systems. A companion volume by the same author, *Numerical Renaissance* (*NR*, occasional forward references to which are made in this text), delves much deeper (at the level of university graduate courses) into the key theories and algorithms in many related areas, specifically extending the foundations laid in Chapters 8 and 11.

Finally, Part III motivates some ideas related to robotic system design, development, and integration, including brief discussions of

- Open vs Proprietary Development Models (Chapter 14),
- Crowd Funding vs Venture Capital (Chapter 15),
- Computer Aided Design & Manufacturing (CAD/CAM) (Chapter 16), and
- various Design Paradigms (Chapter 17).

The text concludes (in Chapter 18) with a detailed case study of a fascinating educational robotics platform, dubbed myMiP, demonstrating multithreaded multirate feedback, as depicted on the book cover.

The numerical codes related to each chapter, designed to run in both Matlab and Octave, are free and open source, and are available at

https://github.com/tbewley/RR

Note that all codes in the Renaissance Robotics codebase are Copyright 2024 by Thomas Bewley, and distributed under the BSD 3-Clause License. Please help us improve this effort by submitting bug fixes, broken links, typos[1], etc. via the above site. Note that a few Easter eggs are also interspersed throughout this text (mostly as links in the pdf version), which are included in an attempt to keep you on your toes[2].

[1] The two dots over the second vowel in common words like in naïve, Noël, and reëlect is called a **diaeresis**, which may be placed over a vowel to indicate that it is sounded in a separate syllable in situations that might otherwise be ambiguous. For example, adding "co" to "operative" gives a word which might easily be mispronounced if some form of diacritic is not used. One could suggest using a hyphen, but then adding a second prefix (as is often done in scientific writing) becomes problematic: both nonco-operative and non-co-operative are downright silly, but noncoöperative works fine. This text, like the *New Yorker*, thus adopts a style that makes extensive use of diaereses. This approach is hopefully well received by anyone named Anaïs, Brontë, Chloë, Eloïse, Gaëlle, Joëlle, Maëlle, Zoë, Ismaël, Joël, Laocoön, Loïc, Maël, Noël, Raphaël, etc, reading this text; to all others, please forgive this idiösyncrasy.

[2] **Reasoning, Circular**: see explanation of **Circular Reasoning** in footnote on Page R-2.

Part I

Technology

Chapter 1

Cybernetics

Cybernetics is the science of communications and automatic control systems in machines and living things. For the desired degree of responsiveness and reliability in cyber-physical systems, the effective coordination of such machines generally requires a certain degree of **edge computing** (**decentralized**, calculated on the machines themselves). **Cloud computing** (**centralized** on large remote clusters of computers, aka servers), together with fast wired (§4.2-4.3) or wireless (§4.4) communication protocols, often complements edge computing for complex coordination tasks. We thus begin this study with a survey of the essential ideas and modern technologies that underlie the remarkable performance of both small computers and large servers today.

Contents

1.1 Bits & bytes, gates, integers, floats, and parity

We focus initially on a brief review of binary states, binary logic, and math with binary forms.

1.1.1 CMOS vs TTL logic levels; binary & hexadecimal number systems

The starting point for the **binary** (two-state) digital logic used in modern computers is the **binary digit** (**bit**), a signal voltage that is either **logical low** (near GND), **logical high** (near the supply voltage, V_{CC}), or quickly transitioning from one of these binary states to the other. In CPU cores, such transitions are synchronized with clock pulses that coordinate the corresponding computations. Different microcontrollers (MCUs) and peripherals, and indeed different regions within a single MCU, use different operating voltages.

Within CPU cores, low-voltage **complementary metal oxide semiconductor** (**CMOS**) logic levels are used, operating at a reduced voltage (usually called V_{DD}, typically less than V_{CC}), often taken as one of the following: {3.3V, 2.5V, 1.8V, 1.5V, 1.2V, 1.0V, 0.9V, ...}, with signals in the range $(0, V_{DD}/3)$ interpreted as logical low, and in the range $(2\,V_{DD}/3, V_{DD})$ interpreted as logical high. To improve performance in light of ever-decreasing transistor sizes, the value of V_{DD} used within CPU cores has been gradually decreasing over the years. Note also that most modern MCUs incorporate one or more low-dropout (LDO) regulators to provide stable power at the precise (reduced) voltage, V_{DD}, necessary for the MCU to function properly.

Between the MCU and other components (elsewhere on the motherboard, on daughterboards, or on the electromechanical system itself), **transistor-transistor logic** (**TTL**) levels are often[1] used, with the range (0, 0.8 V) interpreted as logical low, and (2 V, V_{CC}) interpreted as logical high, where V_{CC} is either 3.3 V or 5 V. MCUs with 3.3 V TTL inputs & outputs (**i/o**) can thus communicate seamlessly with 5V TTL peripherals; however (warning!) this only works if those pins set as inputs on the 3.3V TTL device are rated as **5V tolerant**, which must be checked. If they are not, a **level shifter** must be used between the two connected devices.

A collection of 4 bits is called a **nibble**, which represents a number between 0_{10} (a.k.a. 0000_2 or 0_{16}) and 15_{10} (a.k.a. 1111_2 or F_{16}), where in this text the subscript indicates the base of the number system used, with 2 denoting binary, 10 denoting decimal, and 16 denoting hexadecimal notations. Similarly, a collection of 8 bits is called a **byte**, which represents a number between 0_{10} (a.k.a. $0000\,0000_2$ or 00_{16}) and 255_{10} (a.k.a. $1111\,1111_2$ or FF_{16}). Many alternative notations are used to indicate the representation of numbers with different bases, including, for example, the representation of 184 in decimal (which is commonly indicated with no ornamentation) as 0b10111000 or 1011 1000b in binary, and as 0xB8 or #B8 in hexadecimal.

A collection of 3 bits may be used to represent a number between 0_8 (a.k.a. 000_2) and 7_8 (a.k.a. 111_2), referred to as an octal (base-8) number. Three octal digits (that is, 9 bits, denoted rwxrwxrwx) are used by the linux chmod command to set {read, write, execute} permissions on a file for the {owner, group, world}.

Tri-state (aka, ternary, three-value logic, or 3VL) is also sometimes used in creative ways in embedded systems, particularly with general purpose input/outputs (GPIOs) driving arrays of buttons and LEDs. That is, a single binary (logical 0 or 1) output signal on some pin can also be set as an input on that device, which effectively puts it into a third state Z, known as **high impedance**. Setting such a pin as a logical 0 output can, for example, drive an LED connected (through a resistor) to V_{CC}, and setting it as a logical 1 output can drive a different LED connected (through a resistor) to GND, whereas setting such a pin to the high impedance state Z (that is, setting it as an input) turns both connected LEDs off. Ternary logic circuits, operating at 3 distinct voltage levels, can also be developed; in such a setting, a ternary digit is sometimes referred to as a **trit**[2].

Using multi-level cell (MLC) flash memory technology, four-value logic (4VL; i.e., [two-bit]) is commonly used for each individual storage symbol, and both eight-value logic [three-bit, a.k.a. triple-level cell (TLC)] and sixteen-value logic [four-bit, a.k.a. quadruple-level cell (QLC)] have been developed and implemented.

[1] A notable exception is that daughterboards for the 96boards family of motherboards operate i/o at CMOS signal levels of 1.8V. Some high-voltage CMOS logic gates (e.g., those in the 74Cxx series) can operate at up to 15V; always check the data sheets!

[2] Ternary computers, based on ternary logic, were developed from 1958-1965 in the Soviet Union, but are not in use today.

SI prefix:	milli-	micro-	nano-	pico-	femto-	atto-	zepto-	yocto-
SI symbol:	m	μ	n	p	f	a	z	y
decimal power:	10^{-3}	10^{-6}	10^{-9}	10^{-12}	10^{-15}	10^{-18}	10^{-21}	10^{-24}
SI prefix:	kilo-	mega-	giga-	tera-	peta-	exa-	zetta-	yotta-
SI symbol:	k	M	G	T	P	E	Z	Y
decimal power	10^{3}	10^{6}	10^{9}	10^{12}	10^{15}	10^{18}	10^{21}	10^{24}
binary prefix	kibi-	mebi-	gibi-	tebi-	pebi-	exbi-	zebi-	yobi-
binary symbol	Ki	Mi	Gi	Ti	Pi	Ei	Zi	Yi
binary power	2^{10}	2^{20}	2^{30}	2^{40}	2^{50}	2^{60}	2^{70}	2^{80}
binary/decimal ratio	1.024	1.0486	1.0737	1.0995	1.1259	1.1529	1.1806	1.2089

Table 1.1: Decimal powers, as used in SI, and binary powers, as used in characterizing computer systems.

A large number of bits (abbreviated with a lowercase b) or bytes (abbreviated with an uppercase B) is indicated using a prefix corresponding to a binary power that is close to, but not quite the same as, the corresponding decimal power used in the International System of Units (SI; see §9.1.1-9.1.2), as indicated in Table 1.1. Thus, unambiguously, a Kib is 1024 bits, a KiB is 1024 bytes, a MiB is 1,048,576 bytes, a GiB is 1,073,741,824 bytes, etc. Quite unfortunately, as of 2024, SI prefixes (representing decimal powers) are still used quite often for the nearby binary powers in the computer literature, commonly denoting 1024 bits as a kb (or Kb), 1024 bytes as a KB, 1,048,576 bytes as a MB, 1,073,741,824 bytes as a GB, etc. We eschew this (sloppy) dominant paradigm in this text, simply by inserting an "i" as the second character of each prefix when denoting storage capacities, communication speeds, etc, as the percentage uncertainty that is introduced by doing otherwise increases as you move to the right in Table 1.1 (which is certainly the trend when quantifying storage capacities and communication speeds as time goes forward!), and encourage hardware manufacturers, retailers, tech reporters, book/wikipedia authors, researchers, instructors, bloggers, gamers, and others to do the same.

1.1.2 Binary logic gates

The notion of voltages along wires considered as **bits**, in states denoted **logical low** (near 0 V) and **logical high** (near V_{CC} or V_{DD}), were defined for TTL and CMOS logic levels in digital electronics in §1.1.1. We next show how three digital gates (NOT, NAND, and NOR[3]) can be implemented in a CMOS setting, using complementary pairs of p-type and n-type enhancement-mode MOSFETs, as discussed further §9.2.3. From there,

- more complex boolean algebra may be implemented using combinational logic, with responses defined by truth tables relating current inputs and outputs, as illustrated in Tables 1.2 through 1.8, and
- more advanced computations may be implemented using synchronous logic, with responses defined by state transition tables relating current inputs and states to subsequent states, as illustrated in Table 1.9.

Note also the following (Fact 1.1 is actually just two special cases of Fact 1.2):

Fact 1.1 (De Morgan's Theorem) *An OR logic gate is equivalent to a NAND logic gate with inverted inputs, and an AND logic gate is equivalent to a NOR logic gate with inverted inputs.*

Fact 1.2 (Functional completeness of NAND and NOR) *(Sheffer 1913) Any Boolean expression can be achieved by a connection of NAND logic gates, or a connection of NOR logic gates.*

The following fairly low-level discussion of the logic gates implemented in modern digital electronics is included here, early in this text, to remove some of the mystery related to how computers and MCUs actually work.

[3]That is to say, NAND, NOT, and NOR, will get you pretty far!

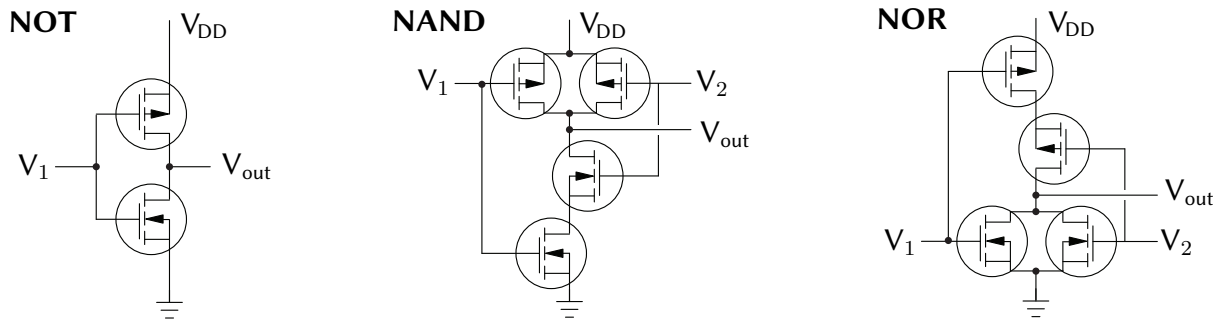


Figure 1.1: Three essential logic gates, NOT, NAND, and NOR, similarly constructed using enhancement-mode complementary MOSFETs (CMOS), which result in **break before make** to avoid **shoot through**.

The CMOS NOT gate in Figure 1.1 operates as follows. When **V_1 is low** (close to GND):
- the channel of the lower (NMOS) transistor is high resistance, effectively **disconnecting V_{out} from GND**,
- the channel of the upper (PMOS) transistor is low resistance, effectively **connecting V_{out} to V_{DD}**.

Conversely, when **V_1 is high** (close to V_{DD}):
- the channel of the upper (PMOS) transistor is high resistance, effectively **disconnecting V_{out} from V_{DD}**,
- the channel of the lower (NMOS) transistor is low resistance, effectively **connecting V_{out} to GND**.

The CMOS NAND gate uses 2 upper (PMOS) transistors in parallel, and 2 lower (NMOS) transistors in series. Thus, when **either or both of V_1 and V_2 is/are low**:
- at least one of the lower (NMOS) transistors is high resistance, effectively **disconnecting V_{out} from GND**,
- at least one of the upper (PMOS) transistors is low resistance, effectively **connecting V_{out} to V_{DD}**.

Conversely, when **both V_1 and V_2 are high**:
- both of the upper (PMOS) transistors are high resistance, effectively **disconnecting V_{out} from V_{DD}**,
- both of the lower (NMOS) transistors are low resistance, effectively **connecting V_{out} to GND**.

The COMOS NOR gate uses 2 upper (PMOS) transistors in series, and 2 lower (NMOS) transistors in parallel. Thus, when **both V_1 and V_2 are low**:
- both of the lower (NMOS) transistors are high resistance, effectively **disconnecting V_{out} from GND**,
- both of the upper (PMOS) transistors are low resistance, effectively **connecting V_{out} to V_{DD}**.

Conversely, when **either or both of V_1 and V_2 is/are high**:
- at least one of the upper (PMOS) transistors is high resistance, effectively **disconnecting V_{out} from V_{DD}**,
- at least one of the lower (NMOS) transistors is low resistance, effectively **connecting V_{out} to GND**.

The resulting truth tables for these three gates are given in Tables 1.2-1.3. Note that **enhancement-mode** MOSFETs (explained further §9.2.3) are used in Figure 1.1; this choice is significant, as such MOSFETs are generally nonconducting between the Source and the Drain until a sufficiently large voltage is applied between the Gate and the Body (which as depicted in Figure 9.7 is usually connected directly to the Drain), thereby largely preventing the **shoot through** of current that would otherwise be associated with a path opening up, momentarily, directly between V_{DD} and GND when one of the inputs changes state. This approach is called **break before make**; the transistors breaks the connection in one direction before making one in the other.

Older literature on digital logic discusses PMOS and NMOS implementations of logic gates at length:
- NMOS implementations replace the upper (PMOS) transistors in Figure 1.1 with "**pull-up resistors**", which are effectively overpowered at the output by the lower transistor(s) when they are open.
- PMOS implementations replace the lower (NMOS) transistors in Figure 1.1 with "**pull-down resistors**", which are effectively overpowered at the output by the upper transistor(s) when they are open.

With advances in transistor technology, both types of logic circuits have largely been superseded by CMOS implementations (without resistors, as seen in in Figure 1.1), which are significantly more power efficient.

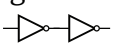
A →	0	1	symbol	best construction	ICs
NOT	1	0		(Figure 1.1a)	4049, 7404
BUFFER	0	1			4050, 7434

Table 1.2: Truth tables and symbols of the two 1-input logic gates, and part numbers of some (of many) convenient families of ICs implementing such gates. Note that, though less efficient, a NOT gate may be constructed from a NAND or a NOR gate (see Table 1.3) with its inputs tied together (see Fact 1.2).

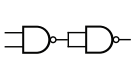
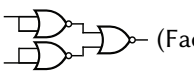
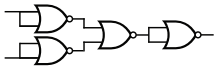
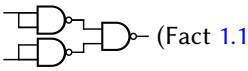
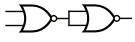
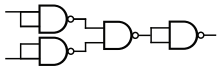
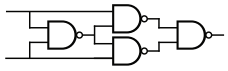
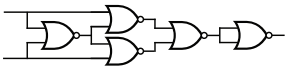
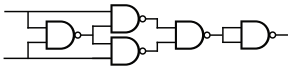
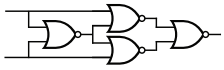
A,B →	0,0	0,1	1,0	1,1	symbol	NAND construction	NOR construction	ICs
AND	0	0	0	1			 (Fact 1.1)	4081, 7408
NAND	1	1	1	0		(Figure 1.1b)		4011, 7400
OR	0	1	1	1		 (Fact 1.1)		4071, 7432
NOR	1	0	0	0			(Figure 1.1c)	4001, 7402
XOR	0	1	1	0				4070, 7486
XNOR	1	0	0	1				4077, 74266

Table 1.3: Truth tables and symbols of the main 2-input logic gates, their construction with just NAND gates or NOR gates (see Fact 1.2), and part numbers of some convenient families of ICs implementing 4x such gates.

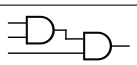
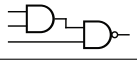
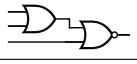
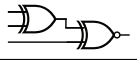
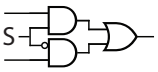
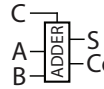
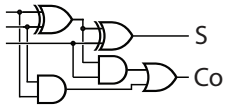
A,B,C →	0,0,0	0,0,1	0,1,0	0,1,1	1,0,0	1,0,1	1,1,0	1,1,1	symbol	construction	ICs
AND	0	0	0	0	0	0	0	1			4073, 7411
NAND	1	1	1	1	1	1	1	0			4023, 7410
OR	0	1	1	1	1	1	1	1			4075, 744075
NOR	1	0	0	0	0	0	0	0			4025, 7427
XOR	0	1	1	0	1	0	0	1			74386
XNOR	1	0	0	1	0	1	1	0			
MUX	0	0	1	1	0	1	0	1			4540, 74153 (2x 4:1)
Adder: S	0	1	1	0	1	0	0	1			4008, 7483 (4 bit)
Co	0	0	0	1	0	1	1	1			

Table 1.4: Truth tables and symbols of several 3-input logic gates, and examples of their construction. All can be constructed using the gates in Tables 1.2 and 1.3 (and, thus, from just NAND or just NOR gates). The last column indicates part numbers of some convenient families of ICs implementing such gates. The **multiplexer** MUX(S,X,Y) assumes S=A is select, and X=B and Y=C are the input data, and sends the selected input to the output; note that an open circle on an input is a shorthand denoting a NOT operation on that input, just as an open circle on an output (e.g., on the NAND and NOR symbols) denotes a NOT operation on that output. The outputs of a single-bit **full adder**, denoted here S and Co, assume A and B are corresponding bits of two binary numbers to be added, and C is the carry over from the sum of the lesser bits; S is then the corresponding bit of the sum, and Co is the carry over into the sum of the next greater bits.

Once the notions of logical levels (0/1) and gates (NOT/BUFFER and AND/NAND/OR/NOR/XOR/XNOR[4]) are developed (see Figure 1.1 and Tables 1.2-1.3), we can put them together to accomplish a variety of important higher-level tasks (see Tables 1.4-1.9), such as AND/NAND/OR/NOR/XOR/XNOR gates with 3 or more inputs, multiplexers / demultiplexers, simple and priority encoders / decoders, adders, SR latches, and JK flip flops.

Modern ICs that implement such higher-level tasks are generally formed directly, via efficient combinations of CMOS transistors that are arranged as necessary to satisfy the truth tables of the desired logic gates. This process results in logic gates that are much simpler and faster than logic gates formed via pedagogical combinations of 2-input NAND and NOR gates, as illustrated in these tables. As just one example, a 3-input NAND gate can be formed simply by modification of a 2-input NAND gate (see Figure 1.1), with 3 upper (PMOS) transistors in parallel and 3 lower (NMOS) transistors in series. As indicated, convenient families of integrated circuits (ICs) with efficient implementations of many such higher-level logic gates, such as the 4000 and 7400 series, are readily available. Such ICs come in many different families, with vastly different nominal operating voltages and current-driving capabilities. The details of their internal constructions are certainly important to those designing such ICs; however, only their timing and power specifications, as described in their corresponding datasheets, are really all that is important to those using them.

The logical behavior of the several gates summarized in Tables 1.2 through 1.9 should (with some effort) be largely self explanatory from their construction, as should the reasoning for their names. The patterns evident in the simple example constructions provided should also give substantial evidence upon which larger versions of such gates may also be constructed, if a specific gate needed is not readily available as a single IC.

Note in particular that a **multiplexer** (MUX) directs the selected input channel (amongst 2^n possible inputs, where n is the number of select lines) to the (one) output channel, and a **demultiplexer** (DEMUX) directs the one input channel to the selected output channel, sending 0 to all the others [see examples in Tables 1.4-1.6]. The symbols for multiplexers and demultiplexers thus often, schematically, indicate multiple-position mechanical switches within, with 2^n positions on one side connecting to one on the other, even though their interior "wiring" is actually accomplished with logic gates. Digital multiplexers allow several different digital channels to share, at different times, a single expensive peripheral, such as a communication channel.

The outputs of a **single-bit full adder** [see Table 1.4, last row] assume A and B are corresponding bits of two binary numbers to be added, and C is the carry over from the sum of the lesser bits; S is then the corresponding bit of the sum, and Co is the carry over into the sum of the next greater bits. **Multiple-bit full adders** [see, for example, Table 1.6] take n bits of two binary numbers A and B, and the possible carry over C from the sum of lesser bits, and efficiently generate the corresponding n bits of the sum S, as well as the possible carry over Co for inserting into the sum of the next greater bits (or, signaling an overflow). Adders are useful to, well, add.

Now imagine an alarm system in which 2^n sensors normally read 0; if a sensor is triggered (changes[5] to 1), it is desirable to alert the owner as to *which* sensor was triggered. This can be accomplished using a small MCU leveraging a 2^n-input, $(n+1)$-output **simple encoder**, which represents the (one) channel that was triggered as an n-bit binary number, plus a flag to indicate when one has triggered. Conversely, a small MCU can trigger 2^n distinct actions (open/close a window, turn on/off a fan, ...), one at a time, using an $(n+1)$-input, 2^n-output **decoder**. [See $n = 2$ examples of each in Table 1.7a.] Taking $n = 6$, with the appropriate encoder or decoder, allows one to monitor 64 different sensors, or to activate (at different times) 64 distinct actions, using only 7 GPIO channels on the MCU coordinating the system, providing a clear path to extend its reach.

[4]XNOR is really "NOT XOR", so logically should be called NXOR; however, by convention, it usually goes by "XNOR". Go figure.

[5]We describe here a setting in which the "active state" is 1, termed **active high**. Most alarms are **active low**, with power provided to/through all the sensors (so they normally read 1); if any circuit is interrupted (at the sensor, or in the wires leading to it), the channel returns 0, and a fault is triggered. This way, even if an accident or bad actor cuts one or both of the wires (red or blue, it doesn't matter) leading to a normally-closed sensor, the alarm is still triggered. [However, if an informed (or, clever) adversary knows that a certain sensor is active low, and can get access to the wires leading to it, he can simply splice the two wires together, thus defeating the sensor.] With a minor effort (by Fact 1.1, replacing OR gates with NAND gates and AND gates with NOR gates), logic circuit designs like these may be converted from active high to active low (see, e.g., the equivalent constructions in Table 1.8).

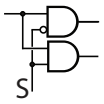
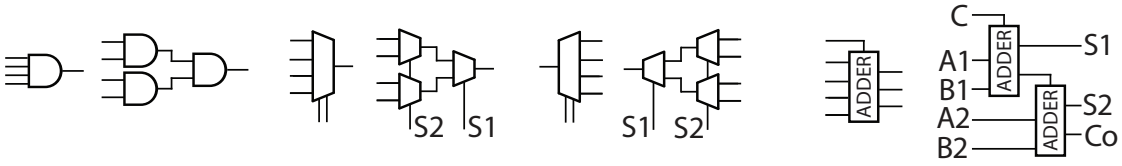
S,D →	0,0	0,1	1,0	1,1	symbol	construction	ICs
A	0	1	0	0			4052, 74139 (2x 1:4)
B	0	0	0	1			

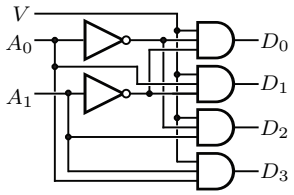
Table 1.5: Truth table and symbol of a 2-output **demultiplexer** (DEMUX), which sends the input to the selected output (and 0 to the other), an example of its construction, and part numbers of ICs implementing such gates.

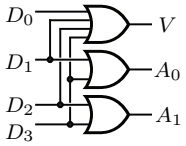


4-input AND 4-input MUX 4-output DEMUX 2-bit full adder

Table 1.6: Symbols and constructions of a few larger logic gates. Larger AND/NAND/OR/NOR/XOR/XNOR gates may all be constructed following the patterns evident at right in Table 1.4. A MUX, given n select lines S1 to Sn, directs the value of the selected input (amongst 2^n possible inputs) to the (one) output; a DEMUX directs the value of the (one) input to the selected output (amongst 2^n possible outputs), sending 0 to the others. An n-bit full adder takes as inputs the bits A1 to An, B1 to Bn, and the possible carry over C from the sum of lesser bits, and generates the bits of the sum S1 to Sn, and the carry over Co into the sum of the next greater bits.

D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	0	0	1	1
0	1	0	0	1	0	1
1	0	0	0	1	1	1





D_3	D_2	D_1	D_0	A_1	A_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

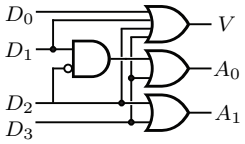


Table 1.7: (a) Truth table (from inputs $\{A_1, A_0, V\}$ to outputs $\{D_3, D_2, D_1, D_0\}$) of a **2:4 decoder**, and (from inputs $\{D_3, D_2, D_1, D_0\}$ to outputs $\{A_1, A_0, V\}$) of a **4:2 simple encoder**, and example constructions of each; note the $n+1$ outputs of a simple encoder are ill defined for other values of the 2^n inputs. (b) Truth table (from inputs $\{D_3, D_2, D_1, D_0\}$ to outputs $\{A_1, A_0, V\}$) of a **4:2 priority encoder**, and an example construction; note the $n+1$ outputs of a priority encoder are all well defined for all possible values of the 2^n inputs. An input listed as x means "don't care"; the output for that case is as listed for any value of this input.

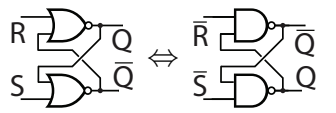
S,R → / $\overline{\text{S}}$,$\overline{\text{R}}$ →	0,0 / 1,1	0,1 / 1,0	1,0 / 0,1	1,1 / 0,0	symbol	equivalent constructions	IC
Q	(latch)	0 (reset)	1 (set)	disallowed!			4043 (4x)
$\overline{\text{Q}}$	(latch)	1	0				

Table 1.8: Truth table, symbol, and construction of a **Set/Reset** (**SR**) **latch**. Taking $\{\text{S,R}\} = \{1, 0\}$ **sets** $\text{Q} = 1$, taking $\{\text{S,R}\} = \{0, 1\}$ **resets** $\text{Q} = 0$, and taking $\{\text{S,R}\} = \{0, 0\}$ **latches** (holds) Q at its most recently-specified (set or reset) state; overline denotes the opposite state. The condition $\{\text{S,R}\} = \{1, 1\}$ is not allowed.

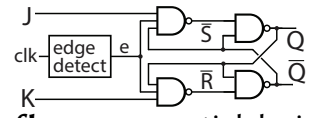
J,K →	0,0	0,1	1,0	1,1	symbol	construction	IC
Q_{next}	Q (latch)	0 (reset)	1 (set)	$\overline{\text{Q}}$ (flip)			4027 (2x)
$\overline{\text{Q}}_{\text{next}}$	$\overline{\text{Q}}$ (latch)	1	0	Q (flip)			

Table 1.9: State transition table, symbol, and construction of a **JK flip flop**, a sequential device which updates Q only at each rising edge of a clock (see Figure 1.2). Taking $\{\text{J,K}\} = \{1, 0\}$ **sets** $\text{Q}_{\text{next}} = 1$, taking $\{\text{J,K}\} = \{0, 1\}$ **resets** $\text{Q}_{\text{next}} = 0$, taking $\{\text{J,K}\} = \{0, 0\}$ **latches** $\text{Q}_{\text{next}} = \text{Q}$, and taking $\{\text{J,K}\} = \{1, 1\}$ **flips** $\text{Q}_{\text{next}} = \overline{\text{Q}}$.

Figure 1.2: Edge detect circuits that limit the JK flip flop to flip states only once during each clock pulse, during (left) the rising edge of each pulse, or (right) the falling edge of each pulse.

The simplistic logic in a simple encoder generally results in a misleading output when multiple sensors are triggered. To handle this situation fully, 2^n inputs to the MCU processing the information would be required. However, if we can prioritize *which* sensors are "most important" in a given system, we can implement a **priority encoder**, as illustrated in Table 1.7b. Consider for example, a system with (high-priority) fire alarms, (low-priority) leak detectors, and (medium-priority) magnetic sensors detecting whether or not the windows are closed. If multiple sensors are triggered, a well-designed priority encoder lets the user know the number of the channel in which the highest priority sensor detects a fault.

Many devices (LEDs, fans, ...) can be driven in an "activate and forget" mode, in which one signal can be sent to turn the device "on", and a second signal can later be sent to turn the device "off"; at other times, the device does not need to be actively connected to the controlling MCU. This functionality is easily implemented with a **set/reset (SR) latch**, as illustrated in Table 1.8, with pull-down resistors attached to its inputs, so the inputs are taken as low when the device is disconnected from the controlling MCU; the logic high construction of an SR latch is built from 2 NOR gates, and the logic low construction is built from 2 NAND gates; either may be built using a total of 8 transistors (see Figure 1.1). As evident by its construction, briefly sending an SR latch the "set" command $\{S, R\} = \{1, 0\}$ (say, using the output of a decoder) is sufficient to put (and, to hold, even when S returns to 0) the output of the latch (that is, the input to the device) in the "on" state $\{Q, \overline{Q}\} = \{1, 0\}$, until the SR latch is sent the "reset" signal $\{S, R\} = \{0, 1\}$, which is sufficient to put (and, to hold, even when R returns to 0) the output of the latch in the "off" state $\{Q, \overline{Q}\} = \{0,1\}$.

The **JK flip flop** generalizes the SR latch upon which it is built (see Table 1.9) by making effective use of the fourth possible state of the inputs, $\{J, K\}$, to **flip** the current state of the latch. It coordinates this flip using a system **clock**, which is simply a binary signal that regularly switches back and forth (usually, quite quickly) from 0 to 1. There are two possible output conditions: the "on" condition $\{Q, \overline{Q}\} = \{1, 0\}$ and the "off" condition $\{Q, \overline{Q}\} = \{0, 1\}$, and there are four possible states of the inputs $\{J, K\}$. Normally, $\{J, K\} = \{0, 0\}$ and $\{\overline{S}, \overline{R}\} = \{1, 1\}$, which holds the (logic low) latch in the second stage of the JK flip flop at its current state. The internal states $\{\overline{S}, \overline{R}\}$ can switch to other values, but only under 4 specific conditions.

- If $\{J, K\} = \{1, 0\}$ and $\overline{Q} = 1$, then $\overline{S} \rightarrow 0$ when e next goes high, setting the latch to $\{Q, \overline{Q}\}_{next} = \{1, 0\}$.
- If $\{J, K\} = \{0, 1\}$ and $Q = 1$, then $\overline{R} \rightarrow 0$ when e next goes high, resetting the latch to $\{Q, \overline{Q}\}_{next} = \{0, 1\}$.

After the output changes state in both cases, the internal states return to the latched condition $\{\overline{S}, \overline{R}\} = \{1, 1\}$.

- If $\{J, K\} = \{1, 1\}$ and $\overline{Q} = 1$, then $\overline{S} \rightarrow 0$ when e next goes high, flipping the latch to $\{Q, \overline{Q}\}_{next} = \{1, 0\}$.
- If $\{J, K\} = \{1, 1\}$ and $Q = 1$, then $\overline{R} \rightarrow 0$ when e next goes high, flipping the latch to $\{Q, \overline{Q}\}_{next} = \{0, 1\}$.

As long as $\{J, K\} = \{1, 1\}$, the JK flip flop will continue switching output states, very quickly, until e goes low; this condition is called **race**. To avoid this undesirable condition, the **edge detect** circuit indicated needs to hold the signal e high only for a few nanoseconds, at the rising or falling edge of each clock pulse. Two edge detect circuits that limit the JK flip flop to flip states only once during each clock pulse are given in Figure 1.2. The value of e output by these simple circuits is zero except in a very narrow slice of time in which the clk signal has switched but the output of the inverter has not yet switched. If the delay associated with this inverter is too short for the flip flop to function reliably, the single inverter can be replaced by three inverters.

The JK flip flop is useful for constructing counters in timers and clocks. Note also that:

- Setting $J = K \triangleq T$ creates a **T flip flop**; when $T = 1$, the T flip flop toggles the state of Q at each clock pulse, thus performing **frequency division**, creating (in Q) a clock that oscillates at half the frequency of clk.
- Setting $J = \overline{K} \triangleq D$ creates a simple **D flip flop** useful for storing data; when $D = 1$, it is set ($Q_{next} = 1$), and when $D = 0$ it is reset ($Q_{next} = 0$). The D flip flop is not susceptible to race (edge detection is unnecessary).

int8	uint8	int16	uint16	int32	uint32	int64	uint64
$-128:127$	$0:255$	$-32{,}768:32{,}767$	$0:65{,}535$	$-2^{31}:2^{31}-1$	$0:2^{32}-1$	$-2^{63}:2^{63}-1$	$0:2^{64}-1$

Table 1.10: Ranges covered by $N = 8$, 16, 32, and 64 bit binary representations of signed and unsigned integers, with $2^{31} = 2{,}147{,}483{,}648$, $2^{32} = 4{,}294{,}967{,}296$, $2^{63} = 9{,}223{,}372{,}036{,}854{,}775{,}808$, $2^{64} = 18{,}446{,}744{,}073{,}709{,}551{,}616$.

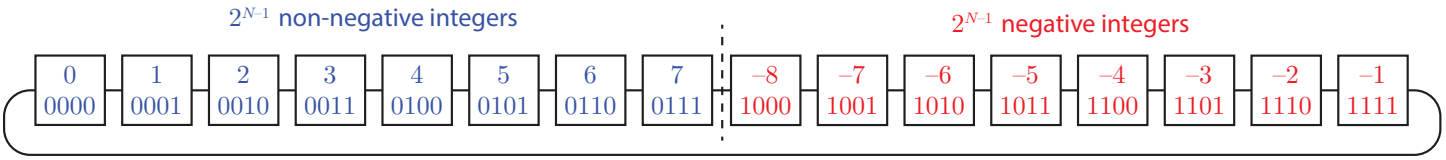


Figure 1.3: Periodic number line useful in visualizing the **two's complement** convention.

1.1.3 Integer & fixed-point representations, and their (fast) arithmetic in ALUs

Integer arithmetic on MCUs is usually formed using binary representations of integers that are $N = 8$, 16, 32, or 64 bits long, and either unsigned or signed, covering the (decimal) integer ranges indicated in Table 1.10.

When storing or transmitting a multiple-byte **word** (containing one or more integers, fixed-point real numbers, or floating-point real numbers; see §1.2) in a computer, the individual bytes stored (or, transmitted over a communication channel) that make up such a word can be ordered using one of two different conventions:

- with the **big endian** convention, the "big end" (that is, the most significant byte, aka **MSB**, in the sequence) is stored first (at the lowest storage address), or transmitted first, whereas
- with **little endian** convention, the "little end" (the least significant byte, or **LSB**) is stored or transmitted first.

For example, the two bytes (16 bits) required for representing the integer A2F3_{16} is stored as A2_{16} at memory address a and F3_{16} at memory address $a+1$ using the big-endian convention, and the same integer is stored as F3_{16} at memory address a and A2_{16} at memory address $a+1$ using the little-endian convention. Within a byte, the order of the bits is usually stored the same (most significant bit to least significant bit) in all computers, regardless of how the bytes are arranged; however, the terms big endian vs little endian may also be used to characterize the order in which individual bits are transmitted over a communication channel.

Signed representations of negative integers are formed using the **two's complement** convention, illustrated for the $N = 4$ case in Figure 1.3, with **negation** given simply by inverting (with NOT gates) the bits of the corresponding integer (in binary form) and adding one. This effectively scoots the set of all 2^{N-1} negative integers included in the representation to the right of the 2^{N-1} non-negative integers on a number line ordered by the raw (unsigned) binary number. Adding 1 (that is, $0\ldots01_2$) to any number on this periodic number line shifts it to the right by one, modulo 2^N (i.e., moving off the right end of the line wraps back around on the left end[6]. Similarly, adding -1 (that is, 1…11) to any number on this line corresponds to shifting it to the left by one, modulo 2^N (that is, moving off the left end of the line wraps back around on the right end). Leveraging this two's complement convention, as summarized by this number line, a regular (unsigned) N-bit full adder[7], as introduced in Table 1.6, corresponds to the **addition** of any positive and negative integers. Of course, **subtraction** is achieved simply by negation of one of the numbers, followed by addition.

Recall that multiplication and long division of binary numbers follow precisely the same process as the mul-

[6]To perform a addition or multiplication of integers, one should be careful to monitor for **integer overflow**, which corresponds to exceeding the range indicated in Table 1.10 (in the case of unsigned integers, this corresponds to falling off the left end or the right end of the unsigned number line; in the case of signed integers, this corresponding crossing the halfway point on the number line, indicated by the vertical dashed line in Figure 1.3 in the $N = 4$ case), and flag an error if it happens. On the other hand, a mod 2^N operation on unsigned integers can be performed quite quickly, simply by ignoring such integer overflow conditions.)

[7]Again, this adder should be implemented ignoring binary overflow condition, corresponding to the "wrapping around" of the operation around the ends of the number line.

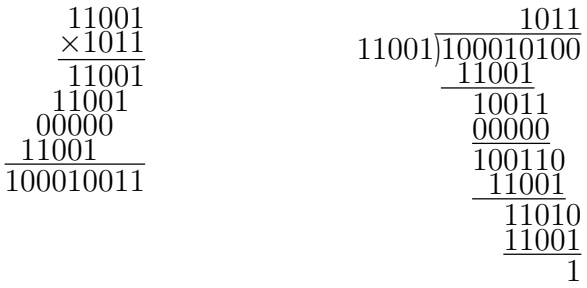


Figure 1.4: (left) Binary multiplication [showing that $11001_2 \times 1011_2 = 100010011_2$, or $25_{10} \times 11_{10} = 275_{10}$], and (right) binary division [showing $100010100_2/11001_2 = 1011_2$, or $276_{10}/25_{10} = 11_{10}$, with remainder 1].

tiplication and long division of decimal numbers as you learned in grammar school (see Figure 1.4). That is, the binary negation, addition, and subtraction operations outlined above, sequenced appropriately, are sufficient to perform binary **multiplication** and **division**.

All modern CPU cores include (fast) **hardware implementations** (by an **arithmetic logic unit**, or **ALU**) of the {**negation**, **addition**, **subtraction**, **multiplication**, **division**} operations discussed above, on both unsigned integers, and signed integers represented in two's complement binary form. Remarkably, these operations all execute in a **single clock cycle**, as the N-bit full adder upon which they are all based, as introduced in Table 1.6, implements combinational logic as discussed previously, with responses defined by truth tables that relate current inputs and outputs (that is, they do not need a clock to coordinate them).

Binary representations of unsigned or signed integers, and the fast (ALU) implementations of $\{+, -, \times, \div\}$ acting thereon, can also be applied directly to real (rational) numbers with a fixed (specified in advance) number of binary digits after the (implied) decimal point. This representation of **fixed point** real numbers, using N bits, is referred to as **Q format**, and is commonly denoted UQ$m.n$ [a.k.a. UQn] for unsigned real numbers, and Q$m.n$ [a.k.a. Qn] for signed real numbers (in two's complement format), where n indicates the number of binary digits after the decimal point, and (optionally) m indicates the number of binary digits before the decimal point, with $m + n = N$. Addition and subtraction of two fixed-point real numbers [once aligned to the same Q format, so they have the same number of binary digits after the (implied) decimal point] is the same as integer addition and subtraction using binary representations; again, integer overflow must be checked for and flagged if it occurs. Multiplication and division of two fixed-point real numbers is, conceptually, the same as integer multiplication and division using binary representations. In addition, note that:

• the product of a Q$m_1.n_1$ fixed point real number and a Q$m_2.n_2$ fixed point real number generally results in a Q$m.n$ fixed point real number with $m = m_1 + m_2$ and $n = n_1 + n_2$, and

• the division of a Q$m_1.n_1$ fixed point real number by a Q$m_2.n_2$ fixed point real number generally results in a Q$m.n$ fixed point real number with $m = m_1 - m_2$, but n may be infinite.

These results must be both rounded (reducing the number of significant digits kept after the decimal point) and checked for overflow in order to fit it into another N bit Q format representation. As much as possible, scaling all fixed-point real variables in a problem (both before and after all the sums, differences, products, and divisions) to be $O(1)$ over the entire operational envelop of the electromechanical system under consideration is particularly convenient, using, e.g., the UQ1.7 (in the range $[0, 1.99219_{10}]$), Q1.7 (in the range $[-1, 0.99219_{10}]$), UQ1.15 (in the range $[0, 1.9999695_{10}]$), and Q1.15 (in the range $[-1, 0.9999695_{10}]$) formats[8]. Note that:

• To convert a real number r into Q$m.n$ format, multiply r by 2^n, round to the nearest integer, and convert this integer to two's complement binary form.

• To convert a number b in Q$m.n$ format back to a real number, consider b as a regular binary number (with no decimal point), convert this binary number (in two's complement form) to an integer, and divide by 2^n.

[8] In general, the range of a UQ$m.n$ number is $[0, 2^m - 1/2^n]$, and the range of a UQ$m.n$ number is $[-(2^{m-1}), 2^{m-1} - 1/2^n]$.

1.1.4 Floating-point representations, and their (fast) arithmetic in FPUs

It is, of course, significantly easier to program, especially at the prototyping stage, using **floating-point arithmetic** [that is, using real numbers represented with a **sign bit**, an **exponent**, and a **significand** (a.k.a. **mantissa**)], so that the scaling of the real numbers can be managed by the CPU core on the fly, and a very wide range of scalings can be encountered without loss of precision. Floating point real numbers, as defined by the ubiquitous IEEE 754 standard, are represented with:
• $N = 16$ bits ("half precision"), with 1 sign bit, 5 bits defining the exponent, and $k = 10$ bits defining the significand, representing numbers from $\pm 6.10 \times 10^{-5}$ to ± 65504 with $\log_{10} 2^{k+1} = 3.3$ decimal digits of precision,
• $N = 32$ bits ("single precision"), with 1 sign bit, 8 bits defining the exponent, and 23 bits defining the significand, representing numbers from $\pm 1.18 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$ with 7.2 decimal digits of precision, or
• $N = 64$ bits ("double precision"), with 1 sign bit, 11 bits defining the exponent, and 52 bits defining the significand, representing numbers from $\pm 2.23 \times 10^{-308}$ to $\pm 1.80 \times 10^{308}$ with 16 decimal digits of precision.

For the feedback control of electromechanical systems, single precision is more than enough, and in most cases half precision is sufficient (if the FPU implements it; as of 2024 most do not, though Armv8.1-M introduces hardware support for half-precision floats to the ARM Cortex M family, starting with the Cortex M55 & M85).

In addition to nonzero normal numbers (that is, floating-point numbers that can be represented in half, single, or double precision as defined above, without leading zeros in their significand), various special values are represented and handled correctly by FPUs implementing the IEEE 754 standard, specifically:
• signed zeros $\{+0, -0\}$ [with $(+0) = (-0)$ for the purpose of comparisons],
• signed infinities $\{+\infty, -\infty\}$ [e.g., $1/(+0) = (+\infty)$, $1/(-\infty) = (-0)$, $(+\infty) * (-2) = (-\infty)$, ...],
• subnormal numbers [that is, smaller floating-point numbers that can be represented in half, single, or double precision at reduced precision, with leading zeros in their significand],
• Not a Numbers (NaNs), handling indeterminant forms [e.g., $(\pm\infty) \times (\pm 0)$, $(\pm 0)/(\pm 0)$, $(+\infty) + (-\infty)$, ...], real operations with complex results [e.g., $\sqrt{-1}$], and operations involving one or more NaNs as arguments.

For example, taking s as the sign bit, e as the exponent, and f as the fractional part of an $N = 32$ bit binary representation of a floating-point number in single precision format as follows,

s	e (8 bits)								f (23 bits)																			
b_{31}	b_{30}	b_{29}	b_{28}	b_{27}	b_{26}	b_{25}	b_{24}	b_{23}	b_{22}	b_{21}	b_{20}	b_{19}	b_{18}	b_{17}	b_{16}	b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0

and defining $e_{\max} = FF_{16} = 255_{10}$ and $e_{\text{off}} = 7F_{16} = 127_{10}$, the IEEE 754 standard interprets cases with:
• an exponent e of 01_{16} to $(e_{\max} - 1)$ as denoting a nonzero normal number given by $(-1)^s \times 2^{e - e_{\text{off}}} \times 1.f$
• an exponent e of 00_{16}, with $f \neq 0$, as denoting a subnormal number given by $(-1)^s \times 2^{-(e_{\text{off}} - 1)} \times 0.f$,
• an exponent e of 00_{16}, with $f = 0$, as denoting a signed zero, with sign given by $(-1)^s$,
• an exponent e of $e_{\max}$, with $f = 0$, as denoting a signed infinity, with sign given by $(-1)^s$, and
• an exponent e of $e_{\max}$, with $f \neq 0$, as denoting an NaN.

The half precision ($N = 16$ bit) format is analogous, with $e_{\max} = 1F_{16} = 31_{10}$ and $e_{\text{off}} = F_{16} = 15_{10}$; the double precision ($N = 64$ bit) format is also analogous, with $e_{\max} = 7FF_{16} = 2047_{10}$ and $e_{\text{off}} = 3FF_{16} = 1023_{10}$.

Interrogation of the individual bits of a floating-point representation might occasionally be useful to the embedded programmer, and in this setting the above explanation should suffice. The actual encoding of the fundamental operations $\{+, -, \times, \div\}$ on real numbers represented in floating-point notation is rather complex, and is taken care of remarkably quickly (again, in many cases, executing in a single clock cycle!) by the **floating point unit**s (**FPU**s) within modern CPU cores, and the MCUs which embed them.

Integer arithmetic (§1.1.3) is significantly simpler for a processor to execute than floating-point arithmetic. Thus, many auxiliary processing units (see §1.5.3-1.5.4), like FMACs and DSPs, and indeed many low-cost MCUs (like the ARM Cortex M0 and some implementations of the M3 and M4), do not include hardware FPUs, and thus any floating-point arithmetic performed must instead be emulated in software on these processors, which

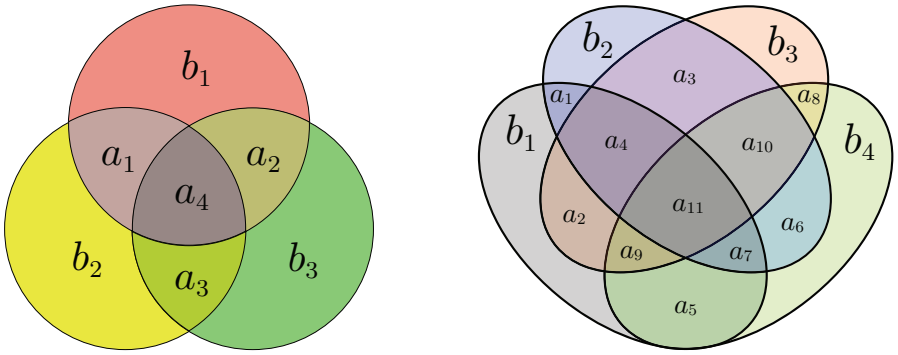


Figure 1.5: Venn diagram illustrations of (left) the [7,4] Hamming code and (right) the [15,11] Hamming code.

is relatively slow. In such settings, it is strongly preferred to use fixed point arithmetic instead, carefully scaling all real numbers in the problem to make full use of the fixed point binary representations used while never encountering overflow over the entire operational envelop of the electromechanical system under consideration (note that this usually takes considerable testing of the system to verify).

1.1.5 Parity checks, error detection, and error correction

When pushing certain subsystems (memory and communication in particular) to their physical limits (high speed, low power, small footprint, etc.), occasional bit errors may occur. There are a variety of simple and effective ways to identify such infrequent errors, and in certain cases even to correct for them.

The simplest approach is to append a **single parity bit** to each set of k data bits that is stored in memory or sent over a communication channel; this parity bit is selected such that the sum (modulo 2) of all the data bits in the set, plus this parity bit, is 0 (if **even parity** is used) or 1 (if **odd parity** is used). When the entire set of $n = k + 1$ bits (data plus parity) is recalled from memory or received on the other end of the communication channel, this sum is again performed, and an error is flagged if it is of the wrong value. This approach is capable of **single error detection** (SED), with two or more errors in any set of n bits causing misinterpretation; note, however, that if single bit errors are random and infrequent, double bit errors will be extremely infrequent.

The idea of using parity bits to check for errors may be extended to facilitate stronger error detection, and even error correction. As shown in Figure 1.5, this is illustrated by the $[n, k]$ **linear binary codes** (LBCs) with:

- $r = 3$ parity bits $\{b_1, b_2, b_3\}$, $k = 4$ data bits $\{a_1, a_2, a_3, a_4\}$, and $n = r + k = 7$ total bits in a $[7, 4]$ LBC, or
- $r = 4$ parity bits $\{b_1, b_2, b_3, b_4\}$, $k = 11$ data bits $\{a_1, \ldots, a_{11}\}$, and $n = r + k = 15$ total bits in a $[15, 11]$ LBC.

In each of these example LBCs, an r set Venn diagram may be drawn with exactly one of the k data bits in each of the intersections. The r parity bits $\{b_1, \ldots, b_r\}$ are then selected such that parity (say, even parity) is achieved by summing the 2^{r-1} bits in each of the r sets in this Venn diagram. If a recalled/received set of n bits is assumed to be corrupted by at most one error, then during the subsequent parity checks of all r sets,

- if parity fails on just a single set, the corresponding parity bit b_i is itself identified as corrupted, whereas
- if parity fails on multiple sets, the data bit a_i corresponding uniquely to that set intersection is corrupted.

In either case, flipping the corrupted bit corrects the error, thus performing **single error correction** (SEC). This approach extends immediately to $[2^r - 1, 2^r - 1 - r]$ LBCs for $r \geq 2$, known as **binary Hamming codes**.

Adding an overall parity bit to the cases shown in Figure 1.5 allows one to detect (if the overall parity check fails) and correct (as before, leveraging the other parity checks) single bit errors, but also to detect but not correct double bit errors (if the overall parity check passes, but two or more of the other parity checks fail), leading to the **single error correction, double error detection** (SECDED) $[2^{r-1}, 2^r - r]$ LBCs for $r > 2$, known as **extended binary Hamming codes**. The idea of storing or sending multiple redundant bits is extended in §1.5.3.3 and §12, to develop and implement LBCs capable of fast multiple bit error detection and correction.

1.2 Central Processing Unit (CPU) cores

Central processing unit (CPU) cores are the main processing units in modern computers. An essential defining characteristic of modern CPUs is the **word size**, which defines

(a) the number of bits in the **data bus** (the parallel wires carrying data within the CPU),
(b) the number of bits in the **memory addresses**, and
(c) the number of bits in the **instruction codes** enumerating the low-level executable commands in the CPU,

all of which are generally integer multiples of the word size, which on modern CPUs is 8, 16, 32, or 64 bits.

Doubling the width of the data bus doubles the amount of information that can be delivered from point A to point B within the CPU in a single clock cycle, but substantially increases the complexity of the circuitry; different tradeoffs are thus reached for the width of the data bus for different CPU designs.

Common memory configurations in modern MCUs include 16 address bits, facilitating the direct addressing of 64 KiB of memory, and 32 address bits, facilitating the direct addressing of 4 GiB of memory. Note that, in many CPUs, the number of physical address pins implemented can actually be less than or even (with a bit of additional logic) greater than the number of address bits. In particular, the 64 address bits of some modern 64-bit CPUs (that is, CPUs with a word size of 64 bits) facilitate the addressing of an absurdly large amount of memory (16 EiB); 64-bit CPUs thus typically implement only between 40 and 52 physical address pins, facilitating the direct addressing of 1 TiB to 4 PiB of memory (reminder: see §1.1.1 for definitions of binary powers).

There are two primary types of computer architectures (i.e., the set of rules that describe the organization of computer systems), the Harvard architecture, which strictly separates memory storage and signal busses for program instructions from those for data, and the von Neumann architecture, in which instructions and data share the same memory and busses. Modern implementations of the Harvard architecture usually relax the strict separation between instructions and data, allowing the instruction memory to actually be accessed as data, and are thus more accurately referred to as Modified Harvard architectures.

There are also two primary types of instruction set architectures (ISAs), RISC (**reduced instruction set computer**) and CISC (**complex instruction set computer**), in addition to a growing number of hybrid approaches that are increasingly blurring the lines between the two. RISC ISAs (pioneered by MIPS, perfected by ARM, and redesigned in an open-standard setting by RISC-V) have a small set of simplified (fixed-length) instructions operating on a large number of registers, and a streamlined instruction pipeline allowing a reduced number of clock cycles per instruction. In contrast, CISC ISAs (notably implemented and perpetuated by x86 CPUs) have a larger set of more complex (variable-length) instructions operating on a smaller number of registers, with each instruction executing a variable number of low-level operations (e.g., load something from memory, perform some arithmetic, store result back in memory). Note that RISC ISAs generally access memory through dedicated simple instructions, whereas CISC ISAs access memory as an integral part of their more complicated (multi-step) instructions.

Modern families of CPUs and MCUs appropriate for embedded applications include

- ARM Cortex A (32- and 64-bit), as implemented by Amlogic, Broadcomm, Rockchip, Samsung, TI Sitara, …,
- ARM Cortex R (32- and 64-bit), as implemented by TI, …
- ARM Cortex M (32-bit), as implemented by Cypress, Infineon, Microchip, Nuvoton, NXP LPC, STM32, …,
- Numerous (32-bit and 64-bit) RISC-V CPUs & MCUs, as implemented by Codasip, Microchip, Espressif, …,
- Intel 8051 (8-bit), as implemented by Cypress, Maxim, Silicon Labs, …,
- Tensilica Xtensa (64-bit), as implemented by Espressif, …
- NVIDIA Carmel (64-bit),
- Qualcomm Kryo (64-bit),
- Microchip AVR (including ATtiny and ATmega) and PIC (8-, 16-, and 32-bit),
- TI MSP430, MSP432, and C2000 (16- and 32-bit),

and many many others; most in this list (except the Intel 8051) are designed around RISC CPU cores.

1.3 Cache-based memory subsystems

The ALU and FPU of a CPU can approach their full speeds doing useful computations only if they can: (a) quickly access both the instructions to be performed next, as well as the data necessary to perform these instructions, and (b) quickly shift the results of these computations to somewhere secure for later use.

As a general rule, the smaller the data storage subsystem, the faster it can be made, but at a significant cost. Ordered from fastest/most expensive/largest footprint per byte on down, the primary storage technologies are:

- Static Random Access Memory (SRAM): 1-5 ns access time, volatile (data lost when powered down). Expensive!
- Dynamic Random Access Memory (DRAM): 5-25 ns access time, volatile, frequent refreshes (~ 1 Hz) required.
- Flash Memory / SD Cards / EEPROM[9]: 50-500 ns access time, non-volatile, limited write endurance.
- Solid State Drives (SSD[10]): 10-100 μs access time, non-volatile, hot swappable, limited write endurance.
- Hard Disk Drives (HDD): 5-20 ms access time, non-volatile, hot swappable, excellent write endurance. Cheap!

Significantly, as the size of a data storage subsystem grows, it generally becomes easier to download/upload increasingly large *blocks* of data, all at essentially the same time, at relatively little added cost (time and energy).

To reduce the mean access time & energy, and overall expense & physical size, required of the memory system (all of which are important in embedded applications), the communication between the CPU and the main memory (DRAM or Flash) [and, to even slower "disk" storage[11]] is often assisted by a hierarchy of smaller/faster cache memory (SRAM & DRAM), together with a **memory management unit** (MMU) or **memory protection unit** (MPU) coordinating its use. Cache memory is often divided into multiple levels, including:

- **L1i**, for queueing up the instructions to be performed next, and
- **L1d**, **L2**, **L3**, and **L4** (or a subset thereof[12], with the smaller numbers enumerating the faster/smaller "lower" levels of the cache hierarchy), both for bringing data to the handful of **registers** holding the data actually used by the ALU and FPU, and for storing the results of the computations performed back in the main memory.

When using a cache-based memory system, small fixed-size **cache blocks** (aka **cache lines**) of contiguous memory are downloaded/uploaded whenever updating the lower levels of the cache hierarchy[13], and larger cache blocks are downloaded/uploaded whenever updating the higher levels of the cache hierarchy, or communicating between the highest level of cache (aka the **last level cache**) and the main memory itself.

The CPU also usually includes a translation lookaside buffer (**TLB**), which translates the virtual addresses used by each program to their corresponding physical addresses in the main memory, for both the instructions to be executed as well as the corresponding data storage[14].

The majority of the silicon area on most modern CPUs is in fact taken up by the MMU, the TLB, and the L1i, L1d, and (sometimes) L2 and L3 memory caches, thus indicating the importance of the cache-based memory system to the overall CPU performance (higher levels of cache, if used, are often incorporated on separate ICs). The several components of a modern cache-based memory system usually interact quite efficiently with little if any intervention by you, the embedded programmer. However, a high-level understanding of how such systems behave can assist you in implementing certain programming directives that can make such systems run even better, and to streamline the data flow when the CPU stalls due to cache conflicts.

[9] Flash comes in two types, NAND and NOR. Flash is a type of EEPROM designed for high speed and density, with large erase blocks ($\gtrsim 512$ bytes) and limited write endurance ($\sim 10^4$ write cycles). The term "EEPROM" is saved for non-volatile memory built with the same technology, but with small erase blocks (1 to 8 bytes) and better write endurance ($\sim 10^6$ write cycles).

[10] SSDs are self-contained subsystems using flash memory together with their own cache memory to both increase effective speed and improve endurance. Many of the concepts discussed in this section extend directly to the control of cache-based SSD systerms.

[11] "Disk" storage may refer to both filesystems and virtual memory on both SSDs and HDDs.

[12] How many levels of cache should be implemented for the best overall system performance generally depends on the total amount of main memory accessible by the system, and the ratio of the CPU speed to the main memory speed, which is often much slower.

[13] At any given level, a **cache entry** generally includes both the copied data as well as a **tag** indicating the corresponding range of addresses in the main memory.

[14] The TLB is often split into an Instruction TLB and Data TLB, and may be split into levels (e.g., L1 ITLB/DTLB, L2 ITLB/DTLB, ...).

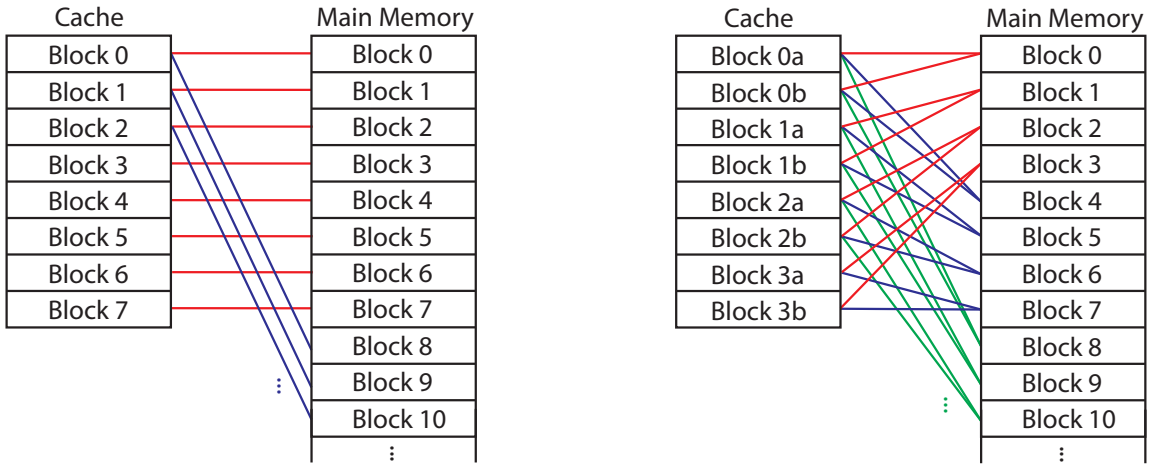


Figure 1.6: Illustrations of (left) a direct mapped cache, and (right) a two-way set associative cache.

When initiating a read or write to/from a particular memory location, the CPU first checks to see if a copy of that memory location is already represented in its L1 cache. If it is (constituting a **cache hit**), the CPU interfaces directly, and quite quickly, with this highest-speed cache. If it is not (a **cache miss**), the MMU must look in successively higher levels of (lower-speed) cache, all the way out to the main memory if necessary, to reach the relevant memory location. The MMU may also create a new cache block, at one or more levels of the cache, containing this memory location together with nearby entries of the main memory; to do this, it must generally **evict** one of the existing cache blocks at each affected level.

Where, exactly, such a new cache block may be placed within a cache is governed by the placement policy associated with that cache level, which may allow the new cache block to placed:

(a) at just a single location, based on the least significant bits of the corresponding memory address block, called a **direct mapped** cache (see Figure 1.6a);
(b) at any of N locations (typically, $N = 2$, 4, or 8), based on the least significant bits of the memory address and the replacement policy used (discussed below), called an N**-way set associative** cache (see Figure 1.6b);
(c) at either of 2 locations, following either the direct-mapped policy mentioned above or a hash function pointing somewhere else, called an **two-way skew associative** cache; or
(d) anywhere it wants, called a **fully associative** cache.

If the placement policy allows a choice to be made in the placement of the new cache block [see (b), (c), and (d) above], this decision is made by the replacement policy of the MMU. Amongst many possible such policies, one common choice is to evict the **least-recently used** cache block. The larger the number of choices in the placement policy, the more places that need to be searched in cache for the requested memory location, but the less likely a very recently loaded cache block (possibly containing useful information for impending calculations) will need to be evicted to make room for a new cache block.

When compiling code for cache-based memory systems, the general goal is to maximize the percentage of cache hits (aka the **hit rate**) in the lowest levels of cache. This goal is achieved with algorithms that are compiled with high degrees of locality of reference, including both **temporal locality**, in which certain variables are reused repeatedly, and **spatial locality**, in which the data needed for subsequent computations is generally stored physically close to each other in the main memory (and is thus likely already present in existing cache blocks, which are loaded when preparing for the preceding computations).

The MMU must implement a rather involved set of rules in order to achieve cache coherence; that is, to make the entire multi-level cache-based memory system appear, for the purpose of programming simplicity, as a single, unified, very fast memory system. The MMU achieves this by carefully coordinating both the *reading* of the main memory and the higher levels of cache by the lower levels of cache, as well as the *writing* of the

data generated by the CPU back to the various levels of cache and, ultimately, back to the main memory.

When reading saved data from the main memory into the various levels of cache, there are two types of approaches that the MMU may implement. With **inclusive** cache designs, which are the most common, smaller and smaller sub-blocks of the data stored in the higher levels of cache and the main memory are duplicated into each successively lower level of cache. This approach simplifies the connection between the various levels of cache (keeping the ankle bone connected to the leg bone, etc), thereby simplifying the problem of maintaining cache coherence, but increases the communication between the various cache levels. With **exclusive** cache designs, on the other hand, two caches never share the same data. This approach avoids repetition, shuns redundancy, eshews reiteration, and resists recapitulation, but leaves the placement policy of the MMU (and/or the embedded programmer, via compiler directives) with the question which data to put into which levels of cache.

When writing the new data generated by the CPU back out to the various levels of cache and, ultimately, all the way to the main memory, there are two types of **write policies** that may be implemented. When using a **write through** policy at a particular cache level, newly updated data at that cache level is copied back immediately to the corresponding section of the next higher level of cache or the main memory. This approach allows the cache block to be overwritten immediately with a different section of memory when necessary, but increases the amount of communication between cache levels. When using a **write back** policy at a particular cache level, on the other hand, the updating of the next higher level of cache or the main memory with the updated data at that cache level is deferred until the corresponding cache block soon needs to be evicted to make room for the caching of a different section of memory. This approach reduces the communication between the different cache levels as well as the number of data writes, which is more efficient, but introduces a possible delay between when the "eviction notice" is received by a particular cache block, and when that block is actually ready to cache a different section of memory. Note that it is particularly important to use a write back policy to the main memory and to SSD when either is implemented on flash, which has limited write endurance.

Whenever a cache contains updated data that has not yet been copied up to the next higher level of cache and the main memory, that section of cache is said to be **dirty**. Note also that, in multicore and multi-CPU systems, a typical cache implementation might be configured as follows:

- each core has a dedicated L1 cache,
- each CPU has a dedicated L2 cache, shared amongst its multiple cores, and
- the entire system has a single L3 cache, shared amongst its multiple CPUs.

Higher levels of cache and the main memory may thus be updated by other CPU cores, as well as by certain peripherals with **direct memory access** (DMA). Whenever a cache contains old data that has not yet been copied down from the next higher level of cache and the main memory, that section of cache is said to be **stale**. Substantial care must be taken by the MMU to keep track of both the dirty and the stale sections of cache at all levels, and to update them when appropriate, in order to keep the cache coherent.

Steps an embedded programmer can take to use cache-based memory systems more efficiency include:

1) structuring and ordering computations in the compiled code to maximize both temporal and spatial locality,
2) keeping certain memory locations, for variables that are reused repeatedly [e.g., indices $\{i, j, k, \ldots\}$, constants c_i, and temporary variables t_i], locked in cache,
3) implementing write through policies for the lower-level cache blocks used primarily for data input to the CPU, which need to quickly replaceable,
4) implementing write back policies for cache blocks used primarily for data storage to the main memory, to minimize unnecessary communication between cache levels,
5) bypassing the use of cache altogether for certain data that is only accessed occasionally, and
6) manually **flushing** (copying back to higher levels) cache blocks that will not be needed again soon.

Good programming languages, through compiler directives, give the programmer a degree of control over such low-level memory management operations, which can make a big difference in the execution speed of a code.

1.4 Hardware for exploiting parallelism

Most numerical algorithms can be arranged such that the majority of the calculations performed do not actually depend upon the results of the immediately preceding calculations. Such situations allow for **parallel computing**, in which some calculations may be done simultaneously (or, nearly so) with others, allowing the entire algorithm to complete much more quickly. Parallelism within numerical algorithms is quantified by its **granularity**: problems with **fine-grained parallelism** have a relatively small number of calculations that may be performed independently before their results must be shared in order to continue, whereas problems with **coarse-grained parallelism** have a relatively large number of computations that may be performed independently before their results must be shared in order to continue. Problems with coarse-grained parallelism naturally evident at the outset of the problem formulation are sometimes said to be **embarrassingly parallel**.

The identification of techniques to expose and exploit parallelism is essential for two key reasons. First, of course, identifying parallelism allows the computer's operating system (see §2) to assign multiple compute resources to the problem at hand simultaneously [i.e., the various ALUs and FPUs within the different CPU cores in the system (see §1.4.3), together with certain other compute resources that may also be available, as surveyed in §1.5.3-1.5.4]. This enables significantly more computational work to be completed per clock cycle.

Equally important, at a lower level, identifying parallelism allows a **self-optimizing compiler** to make much more effective use of all available levels of high-speed cache memory (see §1.3) for each individual CPU core being used, by performing a delicate *regrouping* and *reordering* of the various computations to be performed, thus maximizing both the temporal and spatial locality of the data needed for each and every calculation to be performed along the way. This is best achieved by adhering to the following high-level guidelines:

(a) Write clean codes that clearly/simply reveal the problem structure at hand (e.g., if your computer language allows it, somehow writing A*B for matrix/matrix multiplication, or A\b for Gaussian elimination, instead of looping over all of the individual indices involved in such basic but time-consuming computations yourself).

(b) Use a modern self-optimizing compiler that calls the BLAS (basic linear algebra subprograms) and LAPACK (linear algebra package) software libraries extensively (or, if the programming language or compiler you are using doesn't do this for you, call these routines yourself from within your code, and consider changing to a different programming language or compiler!). These libraries are meticulously hand tuned by each CPU vendor to maximize hit rates in each level of cache for the fundamental linear algebra problems that your code will spend the bulk of its time solving at any given problem size. You are unlikely to do better on your own.

(c) If at all possible, define the problem size at *compile time*, via constants defined in the code header, rather than at run time, via data files that are read in (post compilation). This important (but, often-overlooked) third guideline helps the compiler to decide, at compile time, specifically how to reorder the various loops involved in order to achieve maximum performance from the cache. Indeed, for many (large) problems, the advantage here is so significant that recompiling the code in question immediately before any large run, once the size of the problems to be solved are identified and defined in the code header, can be quite beneficial.

Most numerical algorithms can actually be arranged (or, rearranged) to reveal a *hierarchy* of parallelism within, with some fine-grained parallelism embedded within its innermost loops, and successively coarser-grained parallelism evident in the loops that surround them. Modern CPUs and compilers can effectively exploit many of these different levels of parallelism simultaneously, in order to achieve remarkable degrees of computational efficiency with relatively little specific intervention by the embedded programmer.

It is important to understand the several ways that modern computers exploit parallelism to see other specific things the embedded programmer can do [besides points (a) through (c) above] to help facilitate the parallelization process. Note that the subsections that follow are ordered from techniques best suited to exploit the finest-grained parallelism available (in the innermost loops), to those that are better suited for exploiting successively coarser and coarser grained parallelism.

1.4.1 Instruction pipelining and branch prediction

Even relatively simple (i.e., RISC) instructions may themselves generally be divided up into a number of smaller steps; for example, (a) fetch the instruction, (b) fetch the operands, (c) do the instruction, (d) write the results. Instruction pipelining is a technique for implementing parallelism on a CPU over each of these smaller steps, thus effectively keeping each corresponding part of the ALU or FPU busy doing useful work at each timestep. For example, at a clock cycle when instruction k is just starting with step (a) above, instruction $k-1$ can (simultaneously) be executing step (b), instruction $k-2$ can be executing step (c), and instruction $k-3$ can be finishing up with step (d). For this to work correctly, the calculations must be ordered in such a manner that a fine degree of parallelism is available, such that later commands don't try to fetch the results of earlier commands until they are actually available, which can take a few timesteps.

Branch prediction is a technique used to keep such instruction pipelines full even during the execution of conditional (if-then-else) statements. This is achieved by guessing (based on previous executions of each conditional) which branch the code is most likely to take, and proceeding assuming that the conditional will actually take that direction this time. If it does, the instruction pipeline remains full right through the conditional statement. If it does not, however, the tentative results of each calculation after the conditional must be discarded, before they are written back to memory, and the pipeline re-initialized with the instructions on the other branch of the conditional. Branch prediction is especially valuable in CISC systems, with complex instructions and thus relatively long pipelines, and in codes that frequently encounter conditionals. [Note that the code for handling branch predictions is generally inserted by the compiler, if the appropriate flags are set, and thus need not be written by the embedded programmer.] The overall time penalties associated with incorrect branch predictions may be kept small by (a) minimizing the number of conditional statements that are encountered by the numerical algorithm (eliminating such conditionals altogether from all but the outermost loops of the numerical algorithms used), and (b) using RISC processors, which have relatively short instruction pipelines.

1.4.2 Vectorization (SIMD)

As discussed in §1.1.3 and 1.1.4, the fixed-point and floating-point representations of real numbers that are useful in embedded applications are typically only 16 or 32 bits long, whereas the word length of high-performance CPU cores is 32 or 64 bits, and data bus and register sizes of modern CPUs and DSPs (see §1.5.4) can be even larger (e.g., 128 bits or more). Such an organization facilitates, where useful, the grouping of real numbers together as a **vector**, and performing quickly the same arithmetic operations on all elements of the vector simultaneously (or, nearly simultaneously), leveraging the extensive fine-grained parallelism often present in the innermost loops of substantial numerical algorithms. This basic idea goes by several names; in the early days of computing on Cray supercomputers (including the Cray-1, Cray X-MP, Cray-2, & Cray Y-MP), this process was called **vectorization**, and operated on very large vectors (with, e.g., 64 double-precision floats). The idea of vectorization went dormant in the mid 90's, but was revived for desktop and embedded processors, using much shorter vectors, under the general name of **SIMD** (single-instruction, multiple data), with different implementations appearing under various trademark names including MMX/SSE (Intel), 3DNow! (AMD), Altivec (Freescale), VMX (IBM), Velocity Engine (Apple), and, more recently, Neon and Helium (ARM).

1.4.3 Shared-memory multiprocessing

At the next coarser level of granularity of parallelism in numerical algorithms, multiple substantial tasks can often be identified that can be run completely independently from each other for a while [say, computing $O(10^3)$ or more floating-point operations (FLOPS) before having to share results with those of other tasks in order to

continue]. Such independent tasks are often found in the outermost loops of a code, and do not actually need to contain the same set of commands in order for the compiler to be able to parse them out and organize how to compute them in parallel; this setting is thus occasionally referred to as MIMD, to distinguish it from the SIMD setting required to parallelize innermost loops via vectorization, as discussed in §1.4.2.

The most straightforward way to leverage such coarse-grained parallelism is **multithreading**; that is, the spawning and running of multiple independent "threads" by a single numerical algorithm, each of which may run for a while on a different CPU core (as coordinated by the scheduler, as discussed further in §2.1) before pausing from time to time to synchronize its results with the other threads, but all of which ultimately access the same main memory. This setting is referred to as **shared-memory multiprocessing**, and may be coordinated directly by an embedded programmer from within a numerical code using **OpenMP** compiler directives, or in many cases can be efficiently coordinated by a good self-optimizing compiler.

As discussed in detail in §1.3, the use of high-speed cache memory (often, at multiple levels) has become essential for modern CPUs to reach their full potential, as CPUs are now typically much faster than the main memory that they access, but wide data paths allow large blocks of data to be retrieved from main memory in relatively little additional time (as compared with the time required to retrieve a single byte). In multi-core systems, L1 cache is typically dedicated to each core, L2 cache is dedicated to each CPU (shared amongst all cores on that CPU), and (often) L3 cache is shared amongst all CPUs, providing the gateway to the main memory. The challenge of maintaining cache coherence in multicore settings complicates the execution of complex numerical algorithms using shared-memory multiprocessing, in which data must be shared frequently between the different running threads, though in most applications the problem of maintaining cache coherence is taken care of by the MMU, with relatively little intervention required by the embedded programmer.

Most modern computers with a handful of CPU cores for shared-memory multiprocessing implement some sort of **symmetric multiprocessing** (SMP[15]), in which all compute cores have equal access to all memory and peripherals (usually via some arrangement of a data bus, address bus, and control bus), and may thus be treated essentially equally by the scheduler (see §2.1) for all tasks (i.e., no specific tasks are restricted to certain processors). Following this approach, two specific design paradigms simplify the organization:

(a) **homogeneous computing**, in which only one kind of CPU core is used, and
(b) **uniform memory access** (UMA), in which all cores have equal access to all sections of main memory.

Demands on peak computational performance in embedded systems continue to increase steadily, following the celebrated "Moore's Law" (that is, the observed doubling of the IC density in leading CPUs, and thus their performance, about once every 2 years). At the same time, the maximum clock speeds that CPUs can support is increasing only gradually in recent years, with higher clock speeds requiring higher voltages as well as increased power consumption to operate the CPU. Thus, embedded computers are now tending to include more and more CPU cores. Further, demands on computational performance in most applications are found to vary substantially over time, and power efficiency during the quiescent times is often just as important as peak computational performance during the active times. One approach to achieving an improved balance between *maximizing peak computational performance* and *minimizing time-averaged power consumption* is thus to implement **dynamic voltage and frequency scaling**, automatically reducing both the effective CPU clock speed as well as the voltage driving the CPU, in real time, when the recent average computational load is found to be relatively light[16].

When designing computers to meet even stricter requirements, however, both of the simplifying paradigms (a) and (b) above eventually become limiting factors, and must be relaxed in order to build systems with even greater peak computational performance, and with even lower average power consumption. Thus:

[15] The abbreviation SMP usually denotes symmetric multiprocessing, but is occasionally used more generally for shared-memory multiprocessing, which may or may not be symmetric. We recommend the former, more restrictive use, which is more common.

[16] In this setting, a relevant performance metric is FLOPS per MHz, in addition to peak FLOPS.

• The **heterogeneous computing** paradigm is now quite common, in which the embedded computer includes more than one type of CPU core (one with higher peak performance, and one with lower average power consumption), which may be selectively turned on and off. There are many different ways in which this general idea may be implemented; examples include ARM's big.LITTLE and DynamIQ technologies.
• An emerging design paradigm for embedded computers is **nonuniform memory access** (NUMA), in which each CPU (or, CPU cluster) is closely associated with only a subset of the main memory, and it takes substantially more time to read from or write to memory that is more closely associated with the other CPUs in the system [though all of the main memory shares a single large address space]. This approach was perfected in the field of high performance computing by Silicon Graphics (SGI) under the brand name NUMAlink, and (as of 2024) is only beginning to emerge in computers designed for embedded applications.

Note finally that, akin to branch prediction (see §1.4.1), **speculative execution** of independent threads of a multithreaded code following a conditional statement, or for which there is potentially stale data input, may be performed in the setting of shared-memory multiprocessing if sufficient computational resources are available, with speculative locks used to delay the write-back (or, the deletion) of the results of the speculative section of code until the conditional itself is evaluated, or the potentially stale data input has been verified as correct.

1.4.4 Distributed-memory multiprocessing

To solve even bigger problems, leveraging the coarsest-grained parallelism that can be identified in a numerical algorithm, many independent computers, each with their own dedicated memory, may work together over a fast network operating as a computer cluster. When large centralized computer clusters, and the codes running on them, are particularly well tuned for the coordinated distributed computation of very large individual jobs[17], this setting is often referred to as high performance computing (HPC).

Cluster-based "cloud" computing in the HPC setting is a very natural complement to "edge" computing for many large-scale real-time problems addressed by embedded sensors. Examples of interest include:

• the forecasting of the evolution of the track and intensity of hurricanes or forest fires and, simultaneously, the uncertainty quantification (UQ) related to such forecasts,
• the development of a single detailed map of a region, based on the information gathered from several independent mobile robots, each moving through and exploring different overlapping subregions, and each independently executing their own simultaneous localization and mapping (SLAM) algorithms, etc.

In such problems, a large computation needs to be performed on the cluster, fusing the Big Data being gathered, in real time, from numerous (often, heterogenous) sources (e.g., mobile robots), often using complex physics-based models. At the same time, based on the UQ performed on the cluster, the mobile robots often need to be redispatched intelligently to different subregions, a setting referred to as adaptive observation.

In the HPC setting, distributed computing leverages a fast and reliable communication network (see §4.1), such as[18] Ethernet or InfiniBand, between the independent computers making up the cluster. As opposed to shared-memory multiprocessing (§1.4.3), in which the MMU and a good self-optimizing compiler can often handle most if not all of the low-level details related to cache coherence and the coordination of distinct threads related to a certain job, in distributed-memory multiprocessing the necessary passing of data (aka messages) between the independent computers in the cluster must often be coordinated manually by the programmer from

[17] As opposed, for example, to the maintenance of transactional databases used for stock trades, ticket sales, large-scale search, social media, etc., with the cluster interacting simultaneously, and essentially independently, with a very large number of users.

[18] HPC is a very small market indeed, as compared to consumer electronics (largely supporting web surfing, video games, office productivity applications, etc). HPC today advances mostly by repurposing cutting-edge commercial off-the-shelf (COTS) electronics technologies developed for consumer electronics. In this setting, the possible deployment of Thunderbolt as a potential new technology for networking in HPC clusters is quite interesting.

within the numerical code, which is a rather tedious process. Some variant of the **Message Passing Interface** (MPI)[19] is generally used for this process in the distributed-memory setting, effectively solving (by hand) similar problems as those solved by the MMU (automatically) for maintaining cache coherence in the shared-memory setting, passing messages and blocking new computations only when necessary. Primitive operations used to coordinate message passing and computations in MPI-1 include

- **point-to-point** message passing (from one specific node to another),
- **one-to-all** message passing (aka **broadcast**),
- **all-to-one** message passing, together with an operation like summing (aka **reduce**),
- **all-to-all** message passing, for rearranging the data over the cluster, etc.

Such commands can be either blocking (halting a thread's execution until the command is completed) or non-blocking, or follow a ready-send protocol in which a send request can only be made after the corresponding receive request has been delivered. MPI-2 introduces certain additional operations, including

- **one-sided** put (write to remote memory), get (read from remote memory), and accululate (reduce) operations,
- the ability of an existing MPI process to **spawn** a new MPI process,
- the ability of one MPI process to communicate with an MPI process spawned by a different MPI process, etc.

Note that FT-MPI is a remarkable extension (plug-in) that adds significant fault tolerance capabilities to MPI; Open MPI also includes significant fault tolerance capabilities.

In the field of robotics, the problem of distributed computation is often referred to as **distributed control**. Distributed control systems generally implement several nested control loops on the individual mobile robots or machines (e.g., on an assembly line) involved. **Decentralized control systems** denote controllers that are primarily distributed on each robot or machine, with no central supervisory authority. **Centralized control systems**, in contrast, denote controllers that primarily operate on a central supervisory computer. Most practical control systems for multi-robot teams or multi-machine assembly line operations are some "hierarchical" hybrid between the two, with decentralized low-level/high-speed control feedback on the inner loops (e.g., coordinating the motion of an individual robot arm), coupled with centralized high-level coordination and fault management on the outer loops (adjusting the speed of the assembly line, etc). Mobile robots add the significant complication of very unreliable communication links, a challenge that requires significant care to address.

1.4.5 Summary: enabling the efficient parallel execution of codes

The reordering of the individual calculations within a numerical code, maximizing the temporal and spatial locality of the data needed for each calculation to be performed, and thus maximizing the effectiveness of all available levels of cache memory, is best achieved by using a modern self-optimizing compiler, with a high level of optimization selected, together with steps (a), (b), and (c) described in the introduction of §1.4.

Pipelining (with or without branch prediction) and SIMD vectorization, as discussed in §1.4.1 – 1.4.2, are both facilitated by the remarkable hardware of the modern CPU itself, together with the low-level opcodes used by good self-optimizing compilers to leverage this hardware. The use of both techniques can be activated by you, the embedded programmer, rather easily, simply by compiling your code with the appropriate compiler flags set to enable these features. With today's CPUs and compilers, it is generally not necessary for you to write code in assembler and deal with such low-level opcodes yourself, thus leaving you to attend to higher-level, more consequential issues. The efficient use of shared-memory multiprocessing (§1.4.3) sometimes takes a bit more work, leveraging OpenMP compiler directives to tune the default behavior generated by the compiler when necessary. The use of distributed-memory multiprocessing (§1.4.4) is, as of this writing, much harder, and must usually be coordinated manually by the user (often leveraging MPI), as introduced briefly above.

[19] Some HPC languages, like Coarray Fortran (which is implemented by G95), are beginning to implement coding constructs that that make higher-level parallel programming in the distributed memory setting significantly easier.

1.5 Microcontrollers (MCUs) and associated coprocessors

We now shift topics from the foundational ideas in modern computing, to the technologies that implement these ideas in embedded applications. At the heart of such implementations is a **microcontroller** (MCU[20,21]), which is an integrated circuit (IC) that fuses one or more CPU cores together with an interconnecting bus fabric, a (SRAM-, DRAM-, and/or Flash-based) memory subsystem, and a number of useful coprocessors, such as:

- arithmetic logic units (ALUs) for fast integer and fixed-point computations [§1.1.3],
- floating-point units (FPUs) for fast floating-point computations at specific precisions [§1.1.4],
- programmable interrupt controllers (PICs) to handle signals that trigger specific new actions [§1.5.2],
- general purpose timer/counter units [§1.5.5], and
- communication interfaces [§4] for connecting the MCU to a range of input/output (i/o) devices [§3],

as well as other application-specific integrated circuits (ASICs) useful for commonly-needed functions, such as:

a. dedicated hardware for transcendental function approximation [§1.5.3.1],
b. ring buffers for computing finite impulse response & infinite impulse response (FIR/IIR) filters [§1.5.3.2],
c. cyclic redundancy check (CRC) units, for performing fast detection of bit errors [§1.5.3.3],
d. random-number generators (RNGs) [§2.7], etc.

Some leading MCU families, and the CPUs that they embed, were surveyed briefly in §1.2. As indicated there, popular MCUs range from simple 8-bit devices, with just a few simple coprocessors, to remarkably efficient integrations of high-performance, low-power 32-bit or 64-bit CPUs with high-performance coprocessors (DSPs, GPUs, NPUs, FPGAs, CPLDs, PRUs, etc), together dozens of timers and other independent hardware communication subsystems (each function independently, in real time, *without* loading the main CPU core of the MCU, and often operate with direct memory access). Such useful coprocessors include, specifically,

e. quadrature encoder counters, for quantifying the (clockwise or anticlockwise) rotations of shafts,
f. pulse-width modulation (PWM) generators, for driving servos and ESCs,
g. UART, SPI, and I2C channels, for hooking up other ICs and (nearby) off-board peripherals,
h. CAN and RS485 controllers, for longer-distance communication over differential pairs of wires,
i. USB controllers, for communicating with desktop/laptop/tablet computers and associated peripherals,
j. digital-to-analog and analog-to-digital converters (DACs and ADCs), for interfacing with analog devices,
k. inter-IC sound (I2S) channels and/or serial audio interfaces (SAIs), for audio channels,
l. on-board or off-board oscillators, coin cell power backup, and real-time clocks (RTCs), for scheduled wakeup,
m. integrated op amps, for building analog filter circuits (low-pass, band-pass, notch, PID, lead/lag, ...)
n. memory controllers (e.g., FSMC and quad SPI channels), for hooking up additional memory, etc.

Loading the CPU, other serial comm protocols can be bit-banged using reconfigurable general-purpose input/outputs (GPIOs). An example modern MCU is the STM32G474, a block diagram of which is given in Figure 1.7. This MCU, built around an ARM Cortex M4F (a RISC CPU with 3-stage instruction pipeline, a modified Harvard architecture, and an FPU for single-precision floats), is implemented in the Beret family of boards introduced in §5, and integrates several coprocessors (indeed, in *all* 14 categories, a through n, mentioned above).

[20] In contrast (but, similar in many respects), a **microprocessor** (MPU) is an IC designed to form the heart of a desktop, laptop, or high-performance tablet computer, with hardware subsystems focused more on computational performance, graphics, and efficiently accessing a much larger memory and data storage footprint than a typical MCU.

[21] A third relevant category today is what is often called a **mobile processor** (MP), which is an IC that implements many of the same components as an MCU or MPU, but is tuned specifically for low-power operation, standby, and sleep. Modern MPs, which achieve remarkable flops/MHz, flops/mW, and (due to very large scale manufacturing, for use in smartphones) peak flops/$ ratios on real-world problems, are particularly well positioned for advanced high-level applications in embedded systems (performing vision-based feature recognition and SLAM, machine learning, etc.), as a complement to MCUs for handling the real-time low-level feedback required in motor control applications. Note that the dividing lines between MPs, MPUs, and MCUs continues to blur, and emphasizing the distinction between them is not necessarily productive moving forward.

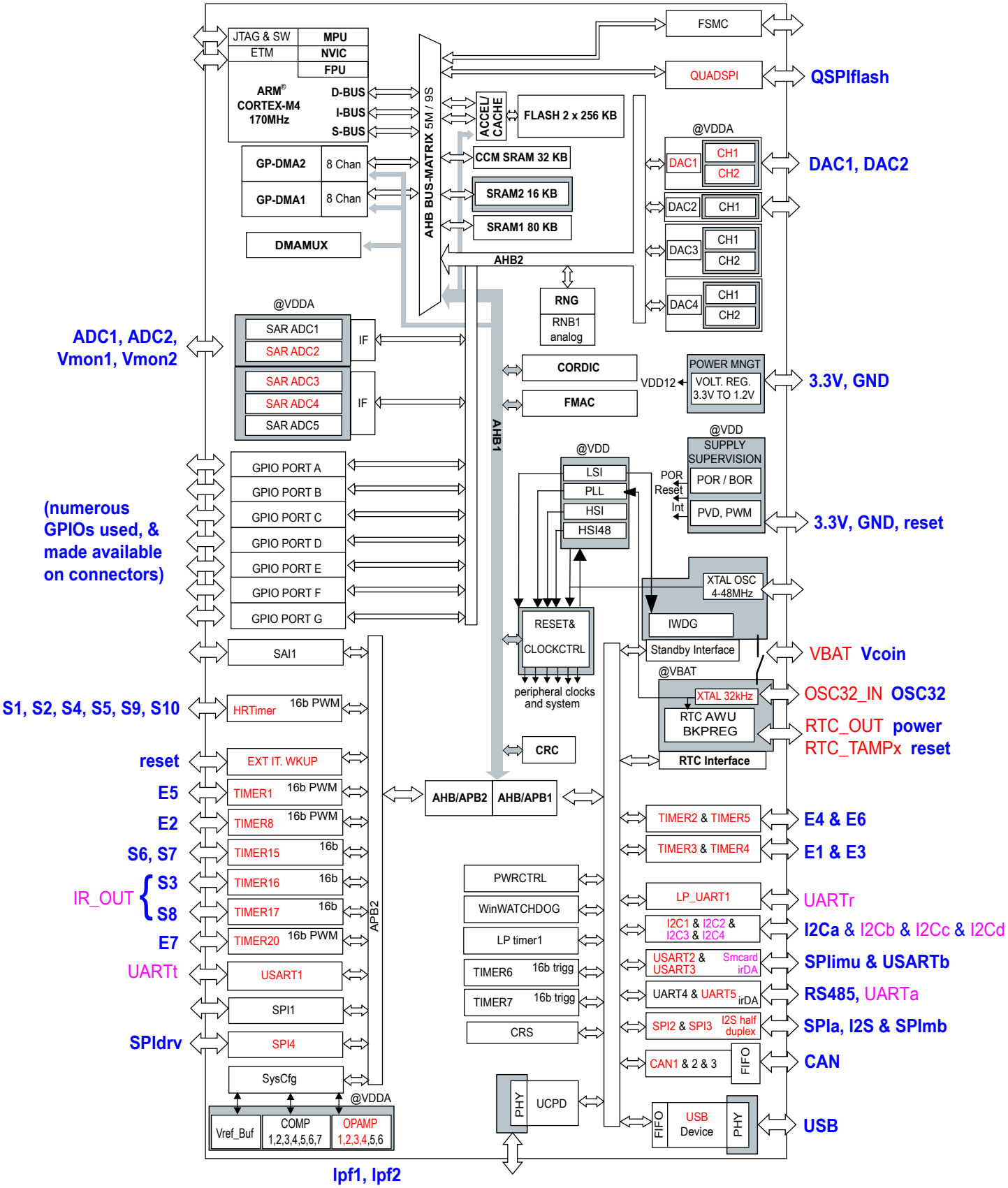


Figure 1.7: Block diagram of the STM32G474 MCU, with the hardware leveraged by the Berets (see §5) highlighted in color (see text). Image adapted from the STM32G474 datasheet, courtesy of STMicroelectronics.

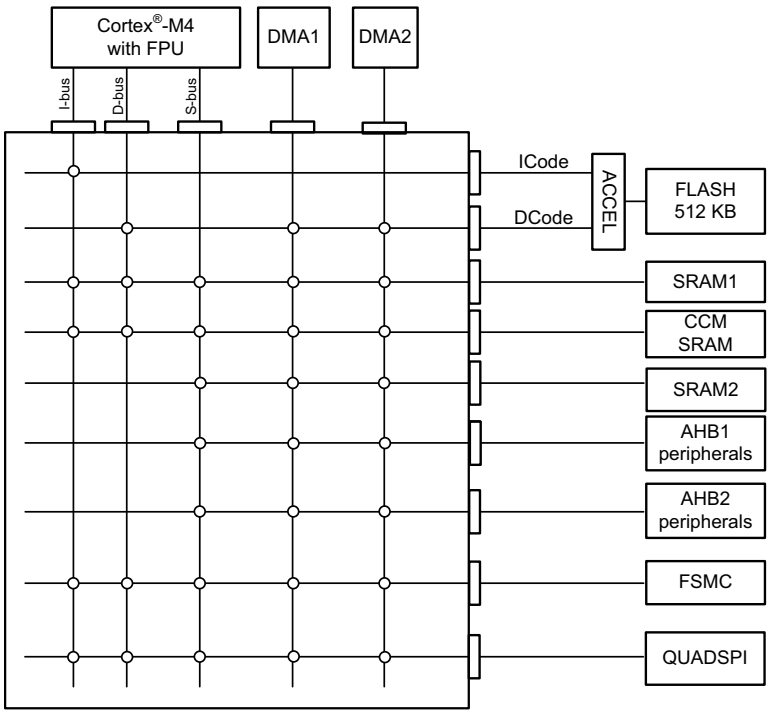


Figure 1.8: Bus matrix connections to/from the ARM Cortex M4 in the STM32G474; circles at intersections in the grid indicate an allowed connection from the corresponding master (at top) to the slave (at right). Image adapted from the STM32G474 datasheet, courtesy of STMicroelectronics; see also ST Application Note AN4031.

1.5.1 Busses, memory management, and direct memory access (DMA)

At the heart of a modern MCU is one or more CPU core(s). The complex fabric interconnecting these CPU core(s) within the MCU to the various coprocessors, to the cache-based memory and data storage system, and to the connected peripherals, is organized into a number of distinct busses, each with specific privileges, as illustrated for the STM32G474[22] in Figure 1.8. Most modern processors follow ARM's open standard Advanced Microcontroller Bus Architecture (AMBA) protocol, which includes the Advanced High-performance Bus (AHB), which is responsible for both the sending of an address to memory as well as the subsequent writing or reading of data or instructions to/from that memory address (via busses ranging from 64 bits to 1024 bits in width), and the lower-complexity Advanced Peripheral Bus (APB), which coordinates lower-bandwidth register and memory access by system peripherals (via a 32-bit bus).

Another essential aspect of modern CPUs is direct memory access (DMA), a feature that allows coprocessors and peripherals to read or update memory locations directly, without tying up the CPU as a choke point. In some implementations, DMA can also be used, without bogging down the CPU, to copy or move data from multiple memory locations into a single communication data stream, or to take data from a single data stream and distribute it to the appropriate memory locations, common processes referred to as scatter/gather I/O.

[22] In the ARM Cortex M4 CPU implemented in the STM32G474 MCU, there are three main busses, the ICode (instruction) interface, the DCode (data) interface, and the System interface, denoted **I-BUS**, **D-BUS**, and **S-BUS** in Figures 1.7 and 1.8.

1.5.2 Programmable interrupt controllers (PICs)

The CPU of an MCU often needs to wait for a trigger (for example, a clock pulse, or a signal from an external peripheral) before beginning a specific new action or computation. The CPU also needs to be able to handle various **exceptions** that occur when something unexpected happens (divide by zero, etc.). Such an event is generically known as an interrupt request (IRQ). There are many possible sources of IRQs, and at times they can arrive at the MCU in rapid succession, and thus need to be carefully prioritized and dealt with by the CPU accordingly. IRQs are handled by a dedicated unit on the CPU called[23] a programmable interrupt controller (PIC). The PIC assigns a priority and a block of code, called an interrupt service routine (ISR), for the CPU to deal with any given IRQ, if/when one is detected.

IRQs are denoted as maskable or non-maskable, which essentially distinguishes whether or not they may be ignored (at least, for the time being) by the ISR that is associated with that IRQ. Interrupts that deal with non-recoverable hardware errors, system reset/shutdown, etc., are often flagged as non-maskable interrupts (NMIs). Common interrupts generated and handled by user code, however, should generally NOT be flagged as NMIs, since NMIs hinder other normal operations (stack management, debugging, ...). Common interrupts that are time critical should instead be flagged as high priority maskable interrupts, and if such IRQs are missed by the system during testing, the behavior of the scheduler (see §2.1) should be adjusted to make certain that such high priority maskable interrupts are set up to be dealt with in a timely fashion.

1.5.3 Application specific integrated circuits (ASICs)

Application specific integrated circuits (ASICs) are dedicated coprocessors that are hard-wired for narrowly-defined purposes. As introduced previously, representative examples include transcendental function generators, ring buffers, cyclic redundancy check calculation units, random-number generators, etc. To illustrate, this section discusses various characteristics of these common types of ASICs. Note that the hardware implementing the timer / counter units discussed in §1.5.5, and the communication subsystems discussed in §1.5.6, may also be considered as ASICs.

1.5.3.1 CORDIC approximation of transcendental functions††

The efficient *software* approximation (to a selectable precision) of various transcendental functions is discussed in detail in §2.6. Specialized *hardware* suitable for approximating such functions even faster (again, to selectable precision), while offloading the CPU for other tasks, may also be implemented. The clever algorithm underlying such hardware is known as CORDIC (coordinate rotation digital computer), and is well suited for compact implementation on both ASICs and more general-purpose coprocessors (DSPs, FPGAs, etc).

We will discuss the CORDIC *algorithm* itself first, including its software and hardware implementations; *interpretation* of the convergence of the CORDIC algorithm is deferred to the end of this section [Figure 1.9].

The operations on $\{x,\, y,\, z\}$ that underlie all six forms of CORDIC are given, at each iteration, by

$$\begin{pmatrix} x \\ y \end{pmatrix}_{i+1} = \bar{K}_i \begin{pmatrix} 1 & -\mu\,\sigma_i f_i \\ \sigma_i f_i & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}_i , \tag{1.1a}$$

$$z_{i+1} = z_i - \sigma_i \alpha_i . \tag{1.1b}$$

That is, at each iteration, a "scaled rotation" is performed on $(x,\, y)$, and z is incremented by $\pm\alpha_i$. The parameters $\{\alpha_i,\, \bar{K}_i\}$ may be precalculated according to the various formula given in Table 1.11, the first few values

[23] The PIC on the ARM Cortex M4, as depicted in Figure 1.7, is called a nested vectored interrupt controller (NVIC).

†† This section, like later sections of this text marked with one or more daggers (†), is a bit harder than those around it, in proportion to the number of daggers used, and may be skipped upon first read without disrupting the continuity of the presentation.

circular	$\mu = 1,$	$\alpha_i = \text{atan}\,(1/2^i),$	$f_i = \tan(\alpha_i) = 1/2^i,$	$\bar{K}_i = 1/\sqrt{1 + 1/2^{2i}}$
linear	$\mu = 0,$	$\alpha_i = 1/2^i,$	$f_i = 1/2^i,$	$\bar{K}_i = 1$
hyperbolic	$\mu = -1,$	$\alpha_i = \text{atanh}(1/2^i),$	$f_i = \tanh(\alpha_i) = 1/2^i,$	$\bar{K}_i = 1/\sqrt{1 - 1/2^{2i}}$

Table 1.11: Formulas for μ, α_i, f_i, and $\bar{K}_i$, for (top) circular, (middle) linear, and (bottom) hyperbolic CORDIC rotations. Defining $K_i = \bar{K}_1\,\bar{K}_2\,\cdots\bar{K}_i$, the first few values of $\{\alpha_i,\ f_i,\ K_i\}$ are reported in Table 1.12.

circular $i = 0, 1, 2, \ldots$	$\alpha_i =$	0.78540,	0.46365,	0.24498,	0.12435,	0.06242,	0.03124,	0.01562,	...
	$f_i =$	1,	1/2,	1/4,	1/8,	1/16,	1/32,	1/64,	...
	$K_i =$	0.70711,	0.63246,	0.61357,	0.60883,	0.60765,	0.60735,	0.60728,	...
linear $i = 0, 1, 2, \ldots$	$\alpha_i =$	1,	1/2,	1/4,	1/8,	1/16,	1/32,	1/64,	...
	$f_i =$	1,	1/2,	1/4,	1/8,	1/16,	1/32,	1/64,	...
	$K_i =$	1,	1,	1,	1,	1,	1,	1,	...
hyperbolic $i' = 1, 2, 3 \ldots$	$\alpha_{i'} =$	0.54931,	0.25541,	0.12566,	0.06258,	0.06258,	0.03126,	0.01563,	...
	$f_{i'} =$	1/2,	1/4,	1/8,	1/16,	1/16,	1/32,	1/64,	...
	$K_{i'} =$	1.15470,	1.19257,	1.20200,	1.20435,	1.20671,	1.20730,	1.20745,	...

Table 1.12: Angles α_i, rotation factors f_i, and cumulative scale factors K_i of the CORDIC algorithm for (top) circular, (middle) linear, and (bottom) hyperbolic rotations. Note the two rotations in the hyperbolic case with $f_4 = f_5 = 1/16$ and $\alpha_4 = \alpha_5 = \text{atanh}(1/16) = 0.06258$ (see text). The full table of coefficients needed to apply CORDIC to achieve single-precision floating-point accuracy in all cases is computed in RR_cordic_init.m.

of which are listed in Table 1.12. The variable μ is just a sign bit, and is set as 1 for circular rotations, 0 for linear rotations, and -1 for hyperbolic rotations. The variable σ_i is also a sign bit, and is selected so that each iteration drives either z (for "rotation" mode) or y (for "vectoring" mode) towards zero as the iterations proceed. The factor f_i (and, in the case of linear rotations, the corresponding angle α_i) is halved at each iteration. In the case of circular and hyperbolic rotations, the first several angles α_i may be stored in small look up tables on the (hardware) CORDIC unit; once α_i becomes sufficiently small (at around iteration $i = 25$), the subsequent α_i are, again, simply halved at each iteration. Finally (important!), defining a *cumulative* scaling factor after n iterations such that $K_n = \bar{K}_1\,\bar{K}_2\,\cdots\bar{K}_n$, which may also be precalculated, multiplication by the individual scaling factors $\bar{K}_i$ in (1.1a) may be deferred, and the cumulative scaling factor K_n instead applied to (x, y) by the CORDIC preprocessing unit, either at the end, or at the beginning, of the n iterations performed.

A full floating-point implementation of the above algorithm is available at RR_cordic_core.m, with extensive preprocessors at RR_cordic.m and RR_cordic_derived.m; the heart of this code is listed in Algorithm 1.1. Note that such a *software* implementation of CORDIC is actually not very efficient as compared with the software approximation of transcendental functions using Chebyshev expansions, as discussed in §2.6. Where CORDIC becomes particularly useful, however, is its realization in specialized *hardware*, including both ASICs and high-performance coprocessors like DSPs and FPGAs (see §1.5.4), using fixed-point binary representations (see §1.1.3) of the real numbers involved. In this setting, the halving operations in Algorithm 1.1 may be accomplished quickly, with single bit shifts (to the right) of the corresponding fixed-point numbers. Further, one can implement the logic of the sign bits (that is, σ and μ) essentially for free. In such hardware, the computational cost of most of the iterations of Algorithm 1.1 in the case of circular or hyperbolic rotations is thus:

- three integer additions, during the generalized rotation of v (1,2) and the increment of v (3),
- one bit shift, during the update of f, and
- one table lookup [or, a second bit shift], to determine the next value of ang (that is, of α_i).

Algorithm 1.1: Main iteration of the CORDIC algorithm; full code available at RR_cordic_core.m.

```
for j=1:n                                                % perform n iterations
  % Compute sign of next rotation (mode=1 for "rotation", mode=2 for "vectoring")
  if mode==1, sigma=sign(v(3)); else, sigma=-sign(v(2)); end

  %%%% BELOW IS THE HEART OF THE CORDIC ALGORITHM %%%%
  v(1:2)=[1 -mu*sigma*f; sigma*f 1]*v(1:2);    % generalized rotation of v(1:2) by f
  v(3)  =v(3)-sigma*ang;                       % increment v(3)

  % update f (divide by 2) [factors {1/2^4, 1/2^13, 1/2^40} repeated in hyperbolic case]
  if mu>-1 || ((j~=4) && (j~=14) && (j~=42)), f=f/2; end
  % update ang from tables, or divide by 2
  if j+1<=cordic_tables.N && rot<3, ang=cordic_tables.ang(rot,j+1); else, ang=ang/2; end
end
% NOTE: the scaling of v by K, if necessary, is done in RR_cordic.m, not in this code.
```

An efficient hardware implementation of CORDIC is discussed in ST AN5325, which establishes that hardware implementations of CORDIC can have a very small silicon footprint, and in many cases of interest (for various transcendental functions, at specified levels of precision) can be substantially faster than computing these same functions using precompiled software libraries (see §2.6). Note in particular (in Table 1.12) that the angles reduce by about a factor of two at each iteration; convergence (i.e., the additional accuracy achieved per iteration) of this algorithm is thus said to be **linear**. Other iterative algorithms we will encounter later have substantially faster convergence; the key to the success of CORDIC is its remarkably simplicity, as itemized above.

In the remainder of this section, we turn to the *interpretation* of what the CORDIC iterations defined above accomplish (Figure 1.9). As mentioned previously, there are $3 \cdot 2 = 6$ forms of the CORDIC algorithm, with:

- three different types of rotations: circular ($\mu = 1$), linear ($\mu = 0$), or hyperbolic ($\mu = -1$) [see Table 1.11], and
- two different modes for determining σ_i:

rotation mode, which takes $\sigma_i = \text{sign}(z_i)$, eventually driving $z_n \to 0$ upon convergence, or (1.2a)

vectoring mode, which takes $\sigma_i = -\text{sign}(y_i)$, eventually driving $y_n \to 0$ upon convergence. (1.2b)

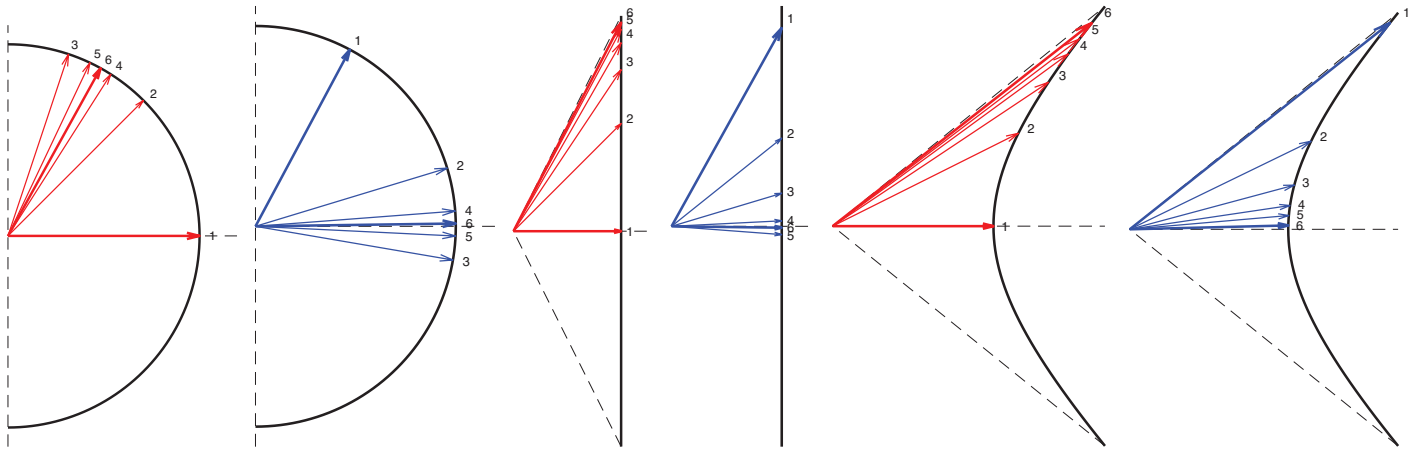


Figure 1.9: Geometric interpretation of the (successively smaller and smaller) rotations of the iterative CORDIC algorithm developed in §1.5.3.1, for (a, b) circular, (c, d) linear, and (e, f) hyperbolic rotations, illustrating both (a, c, e) "rotation" mode, which performs a generalized rotation of the vector (x_0, y_0) [illustrated here for $(x_0, y_0) = (1, 0)$] by the angle z_0, and (b, d, f) "vectoring" mode, which rotates the vector (x_0, y_0) to the positive x axis, while incrementing z_0 by the angle Δz required for such a rotation. Code at RR_cordic_viz.m.

Interpretation in the case of circular rotations.
For circular rotations ($\mu = 1$), noting that $\cos^2 x + \sin^2 x = 1$ and $\tan x = \sin x / \cos x$, and thus

$$\cos\alpha_i = 1/\sqrt{1+\tan^2\alpha_i} = 1/\sqrt{1+2^{-2i}} = \bar{K}_i \quad \text{and} \quad \sin\alpha_i = \tan\alpha_i/\sqrt{1+\tan^2\alpha_i} = \bar{K}_i\, f_i,$$

the rotation in (1.1a) may be written [note: the scaling by $\bar{K}_i$ is deferred in the code]

$$\begin{pmatrix} x \\ y \end{pmatrix}_{i+1} = G_i \begin{pmatrix} x \\ y \end{pmatrix}_i \quad \text{where} \quad G_i = \bar{K}_i \begin{pmatrix} 1 & -\sigma_i f_i \\ \sigma_i f_i & 1 \end{pmatrix} = \begin{pmatrix} \cos(\sigma_i\alpha_i) & -\sin(\sigma_i\alpha_i) \\ \sin(\sigma_i\alpha_i) & \cos(\sigma_i\alpha_i) \end{pmatrix}; \tag{1.3}$$

this is called a Givens rotation, and corresponds to an anticlockwise rotation of $(x, y)_i$ by the angle $(\sigma_i\alpha_i)$ at each iteration. Of course, successive Givens rotations accumulate; denoting $\phi_3 = \phi_2 + \phi_1$, $c_i = \cos(\phi_i)$, $s_i = \sin(\phi_i)$, and applying the identities $c_2c_1 - s_2s_1 = c_3$ and $s_2c_1 + c_2s_1 = s_3$, this may be verified as follows:

$$\begin{pmatrix} c_2 & -s_2 \\ s_2 & c_2 \end{pmatrix} \begin{pmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{pmatrix} = \begin{pmatrix} c_2c_1 - s_2s_1 & -c_2s_1 - s_2c_1 \\ s_2c_1 + c_2s_1 & -s_2s_1 + c_2c_1 \end{pmatrix} = \begin{pmatrix} c_3 & -s_3 \\ s_3 & c_3 \end{pmatrix}. \tag{1.4}$$

Thus, successive applications of (1.3) result in a total Givens rotation of the original $(x_0,\, y_0)$ vector by $\alpha = \sum_{i=0}^{n} \sigma_i\alpha_i$. Note that the α_i are scaled by a factor of 0.5, or slightly larger, at each iteration; as a result, for large n and by appropriate selection of the σ_i, total rotations anywhere in the range $-\alpha_{\text{max}} \le \alpha \le \alpha_{\text{max}}$ are possible, where $\alpha_{\text{max}} = \sum_{i=0}^{n} |\sigma_i\alpha_i| = \sum_{i=0}^{n} \alpha_i = 1.743287$ (that is, a bit over $\pi/2$). Thus:

• Using rotation mode (1.2a), selecting $\sigma_i = \text{sign}(z_i)$ at each iteration so that $z_n \to 0$ for large n, a total Givens rotation of $-\alpha_{\text{max}} \le z_0 \le \alpha_{\text{max}}$ radians [and a cumulative scaling of K_n^{-1}] is applied to the $(x_0,\, y_0)$ vector [see Table 1.13]. As a special case, defining $(x_0,\, y_0) = (K_n,\, 0)$, we have $(x_n,\, y_n) \to (\cos z_0,\, \sin z_0)$ in this mode.

• Using vectoring mode (1.2b), selecting $\sigma_i = -\text{sign}(y_i)$ at each iteration, the original $(x_0,\, y_0)$ vector is rotated [if K_n is applied] along a curve of constant $x^2 + y^2$ (that is, along a curve of constant radius from the origin) such that $y_n \to 0$, while the increments of z_i in (1.1b) again keep track of the total rotation performed in the process of rotating the vector $(x_0,\, y_0)$ to $(x_n,\, 0)$, so that $(x_n,\, z_n) \to (K_n^{-1}\sqrt{x_0^2 + y_0^2},\, z_0 + \text{atan}\,(y_0/x_0))$.

Interpretation in the case of hyperbolic rotations.
For hyperbolic rotations ($\mu = -1$), noting that $\cosh^2 x - \sinh^2 x = 1$ and $\tanh x = \sinh x / \cosh x$ and thus

$$\cosh\alpha_i = 1/\sqrt{1-\tanh^2\alpha_i} = 1/\sqrt{1-2^{-2i}} = \bar{K}_i \quad \text{and} \quad \sinh\alpha_i = \tanh\alpha_i/\sqrt{1-\tanh^2\alpha_i} = \bar{K}_i\, f_i,$$

the transformation in (1.1a) may be written [note: the scaling by $\bar{K}_i$ is deferred in the code]

$$\begin{pmatrix} x \\ y \end{pmatrix}_{i+1} = H_i \begin{pmatrix} x \\ y \end{pmatrix}_i \quad \text{where} \quad H_i = \bar{K}_i \begin{pmatrix} 1 & \sigma_i f_i \\ \sigma_i f_i & 1 \end{pmatrix} = \begin{pmatrix} \cosh(\sigma_i\alpha_i) & \sinh(\sigma_i\alpha_i) \\ \sinh(\sigma_i\alpha_i) & \cosh(\sigma_i\alpha_i) \end{pmatrix}. \tag{1.5}$$

This transformation is called a "hyperbolic rotation". Successive transformations by H_i also accumulate; denoting $\phi_3 = \phi_2 + \phi_1$, $C_i = \cosh(\phi_i)$, $S_i = \sinh(\phi_i)$, and applying the identities $C_2C_1 + S_2S_1 = C_3$ and $S_2C_1 + C_2S_1 = S_3$, this may be verified as follows [cf. (1.4)]:

$$\begin{pmatrix} C_2 & S_2 \\ S_2 & C_2 \end{pmatrix} \begin{pmatrix} C_1 & S_1 \\ S_1 & C_1 \end{pmatrix} = \begin{pmatrix} C_2C_1 + S_2S_1 & C_2S_1 + S_2C_1 \\ S_2C_1 + C_2S_1 & S_2S_1 + C_2C_1 \end{pmatrix} = \begin{pmatrix} C_3 & S_3 \\ S_3 & C_3 \end{pmatrix}. \tag{1.6}$$

Thus, successive applications of (1.5) result again in a total rotation of the $(x_0,\, y_0)$ vector by $\alpha = \sum_{i=0}^{n} \sigma_i\alpha_i$. In contrast with the circular case, the α_i in the hyperbolic case are scaled by a factor of 0.5, or slightly *smaller*, as

	rotation mode ($z_n \to 0$)	vectoring mode ($y_n \to 0$)
circular ($\mu = 1$)	$\begin{pmatrix} x_n \\ y_n \end{pmatrix} \to K_n^{-1} \begin{pmatrix} \cos z_0 & -\sin z_0 \\ \sin z_0 & \cos z_0 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$	$\begin{pmatrix} x_n \\ z_n \end{pmatrix} \to \begin{pmatrix} K_n^{-1}\sqrt{x_0^2 + y_0^2} \\ z_0 + \text{atan}\,(y_0/x_0) \end{pmatrix}$
linear ($\mu = 0$)	$\begin{pmatrix} x_n \\ y_n \end{pmatrix} \to \begin{pmatrix} x_0 \\ y_0 + z_0\, x_0 \end{pmatrix}$	$\begin{pmatrix} x_n \\ z_n \end{pmatrix} \to \begin{pmatrix} x_0 \\ z_0 + y_0/x_0 \end{pmatrix}$
hyperbolic ($\mu = -1$)	$\begin{pmatrix} x_n \\ y_n \end{pmatrix} \to K_n^{-1} \begin{pmatrix} \cosh z_0 & \sinh z_0 \\ \sinh z_0 & \cosh z_0 \end{pmatrix} \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}$	$\begin{pmatrix} x_n \\ z_n \end{pmatrix} \to \begin{pmatrix} K_n^{-1}\sqrt{x_0^2 - y_0^2} \\ z_0 + \text{atanh}(y_0/x_0) \end{pmatrix}$

Table 1.13: Convergence of the CORDIC algorithm for large n. Leveraging various identities, several derived functions may also be determined, as implemented in RR_cordic.m and RR_cordic_derived.m.

i is increased. Thus, in order to assure that *all* angles over a continuous range can be reached by a set of successive rotations, the typical approach used is to do two rotations associated with the angles $\alpha = \text{atanh}(1/2^4)$, $\text{atanh}(1/2^{13})$, and $\text{atanh}(1/2^{40})$ [see, e.g., Table 1.12]. With three such double-rotations built into the algorithm, it may be shown that, for large n and by appropriate selection of the σ_i, total rotations anywhere in the range $-\alpha_{\text{max}} \le \alpha \le \alpha_{\text{max}}$ are possible, where now $\alpha_{\text{max}} = \sum_{i=0}^{n} \alpha_i = 1.118173$. Thus:

• Using rotation mode (1.2a), selecting $\sigma_i = \text{sign}(z_i)$ at each iteration so that $z_n \to 0$ for large n, a total generalized rotation of $-\alpha_{\text{max}} \le z_0 \le \alpha_{\text{max}}$ radians [and a cumulative scaling of K_n^{-1}] is applied to the $(x_0,\, y_0)$ vector [see Table 1.13]. As a special case, defining $(x_0,\, y_0) = (K_n,\, 0)$, we have $(x_n,\, y_n) \to (\cosh z_0,\, \sinh z_0)$.

• Using vectoring mode (1.2b), selecting $\sigma_i = -\text{sign}(y_i)$ at each iteration, the original $(x_0,\, y_0)$ vector is rotated [if K_n is applied] along a curve of constant $x^2 - y^2$ such that $y_n \to 0$, while the increments of z_i in (1.1b) again keep track of the total rotation performed in the process of rotating the vector $(x_0,\, y_0)$ to $(x_n,\, 0)$, so that $(x_n,\, z_n) \to (K_n^{-1}\sqrt{x_0^2 - y_0^2},\, z_0 + \text{atanh}(y_0/x_0))$.

Interpretation in the case of linear rotations.
Finally, for linear rotations ($\mu = 0$), the transformation in (1.1a) may be written

$$\begin{pmatrix} x \\ y \end{pmatrix}_{i+1} = J \begin{pmatrix} x \\ y \end{pmatrix}_i \quad \text{where} \quad J = \begin{pmatrix} 1 & 0 \\ \sigma_i f_i & 1 \end{pmatrix}. \tag{1.7}$$

Again, successive transformations by J accumulate, which may be verified as follows:

$$\begin{pmatrix} 1 & 0 \\ f_2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ f_1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ f_2 + f_1 & 1 \end{pmatrix} \tag{1.8}$$

Thus, successive applications of (1.7) result in a translation of y_0 by $\sum_{i=0}^{n} \sigma_i f_i x_0$ (note that the x_i remain constant, due to the first row of J). The f_i in the linear case are exactly halved at each iteration, assuring convergence, for large n and by appropriate selection of the σ_i, of total translations anywhere in the range $-\Delta y_{\text{max}} \le \Delta y \le \Delta y_{\text{max}}$, where $\Delta y_{\text{max}} = \sum_{i=0}^{n} |\sigma_i f_i x_0| = c\,|x_0|$, where we have initialized $\alpha_0 = f_0 = 1$ (see Table 1.12), so that $c = 2$ (but other choices are certainly possible). Thus:

• Using rotation mode (1.2a), selecting $\sigma_i = \text{sign}(z_i)$ at each iteration so that $z_n \to 0$ for large n, a total translation of $\Delta y = z_0\, x_0$ is applied to y_0 [see Table 1.13]. As a special case, defining $y_0 = 0$, we have $y_n \to z_0\, x_0$.

• Using vectoring mode (1.2b), selecting $\sigma_i = -\text{sign}(y_i)$ at each iteration, the original $(x_0,\, y_0)$ vector is rotated along a line of constant x, such that $y_n \to 0$. Noting that $\Delta y = z_0 x_0$ in rotation mode, it is seen that vectoring mode is again its complement, with $\Delta z = y_0/x_0$ [see Table 1.13].

In practice, linear mode is useful for approximating multiply/accumulate and divide/accumulate operations on very simple hardware that is only capable of integer addition and bit shifts.

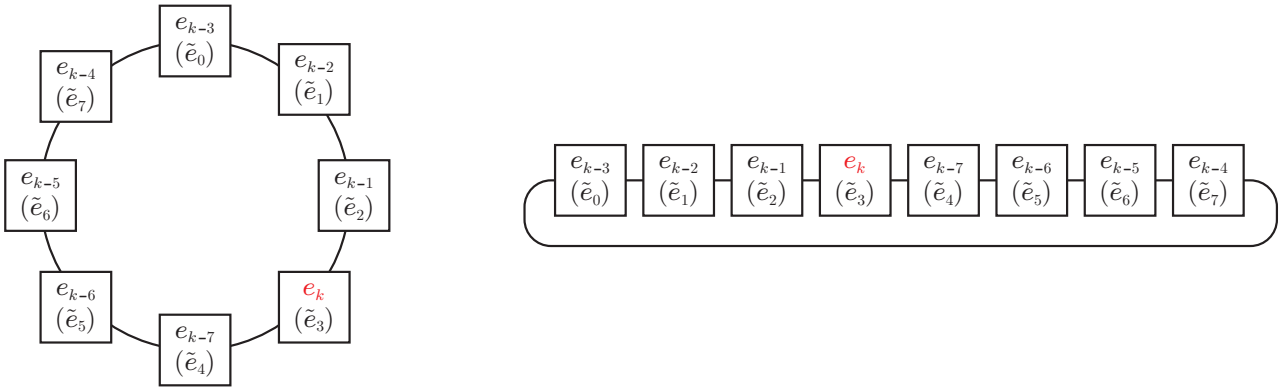


Figure 1.10: A ring buffer of the DT signal e_k and its 7 most recent tap delays at timestep $k = 11$: (left) as laid out conceptually, as a ring, and (right) as laid out in memory. At timestep $k = 12$, the next value, e_{12}, replaces the value in memory location $\tilde{e}_4$, the counter k is incremented by 1, and the other existing values of $\tilde{e}$ stay put.

1.5.3.2 Ring buffers

Many of the essential operations that an embedded controller needs to implement are linear discrete-time (DT) difference equations (see §8.3.3) that may be written as **finite impulse response** (FIR) filters of the form

$$u_k = b_0\, e_k + b_1\, e_{k-1} + \ldots + b_n\, e_{k-n}, \tag{1.9a}$$

or **infinite impulse response** (IIR) filters of the form

$$u_k = -a_1\, u_{k-1} - \ldots - a_m\, u_{k-m} + b_0\, e_k + b_1\, e_{k-1} + \ldots + b_n\, e_{k-n}. \tag{1.9b}$$

To perform such computations quickly, in addition to fast access (see §1.3) to the (fixed) a_i and b_i coefficients, fast access to current and recent values (aka **tap delays**) of the DT signals e_k and (in the case of IIR filters) u_k are needed. Instead of shifting all of these most recent values in memory at every timestep, a much faster approach is to use a ring buffer (aka circular buffer), such as that illustrated in Figure 1.10 (with $r = 8$ elements). With this approach, at each timestep k, the most recent value of e_k is stored in memory location $\tilde{e}_{\text{mod}(k,r)}$ [that is, within a ring buffer with $r \geq n$ memory locations allocated] using modular arithmetic, and u_k is given by:

$$\tilde{u}_{\text{mod}(k,r)} = b_0\, \tilde{e}_{\text{mod}(k,r)} + b_1\, \tilde{e}_{\text{mod}(k-1,r)} + \ldots + b_n\, \tilde{e}_{\text{mod}(k-n,r)} \qquad \text{or} \tag{1.10a}$$

$$\tilde{u}_{\text{mod}(k,r)} = -a_1\, \tilde{u}_{\text{mod}(k-1,r)} - \ldots - a_m\, \tilde{u}_{\text{mod}(k-m,r)} + b_0\, \tilde{e}_{\text{mod}(k,r)} + b_1\, \tilde{e}_{\text{mod}(k-1,r)} + \ldots + b_n\, \tilde{e}_{\text{mod}(k-n,r)}. \tag{1.10b}$$

With this approach, it is unnecessary to shift each of the saved values of e and u in memory by one location at each timestep, instead just advancing the index k used to reference these values in their (fixed, until replaced) locations in the ring buffers, and using this index (and reduced values of it, like $k - j$) in a modulo fashion.

FIR filters (1.10a) and IIR filters (1.10b) are needed so often in embedded computing that many modern CPU cores targeting applications in robotics and cyberphysical systems include specialized hardware or software implementations of both the ring buffers themselves (with the required mod command on the indices handled automatically) together with the additional low-level multiply/add circuitry or code required to implement such filters remarkably quickly, without significantly burdening the available CPU core(s).

In many DT filters, dubbed **strictly causal** (see §8.3.3.2), $b_0 = 0$. In such problems, (1.10a) or (1.10b) can simply be calculated between timestep $k - 1$ and timestep k.

In the case of **semi-causal** filters, however, $b_0 \neq 0$. In such problems, the strictly causal part of the RHS of (1.10a) or (1.10b) [which may involve a substantial number computations if n or m is large] can still be calculated between timestep $k - 1$ and timestep k. As soon as the new value of e_k becomes available, the RHS can then be

updated by adding $b_0 \cdot e_k$, and then the result may be applied directly on the output as u_k, thus applying the output u_k very very soon after the input e_k is received, though not quite instantaneously.

Ring buffers, as described above, can be implemented in dedicated hardware, such as ST's FMAC units (see ST AN5305), or in software, such as when using ARM Helium. Performing such computations in ASIC hardware on the MCU has the obvious advantage of offloading the CPU core; however, the size (and, the associated capabilities) of such ASICs needs to be decided upon when the (general purpose) MCU is designed, when the demands of the ultimate application are largely unknown. Performing such computations with streamlined software constructs on the CPU core of the MCU leads to a much more scalable solution (from small filters to large) to better fit the demands of the end application. Thus, ring buffers for MCUs targeting a narrow range of applications are most efficiently implemented on appropriately-sized ASICs; for general-purpose MCUs targeting a more broad range of applications, software-based solutions might be preferred.

1.5.3.3 Cyclic redundancy check calculation units

The use of parity bits to detect and/or correct occasional bit errors in memory and communication systems was introduced in §1.1.5. For this purpose, **linear binary codes** (**LBCs**) are designed for vectors in $\mathbf{F}_2^n$, with one bit in each of n elements, each considered as a **finite field** $\mathbf{F}_2 = \{0, 1\}$, with the fundamental operations of addition ($+$, aka XOR) and multiplication ($\cdot$, aka AND) **closed** (i.e., with no carry), as defined by:

$\mathbf{F}_2$:

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

An LBC is defined by two matrices, an $r \times n$ *parity-check matrix* H and an $n \times k$ *basis matrix* V, each with elements in $\mathbf{F}_2$, such that $H\,V = 0$. The use of an LBC to communicate data over a noisy channel is straightforward:

- **group** the data into *vectors* (blocks) of length k, with a single bit in each element;
- **code** each resulting data vector $\mathbf{a} \in \mathbf{F}_2^k$ into a longer codeword $\mathbf{w} \in \mathbf{F}_2^n$, with $n = k + r$, via $\mathbf{w} = V\mathbf{a}$;
- **transmit** the corresponding codeword $\mathbf{w}$ over the noisy channel;
- **receive** the (possibly slightly corrupted) message $\hat{\mathbf{w}} \in \mathbf{F}_2^n$ on the other end, and
- **decode** the message $\hat{\mathbf{w}}$ leveraging H, noting that $H\hat{\mathbf{w}} = 0$ corresponds to no bit errors, and find the most likely codeword $\mathbf{w}$ corresponding to the received message $\hat{\mathbf{w}}$, and the data $\mathbf{a}$ that generated it.

In such LBCs, r is the number of redundant bits, and each valid codeword differs in at least $d > 1$ bits, which facilitates error detection and/or correction. The identification of matrices $\{V_{[n,k,d]_2}, H_{[n,k,d]_2}\}$ that define efficient LBCs (that is, with maximum d for minimum r) is nontrivial, and is discussed at length in §12.

A **cyclic** LBC may be transformed to a form in which the $r \times n$ parity-check matrix $H^c_{[n,k,d]_2}$ and the $n \times k$ basis matrix $V^c_{[n,k,d]_2}$, where $n = r + k$ and $H^c_{[n,k,d]_2} V^c_{[n,k,d]_2} = 0$ (on $\mathbf{F}_2$), have the special form

$$H^c_{[n,k,d]_2} = \begin{pmatrix} h_k & h_{k-1} & \dots & h_0 & & & 0 \\ & h_k & h_{k-1} & \dots & h_0 & & \\ & & \ddots & \ddots & \ddots & \ddots & \\ 0 & & & h_k & h_{k-1} & \dots & h_0 \end{pmatrix}, \quad V^c_{[n,k,d]_2} = \begin{pmatrix} v_0 & & & 0 \\ v_1 & v_0 & & \\ \vdots & v_1 & \ddots & \\ v_r & \vdots & \ddots & v_0 \\ & v_r & \ddots & v_1 \\ & & \ddots & \vdots \\ 0 & & & v_r \end{pmatrix}, \tag{1.11}$$

where (for an LBC) $h_k = h_0 = v_0 = v_r = 1$. The nontrivial elements of these matrices are often summarized with a **basis polynomial** $v(z) = v_r z^r + \ldots + v_1 z + v_0$ and a corresponding **parity check polynomial** $h(z) = h_k z^k + \ldots + h_1 z + h_0$, defined mutually such that $h(z)\,v(z) = z^n - 1 = 0$, for a given number of redundant bits r and data bits k in each codeword of length $n = r + k$.

A **single error detecting** (SED) LBC takes $r = 1$ regardless of k; this redundant bit is called a **parity** bit, and is selected such that the $n = k + 1$ bits of each codeword add (on $\mathbf{F}_2$) to zero if even parity is used, or to one if odd parity is used; any 2 valid codewords in this LBC differ in at least $d = 2$ bits. Parity is checked again when decoding, and a single bit error can be detected but not corrected if the parity has changed. Of course, during both coding and decoding, parity is easily checked in hardware using a multiple-input XOR logic gate.

In an **error correcting code** (ECC), we need $r > 1$ and $d > 2$. The most broadly used ECCs in embedded computing, as they may be coded and decoded quickly in hardware, are the **single-error-correcting** (SEC) **binary Hamming codes** $[n, k, d] = [2^r - 1, 2^r - 1 - r, 3]$, as introduced in §1.1.5 and Figure 1.5. These include (for $r = 4$ through 8) $[15, 11, 3]$, $[31, 26, 3]$, $[63, 57, 3]$, $[127, 120, 3]$, $[255, 247, 3]$, and **shortened** versions thereof (with some data bits eliminated, thus giving codes in which k is a power of 2), including $[12, 8, 3]$, $[21, 16, 3]$, $[38, 32, 3]$, $[71, 64, 3]$, $[136, 128, 3]$; these codes take, respectively, $v_4(z) = z^4 + z + 1$, $v_5(z) = z^5 + z^2 + 1$, $v_6(z) = z^6 + z + 1$, $v_7(z) = z^7 + z^3 + 1$, $v_8(z) = z^8 + z^4 + z^3 + z^2 + 1$ (see Tables 12.4 and 12.6).

Binary Hamming codes

Many other codes with even greater error detection and correction capability may be transformed into cyclic form (see §12.7-12.8) and used similarly.

Adding a single overall parity check to binary Hamming codes generate the **single-error-correcting, double-error detecting** (SECDED) **extended binary Hamming codes** $[n, k, d] = [2^{r-1}, 2^{r-1} - r, 4]$, which include (for $r = 5$ through 9) $[16, 11, 4]$, $[32, 26, 4]$, $[64, 57, 4]$, $[128, 120, 4]$, $[256, 247, 4]$, and **shortened** versions thereof, including $[13, 8, 4]$, $[22, 16, 4]$, $[39, 32, 4]$, $[72, 64, 4]$, $[137, 128, 4]$. This

1.5.3.4 True random number generators

Random number generation are usually implemented in software (see §2.7)...

1.5.4 Coprocessors: DSPs, GPUs, NPUs, FPGAs, CPLDs, PRUs

More general-purpose coprocessors than ASICs, but with more specialized structure than CPUs, are sometimes called application-specific standard parts (**ASSPs**) or application-specific instruction-set processors (**ASIPs**). Many are perhaps best considered as some kind of System-On-Chip (**SoC**). Regardless of ambiguity in the literature on precisely what to call this general category of coprocessor, there are a handful of well-defined classes of coprocessors in this general category that are of principal importance in many modern MCUs, including:

DSP

GPU A GPU consists of multiple SIMD units with a large amount of associated memory.

NPU

FPGA

CPLD

PRU

1.5.5 Timer / counter units

PWM

Encoders

1.5.6 Dedicated communication hardware

The major wired and wireless communication protocols available today for embedded systems include PWM, UART, I2C, SPI, CAN, RS485, USB, Ethernet, Wifi, and Bluetooth, among others, as discussed further in §4.

Most MCUs implement dedicated hardware to support a number of these communication modes.
Typically, this hardware is capable of direct memory access (DMA).

1.5.7 Pin multiplexing

1.6 Single Board Computers (SBCs)

1.6.1 Subsystem integration: SiPs, PoPs, SoCs, SoMs, and CoMs

Integration of ICs:

- System-In-Package (SiP),
- Package-On-Package (PoP),
- System-On-Chip (SoC),
- System On Module (SoM),
- Computer On Module (CoM)

acronyms

1.6.2 Power management

i. ultra-low standby and sleep modes for battery-based operation, with various cues available for wakeup, etc.,

1.6.2.1 Sleep/wake modes, real-time clocks

IoT and low-power modes
Clock speed regulation

1.6.2.2 Switching regulators

efficiency vs. ripple rejection & voltage stability/accuracy

1.6.2.3 Switching regulators

1.6.2.4 Low-dropout (LDO) regulators

LDO
Power
Internet of Things

1.6.3 Case study: Raspberry Pi

Daughterboards
A detailed case study of a powerful class of daughterboards, dubbed Berets, is provided in §5.

model	connectivity	MCU	specs	PCB
UNO rev3		ATmega328P[8]	1x 20 MHz AVR	69 x 53
Mega 2560 rev3		ATmega2560[8]	1x 16 MHz AVR	102 x 53
MKR WAN 1300	LoRa	SAMD21[32]	1x 48 MHz M0+	68 x 25

Table 1.14: Some popular Arduino boards, and the principal specs of their Microchip MCUs; $()^N$ denotes an N-bit MCU.

	popular models	Berets	connectivity	MCU	specs	size
96Boards	Qualcomm RB3	Green	W5, BT5, GigE	SDA845 SoC[64]	4x 2.8 GHz A75 + 1.8 GHz 4x A55, DSP	85 x 54
	Shiratech Stinger96	Green	LTE	STM32MP157[32]	2x 800 MHz A7, 1x 209 MHz M4, GPU	85 x 54
ASUS	Tinkerboard	Rasp	W4, B4, GigE	RK3288[32]	4x 1.8 GHz A17, GPU	86 x 54
Banana Pi*	Banana Pi M64	Rasp	W5, BT4, GigE	Allwinner A64[64]	4x 1.2 GHz A53, GPU	92 x 60
	Banana Pi M4	Rasp	W5, BT4.2, GigE	RTD395[64]	4x A53, GPU	92 x 60
BeagleBoard*	BeagleBone AI	Black	W5, BT4.2, GigE	AM5729[32]	2x 1.5 GHz A15, 2x DSP, GPU, PRU, 4x M4	86 x 53
	BeagleBone Blue		W4, BT4.1	OSD3358 SiP[32]	1x 1 GHz A8, PRU, 1x M3	86 x 53
	BB Black Wireless	Black	W4, BT4.1	OSD3358 SiP[32]	1x 1 GHz A8, PRU, 1x M3	86 x 53
Hardkernel	ODROID XU4	Rasp†	GigE	Exynos 5422[32]	4x 2.1 GHz A15, 4x 1.4 GHz A7, GPU	83 x 58
Khadas	VIM3	Rasp	module	Amlogic A311D[64]	4x 2.2 GHz A73, 2x 1.8 GHz A53, GPU, NPU	82 x 58
Libre*	Le Potato	Rasp	LAN	Amlogic S905X[64]	4x 1.512 GHz A53, GPU	85 x 56
NanoPi*	NEO2	Red	GigE	Allwinner H5[64]	4x 1.5 GHz A53	40 x 40
NVIDIA	Jetson Nano Dev Kit	Rasp	GigE	(custom)[64]	4x 1.43 GHz A57, 128x Maxwell GPU	100 x 80
	Jetson Xavier NX	Rasp	W5, BT5, GigE	(custom)[64]	6x Carmel, 384x Volta GPU + 48x Tensor	103 x 90
Orange Pi*	Orange Pi 4B	Rasp	W5, BT5, GigE	RK3399[64]	2x 1.8GHz A72, 4x 1.4GHz A53, GPU	91 x 56
	OPi Zero LTS	Red	W4, LAN	AllWinner H2+[32]	4x 1.296 GHz A7, GPU	48 x 46
Pine64*	RockPro64	Rasp	GigE, module	RK3399[64]	2x 1.8GHz A72, 4x 1.4GHz A53, GPU	133 x 80
Radxa*	Rock Pi 4	Rasp	W5, BT5, GigE	RK3399[64]	2x 1.8GHz A72, 4x 1.4GHz A53, GPU	85 x 54
	Rock Pi S	Red	W4, BT4, LAN	RK3308[64]	4x 1.3 GHz A35	38 x 38
Raspberry Pi	Raspberry Pi 4B	Rasp	W5, BT5, GigE	BCM2711[64]	4x 1.5 GHz A72	85 x 56
	Raspberry Pi 3A+	Rasp	W5, BT4.2	BCM2837B0[64]	4x 1.4 GHz A53	65 × 56
	RPi Zero W	Rasp	W4, BT4.1	BCM2835[32]	1x 1 GHz ARM11	65 x 30

Table 1.15: Some popular linux-based Single-Board Computers (SBCs), and their principal specs; $()^N$ denotes an N-bit MCU. *SBCs from Banana Pi, BeagleBoard, Libre, NanoPi, Orange Pi, Pine64, and Radxa are Open hardware designs, facilitating derivative designs. Any SBC compatible with the Raspberry Beret is also compatible with the Red Beret. †A shifter shield is required to connect a Raspberry or Red Beret to the ODROID XU4. In the above list, LAN implies 100Base-T ethernet (i.e., 10x slower than GigE).

Chapter 2

Embedded programming

Contents

2.1 Multithreading and scheduling

A central element in the efficient use of limited computational resources for the coordination of complex electromechanical systems is *multithreading*; that is, the simultaneous running of many threads (a.k.a. *processes* or *tasks*), each at different rates and priorities, on a microcontroller with only a handful of CPU cores.

The coordination of multiple threads on a microcontroller is handled by the part of the OS called the *scheduler*. At any moment, a thread can be in one of three states: *executing* (a.k.a. *running*), *ready* (to run again, or to run some more...), or *waiting* (to be shifted back to the ready state). The component of the scheduler that shifts threads from the *ready list* to actually executing on the CPU is called the *dispatcher*.

Time-critical threads in an embedded setting generally require short periods of computation, called *CPU bursts*, followed by idle wait periods [during which file or bus i/o might be performed]. A request to the scheduler for a thread to begin a new CPU burst is initiated by some sort of *trigger* (a.k.a. *interrupt*) signal, such as

(a) a *timer*, which triggers requests for new CPU bursts on a thread at precisely predefined intervals Δt,
(b) a *delay*, which triggers such requests a set time after completion of previous CPU bursts on the same thread,
(c) a notification of the completion of a file or bus i/o (read or write) previously requested by the thread,
(d) a notification generated by the physical system, such as when a target temperature is reached, or
(e) a notification of new user input.

Other threads (e.g., video encoding) in an embedded setting are *CPU-bound* (requiring much longer computation time on the CPU), and may be worked on from time to time in the background, when the CPU bursts associated with all of the currently-triggered time-critical (higher priority) threads are complete. Note that:

(i) a running thread may shift back to the *waiting list* because its current CPU burst is complete, and the thread needs to wait for its next trigger (see above – in particular, a request for file or bus i/o is usually *blocking*, meaning that the corresponding thread is shifted back to the waiting list until the i/o is complete),
(ii) a running thread may *terminate*, simply because it finishes its task completely, or
(iii) a running thread may be *preempted* once the length of time (a.k.a. *quantum*) alotted to it is expired, or a higher-priority (time-critical) thread is triggered and needs to run, with the scheduler moving the preempted thread back to the ready list before the current set of computations in that thread complete, thus giving other threads a chance to run.

The *scheduling algorithm* (a.k.a. *scheduling policy*) is the set of rules used to determine the sequence that the threads in the ready list will be run, and the quantum that each thread is allowed to run before it is preempted to give CPU time to other threads. A scheduling algorithm must balance several competing objectives based on a limited amount of information regarding what might happen next, including:

(1) *respecting assigned priorities*: the user should be able to assign which threads are most important to complete in a timely fashion, and this preference should be enforced (thus giving "real-time" behavior – see §2.1.6),
(2) *responsiveness*: interactive threads should react quickly,
(3) *efficiency*: the CPU should be kept doing productive work all the time, minimizing the overhead involved in switching threads (see point iii above), and making maximum use of microcontroller I/O subunits (which are generally slow compared to the CPU), so that waiting on these I/O subunits does not hold up other threads,
(4) *fairness*: each thread of the same priority should receive about equal access to CPU time,
(5) *throughput*: the number of threads that accomplish something significant per second should be maximized,
(6) *avoiding starvation*: even low-priority threads should get a chance to run from time to time, and
(7) *graceful degradation*: as the CPU demands approach 100% or more, performance on all threads (particularly the lower-priority threads) should degrade gradually, and none of the threads should freeze.

Different compromises between these competing objectives are reached by different scheduling policies and different choices of the time quantums used, as illustrated by the following examples.

2.1.1 First-In, First-Out (FIFO) scheduling

To understand what a scheduler does, it is enlightening to consider first the simplest, non-preemptive First-In, First-Out (FIFO) scheduler. This scheduler simply waits for the CPU burst in the currently running thread to complete, and for the thread to enter the waiting list on its own (because, to continue, it needs to wait for a new trigger – like a timer interrupt, a notification of the completion of a blocking i/o request, etc). The dispatcher then shifts the oldest thread on the ready list over to begin executing on the CPU. Once any waiting thread receives the trigger it is waiting for, that thread is moved from the waiting list back the end of the ready list.

Though extremely simple, and effective at minimizing the overhead involved in switching threads, the FIFO approach reaches a relatively poor compromise between the seven competing objectives described in the previous section: it does not respect assigned priorities for time-critical tasks, interactive threads can be unresponsive when long CPU-bound tasks come up to run, etc. FIFO scheduling on its own is thus generally not recommended in practice (except in limited, controlled circumstances).

2.1.2 Round Robin (RR) scheduling

Round Robin (RR) scheduling amounts simply to a preemptive variant of FIFO scheduling, which improves upon the properties of the FIFO approach by limiting the quantum of time that any thread can tie up the CPU before the next thread gets a chance to run.

Using a large quantum, RR scheduling is effectively the same as FIFO scheduling, whereas using a smaller quantum results in more frequent switching between ready threads, which makes the overall system more responsive. However, reducing the quantum also increases percentage of time involved in switching threads (which typically takes a few ms), which reduces efficiency. For example, assuming a 3 ms switch time, a policy with 10 ms quantums spends $3/(10+3) = 23\%$ of the time switching, whereas a policy with 50 ms quantums spends $3/(50+3) = 5.7\%$ of the time switching. A compromise must thus be reached with an intermediate quantum (typically 10 to 50 ms) that provides both sufficient responsiveness and also reasonable efficiency.

2.1.3 Shortest Remaining Time First (SRTF) scheduling

Shortest Remaining Time First (SRTF) scheduling is a variant of RR scheduling that, based on historical averaging, estimates the upcoming CPU burst time associated with each thread on the ready list and, whenever the CPU becomes available, shifts the thread on the ready list with the shortest estimated CPU burst time over to begin executing on the CPU. A quantum is again used, so any thread with an actual CPU burst longer than the quantum is again preempted, and moved back to the ready list when its time is up.

A challenge with this approach is estimating future CPU burst times for any thread in the ready list, based only on previous CPU bursts in the same thread. One way to obtain such an estimate, E_n, of the n'th CPU burst time, B_n, is via an *exponential average* (a sort of IIR filter) given by $E_{n+1} = aB_n + (1-a)E_n$ for $n \geq 2$ with $0.1 \lesssim a \leq 1$, where we initialize $E_1 = 0$ and $E_2 = B_1$, with $n = 1$ corresponding to the first CPU burst.

An advantage of SRTF scheduling is that it tends to move interactive tasks (with, typically, short CPU bursts) to the head of the ready list (thus improving responsiveness) and, by running the shortest tasks first, it reduces the *mean response time* of the system (that is, the average time a thread spends between entering the ready list to the completion of its corresponding CPU burst), thus maximizing throughput.

2.1.4 Priority scheduling, dynamic priority adjustment, and multilevel schedulers

To allow the user to indicate a preference regarding which threads are most important to complete in a timely fashion (objective number 1 discussed above for schedulers for embedded systems), some sort of *priority schedul-*

ing is required. In the simplest, *static* form of such a policy, priorities are assigned (either *externally*, by the user, or *internally*, by the OS) for the life of each thread, and the threads on the ready list with the highest priority run first, with higher-priority threads preempting lower priority threads that may already be running as soon as they are moved to the ready list. In the event that multiple threads in the ready list are assigned the same priority, one of the simple policies described above (FIFO, RR, or SRTF) is used to break the tie; preempting may still be used, of course, to prevent individual threads from consuming the CPU for too long.

A clear advantage of priority-based approaches is that their behavior is easily predicted, and high-priority threads (e.g., those responsible for time-critical machine control loops, and interactive response) may be set to always run in a timely fashion, as needed. A challenge with such approaches is that some lower-priority threads may ultimately be completely starved of CPU time when the total requested CPU load exceeds 100%. To address this challenge, *dynamic* forms of this policy are sometimes used. Dynamic approaches occasionally boost the priority of some low-priority threads that haven't run in a while, thus making sure that they at least get a limited opportunity to run (this is sometimes referred to as *process aging*). Once such a boosted lower-priority thread runs for a full quantum, its priority is reduced back towards its original value. Often, longer quantums are implemented by the scheduler at lower priority levels, so with such dynamic approaches a thread can effectively settle into a priority level with a quantum that matches its typical CPU burst time, which is efficient. Note that such dynamic priority adjustments may be implemented in such a way as to never exceed the priorities assigned to the highest-priority ("real-time") threads.

It is common for a priority-based scheduler to group threads (distributed over about a hundred different priority levels) into a handful of *priority classes* [a.k.a. *priority queues*, for "real-time" (e.g., machine control) processes, system (a.k.a. kernel) processes, interactive processes, background processes, etc], each with a (possibly) different scheduling policy (like RR), and each with its own range of quantums implemented. Each of these priority classes themselves span well over a dozen priority levels, so priority-based scheduling algorithms (with or without dynamic priority adjustment) may still be used within each class. A *multilevel scheduler* may then choose to devote the CPU, when fully loaded, a certain maximum percentage of time to each class of processes, and to use a simpler priority-based scheduling policy within each class. The *Completely Fair Share* (CFS) scheduler implemented in modern Linux kernels is a general purpose multilevel scheduler implementing dynamic priority adjustment within a handful of priority classes, including a RR scheduler at the highest priority levels for "real-time" tasks.

2.1.5 Multicore: load balancing, processor affinity, & power management

SMP and HMP

Most modern embedded processors can actually run multiple threads at the same time, including:

• systems with multithreaded cores, which present themselves as two virtual cores to the scheduler, allowing multiple instructions [e.g., integer operations (IOPs) and floating-point operations (FLOPS)] to execute simultaneously on a single core, as long as they don't compete for the same resources;
• systems with multiple cores on one CPU, or with multiple CPUs, with or without shared memory caches but all with Uniform Memory Access (UMA) to all of the main memory, either:
- in a *symmetric multiprocessing* (SMP) arrangement, in which all cores are equivalent, or
- in a *heterogeneous multiprocessing* (HMP) arrangement, as in ARM's big.LITTLE and DynamIQ implementations, which combine high-performance cores (for computationally-intensive, time-critical tasks), and high-efficiency cores (for simpler, lower-priority tasks);
• systems with multiple CPUs in a Nonuniform Memory Access (NUMA) arrangement, in which each compute core has a certain portion of the main memory closely affiliated with it, and thus can reach some parts of the it faster than others (such systems may also be SMP or HMP) - embedded processors with large GPU-based computational subsystems, and those with dedicated "Neural Processing Units", are typical examples.

The same general considerations and scheduling policies discussed previously still apply in these settings, but now with the complex additional consideration of needing to manage the delicate question of which core should be used to run a particular thread next, and which cores can be run at reduced clock speeds, or powered down entirely, during relatively idle periods of time in order to save power.

These delicate questions need to be handled carefully by modern schedulers in order to balance computational throughput and power efficiency in the system, and very different solutions are needed for servers, laptops, cellphones, and microcontrollers for "real-time" control of embedded systems. Notably, balancing computational performance with power efficiency is becoming increasingly important in all types of computational platforms, and solutions originally developed for small battery-powered systems (cellphones) are working their way up to laptops and large server farms, which are increasingly limited by power considerations.

In multicore settings, the issues of *processor affinity* and *load balancing* must be addressed. That is, it is usually much more efficient to run a new CPU burst on the same core (or at least on the same CPU) that a thread ran on previously, because the memory cache corresponding to that core (or CPU) is probably already set up with much of the data that that thread needs to run again. However, sometimes threads still need to be shifted from one core to another in order to balance the load across multiple cores in the system, as the scheduler seeks to maintain its target balance between computational throughput and power efficiency. Hierarchical *scheduling domains* are often introduced in order to handle these questions, with lower-level schedulers handling each individual core, and higher-level schedulers occasionally moving threads from one core to another as necessary (i.e., whenever a given core becomes relatively overloaded, or underloaded, with tasks to complete). To improve the predictable performance of the most time-critical threads, including those that might share certain cached data, it is often beneficial to implement *hard processor affinity* for such threads, binding them permanently to specific "reserved" cores, while allowing the OS to manage the other threads that might come and go on the system (but possibly restricting the other major threads on the system from running on the reserved cores, thereby preventing them from interfering with the most time-critical processes).

Thankfully, the complex coupled problems of scheduling, load balancing, and power management for SMP and HMP multicore systems are generally taken care of by the OS, not by the embedded programmer, and the sophistication with which modern schedulers for multicore systems address these problems, to appropriately balance computational performance with power efficiency, is evolving rapidly. However, understanding generally how such schedulers work is essential for the embedded programmer, in order to select and use a scheduler

appropriately, and to tweak its behavior effectively (in particular, to set priorities correctly, and to use hard processor affinity where appropriate), in order to strike the desired balance between the seven objectives outlined previously: namely, to get sufficiently reliable "real-time" performance for high-priority time-critical tasks (see §2.1.6), sufficient responsiveness from interactive tasks, and efficient performance on all other threads that the computational system needs to manage, even as the computational system becomes fully loaded.

2.1.6 Characterizing "real time" application requirements

In §2.1, the #1 objective listed for a scheduler on a microcontroller is that the user should be able to assign which threads are most important to complete in a timely fashion, and that these preferences should somehow be enforced. In embedded systems, we need to define the importance of such preferences with precision. Consideration must first be given to the application itself. Embedded programmers often categorize controllers based on the consequences of not completing a task within a specified time constraint (a.k.a. deadline):

• *hard real-time* controllers are designed for systems in which a missed deadline may result in total system failure [an assembly line shuts down and needs to be physically repaired, a rocket blows up, ...];
• *firm real-time* controllers are designed for systems in which, after a missed deadline, the utility of a result is zero [a single part will be rejected (automatically) off an assembly line, a toy falls over, ...]; and
• *soft real-time* controllers are designed for systems in which, after a missed deadline, the utility of a result is reduced somewhat [an RC car is momentarily unresponsive to user input, a hamburger bun is slightly singed, ...].

In addition to specifying the relevant deadlines themselves, the above characterizations of the consequences of missed deadlines are valuable when deciding how to allocate limited computational resources to potentially complex electromechanical systems.

Hard real time requirements are actually somewhat rare in well-designed mechanical systems; examples might include the control of an unstable chain reaction, or a pacemaker for a human heart. In hard real-time systems, particularly those that are safety-critical, mathematical guarantees of no missed deadlines are often required. Guaranteeing such hard real-time behavior is generally only possible by applying a controller in a relatively isolated setting with simple (and, thus, highly predictable) bare-metal programming (see §2.2.1), without several other threads running simultaneously that might occasionally throw the timing off.

More often than not, however, threads running on embedded systems call for firm real-time and/or soft real-time behavior. In such systems, the priority-based preemptive scheduling strategies described above, as implemented by a well-designed OS (e.g., the PREEMPT_RT patch of the Linux kernel) and used properly by a careful programmer, are most often entirely sufficient.

2.2 Operating Systems (OSs)

2.2.1 Bare-metal programming

In this section and the two that follow, we outline the three fundamental programming paradigms for embedded systems, in order of simplicity.

Arduino
ladder logic
programmable logic controllers (PLCs) used in industrial control applications

2.2.2 Real-Time Operating Systems (RTOSs)

nuttx (posix compliant)

keil RTX
Real Time Executive for Multiprocessor Systems (RTEMS)
real time linux
more realtime linux
Case study: FreeRTOS

2.2.3 Linux

2.2.3.1 Embedded Linux distros

distros (distributions)
Debian (derivatives: Ubunto, Raspberry Pi OS).
Yocto. OpenWrt.
Commercial: Wind River Linux. Red Hat Embedded
Look for lightweight IoT version (but, man pages are useful...)
shells

2.2.3.2 Chmod

2.2.3.3 Makefiles

real time computing

2.2.4 Realizing hard real time with Linux: dual-kernel approaches vs. RTL

PREMPT-RT
NTP service

2.2.5 Android

2.2.6 Robot Operating System (ROS)

2.3 Programming languages

Many programming languages are growing in importance in different aspects of robotics, including CUDA for GPU programming, TinyML for machine learning,

2.3.1 C, C++

2.3.2 Python

2.3.3 Graphical programming environments

Scratch
Simulink
Labview

`ping 192.168.8.1`	measure speed of connection to machine with IP number `192.168.8.1`
`ssh 192.168.8.1`	securely open a shell on `192.168.8.1`
`echo $0`	show what kind of shell you are currently in
`uname -a`	show info about processor architecture, system hostname, and kernel version
`zsh` or `bash` ; `exit`	spawn & enter a new `zsh` or `bash` shell (inside current shell); exit this shell
`chsh -s /bin/zsh`	change your default shell to `zsh` (recommended, if it isn't already)
`pwd`	print the name of the current working directory
`ls -lah`	list all files in current directory, including ownership, privileges, and size
`mkdir foo`	make a new directory named `foo`
`cd foo` ; `cd ..`	change directory to `foo`; change back to parent directory
`touch bar`	create a new file named `bar` (or, just update its timestamp)
`echo 'hello' > bar`	create (or, erase and create) the file `bar`, and write "`hello`" to this file
`echo 'world' >> bar`	append "`world`" to the file `bar` (or, create and write to this file)
`man echo` ; (space) ; `q`	display detailed manual page (alternative to Google) for the command `echo`
`cat bar`	show contents of the file `bar` (all at once)
`less bar` ; (space) ; `q`	show contents of the file `bar` (pausing after each screenfull)
`head bar` ; `tail bar`	show the 10 lines at the head (or, the tail) of the file `bar`
(up arrow) ; (down arrow)	scroll up to recently executed commands; scroll down
`history`	show a list of recently executed commands
`hist` (tab)	complete (as far as possible) name of command(s) starting with "`hist`"
`rm bar`	remove (warning: permanently!) the file named `bar`
`rmdir foo`	remove directory `foo`, but only if it is empty
`rm -rf foo`	remove recursively the directory `foo` and all files contained in it (danger!!!)
`cp foo/bar* foo1/.`	copy all files starting with the letters `bar` in `foo` into the directory `foo1`
`cp -r foo foo1`	copy recursively everything in `foo` to the directory `foo1`
`mv bar foo/bar1`	move and rename the file `bar` as `bar1` inside the directory `foo`
`chmod 644 bar`	change mode (§2.2.3.2) of `bar` to read/write for owner, read for group & world
`chown foo1:foo bar`	change ownership of file `bar` to user `foo1` and group `foo`
`sudo rm bar`	do the command `rm bar` as superuser (danger!)
`su` ; `exit`	enter superuser mode for subsequent commands (danger!!!); exit su mode
`df -h`	report disk free space on the available filesystems
`du -sh foo`	report significant disk use within directory `foo`
`grep psfrag *.tex`	search files ending in `.tex` (in current directory) for the string "`psfrag`"
`top`	periodically report a list of all running threads, sorted by top CPU usage
`ps -ef`	report all running processes (once)
`ps -ef |grep kernel`	pipe output of `ps` to `grep`, to extract the lines with "`kernel`" in them
`file bar`	test `bar` to determine what type of file it is
`find bar`	scan current directory and all its children for filenames containing `bar`
`tar cvfz fb.tgz fb`	compress all contents of `fb` into a (compact) gzipped tarball `fb.tgz`
`scp fb.tgz bar:.`	securely copy `fb.tgz` to machine with name `bar` on local network
`tar xvf fb.tgz`	extract contents of `fb.tgz`, retaining its original directory structure
`alias l='ls -lah'`	use "`l`" as a shorthand alias for the command "`ls -lah`" in this shell
`env`	list all aliases and other environmental variables defined in this shell
`vim` ; `nano`	command-line text editors (see §2.4.1.2) available in all linux distros
`~/.bashrc`	initial run commands executed when a `bash` or `zsh` shell is spawned
`make foo`	run commands in Makefile (see §2.2.3.3) to make an executable `boo`

Table 2.1: Some essential unix/linux/mac commands (in zsh and bash). Explore! You'll find your way quickly...

2.4 Text editing & command-line programming versus IDEs

2.4.1 Command-line programming

2.4.1.1 Workflow: edit locally, sync files with SBC, compile, link, run, rinse, repeat

The mind-numbing repetitiveness of this process is sometimes humorously referred to as the shampoo algorithm; a possibly more efficient alternative, once you have everything set up correctly, is to use an IDE (see §2.4.2).

2.4.1.2 Text editors

text editors built in to modern linux implementations include vim and nano ...

GUI text editors built in to other modern operating systems include, notably, TextEdit (Mac) and Notepad++ (Windows) ...

Code editors that you can run on your Windows or Mac laptop or desktop include Visual Studio Code, Sublime Text, and Atom.

2.4.1.3 Command-line scp/sftp/rcp vs FTP Clients

SFTP

FileZilla

2.4.2 Programming in an Integrated Development Environment (IDE)

XCode

Eclipse,

STM32CubeIDE (for STM32 devices, based closely on Eclipse)
ARM Keil MDK (for ARM devices)
Visual Studio Code (Microsoft),

NetBeans
Code::Blocks
CodeLite
Qt Creator,

PyCharm (for Python),

MPLAB X (PIC, AVR)

2.4.2.1 Workflow: debug directly within the IDE

Case study: Eclipse

Version of Eclipse for STM32CubeIDE

2.5 Debuggable, maintainable, and portably fast coding styles

self-optimizing compilers

2.5.1 Platform-optimized libraries: BLAS, LAPack, FFTW

2.5.2 POSIX compliance

2.6 Software approximation of special functions

Most modern microprocessors have fast hardware and/or software libraries built in to compute many important functions; on some small microcontrollers (see §1.5), however, you must often be prepared to approximate such special functions efficiently yourself. With this section, we've got your back.

As mentioned previously, in embedded applications, we are primarily interested in half precision and single precision implementations, which form our focus here.

Significant attention has been put into developing efficient and accurate numerical approximations of important transcendental functions ($\sin, \cos, \tan, \text{asin}, \text{acos}, \text{atan}, \exp, \ln, \log_{10}, \log_2, \ldots$). A comprehensive treatment of this subject, which presents many of the commonly needed (complicated-to-derive, yet simply-to-use) formula, some of which are summarized below, is Hart (1978), *Computer Approximations*.

The following formula (which may be computed using single precision arithmetic) approximates $\cos(z)$ over the range $0 \le z \le \pi/2$ to about 3.2 decimal digits (appropriate for use in half precision applications):

$$c_1 = 0.99940307, \quad c_2 = -0.49558072, \quad c_3 = 0.03679168 \quad \Rightarrow \quad \cos(z) \approx c_1 + z^2(c_2 + c_3\, z^2), \tag{2.1a}$$

and the following formulae (which may be computed using double precision arithmetic) approximates $\cos(z)$ over the range $0 \le z \le \pi/2$ to about 7.3 decimal digits (appropriate for use in single precision applications):

$$\begin{aligned} &c_1 = 0.999999953464, \;\; c_2 = -0.4999999053455, \;\; c_3 = 0.0416635846769, \;\; c_4 = -0.0013853704264, \\ &c_5 = 0.00002315393167 \quad \Rightarrow \quad \cos(z) \approx c_1 + z^2(c_2 + z^2(c_3 + z^2(c_4 + c_5\, z^2))). \end{aligned} \tag{2.1b}$$

Note the tradeoff: the first approximation is simpler (smaller table of numbers and faster to compute, but less accurate), while the second is more complex (larger table of numbers and slower to compute, but more accurate). This tradeoff is evident in all such approximations. To extend the range to $-\infty \le x \le \infty$, note that

$$\cos(x) = \begin{cases} \cos(y) & \text{if } q = 0, \\ -\cos(\pi - y) & \text{if } q = 1, \\ -\cos(y - \pi) & \text{if } q = 2, \\ \cos(2\pi - y) & \text{if } q = 3, \end{cases} \qquad \text{where} \qquad \begin{aligned} &c = \lfloor x/(2\pi) \rfloor, \\ &y = x - 2\pi c, \;\; (\text{and thus } \; 0 \le y \le 2\pi), \\ &q = \lfloor y/(\pi/2) \rfloor \in \{0, 1, 2, 3\}, \end{aligned} \tag{2.1c}$$

where $\lfloor y \rfloor = \text{floor}(y)$ denotes the rounding of y *down* to the nearest integer, and $\cos(z)$ may be approximated (in any of the four cases) using (2.1a) or (2.1b). Applying (2.1c) in order to extend the approximation (2.1a) or (2.1b) to the larger range $-\infty \le x \le \infty$ is called **range reduction**. With the above formulae, $\cos(x)$ and

$$\sin(x) = \cos(x - \pi/2) \tag{2.2}$$

may be computed efficiently for half and single precision applications for any real x.

Similarly, the following formula (which may be computed using single precision arithmetic) approximates $\tan(z)$ over the range $0 \le z \le \pi/4$ to about 3.2 decimal digits (appropriate for half precision applications):

$$c_1 = -3.6112171, \quad c_2 = -4.6133253 \quad \Rightarrow \quad z_0 = 4z/\pi, \quad \tan(z) \approx c_1\, z_0/(c_2 + z_0^2), \tag{2.3a}$$

and the following formula (which may be computed using double precision arithmetic) approximates $\tan(z)$ over the range $0 \le z \le \pi/4$ to about 8.2 decimal digits (appropriate for single precision applications):

$$\begin{aligned} &c_1 = 211.849369664121, \quad c_2 = -12.5288887278448, \quad c_3 = 269.7350131214121, \\ &c_4 = -71.4145309347748 \quad \Rightarrow \quad z_0 = 4z/\pi, \quad \tan(z) \approx z_0\,(c_1 + c_2\, z_0^2)/(c_3 + z_0^2(c_4 + z_0^2)). \end{aligned} \tag{2.3b}$$

To extend the range of either approximation to $-\infty \le x \le \infty$, note that

$$\tan x = \begin{cases} \tan y & \text{if } o = 0, \\ 1/\tan(\pi/2 - y) & \text{if } o = 1, \\ -1/\tan(y - \pi/2) & \text{if } o = 2, \\ -\tan(\pi - y) & \text{if } o = 3, \\ \tan(y - \pi) & \text{if } o = 4, \\ 1/\tan(3\pi/2 - y) & \text{if } o = 5, \\ -1/\tan(y - 3\pi/2) & \text{if } o = 6, \\ -\tan(2\pi - y) & \text{if } o = 7, \end{cases} \quad \text{where} \quad \begin{array}{l} c = \lfloor x/(2\pi) \rfloor, \\ y = x - 2\pi c, \ \ (\text{and thus } \ 0 \le y \le 2\pi), \\ o = \lfloor y/(\pi/4) \rfloor \in \{0, 1, 2, 3, 4, 5, 6, 7\}. \end{array} \tag{2.3c}$$

The following formula (which may be computed using double precision arithmetic) approximates $\operatorname{atan} z$ over the range $0 \le z < \operatorname{atan}(\pi/12)$ to 6.6 decimal digits (appropriate for single precision applications):

$$c_1 = 1.6867629106, \ \ c_2 = 0.4378497304, \ \ c_3 = 1.6867633134 \ \ \Rightarrow \ \ \operatorname{atan} x \approx x\,(c_1 + x^2\, c_2)/(c_3 + x^2). \tag{2.4a}$$

To extend the range to $-\infty \le x \le \infty$, define $c = \tan(\pi/6)$, and apply any or all of the following identities

$$\begin{array}{ll} \operatorname{atan} x = -\operatorname{atan}(-x) & \text{if } x < 0, \\ \operatorname{atan} x = \pi/2 - \operatorname{atan}(1/x) & \text{if } 1 < x, \\ \operatorname{atan} x = \pi/6 + \operatorname{atan}[(x - c)/(1 + cx)] & \text{if } \tan(\pi/12) < x \le 1. \end{array} \tag{2.4b}$$

Representation of the acos and asin functions is thus straightforward, as

$$\operatorname{acos} x = 2 \operatorname{atan}[\sqrt{1 - x^2}/(1 + x)], \quad \operatorname{asin} x = 2 \operatorname{atan}[x/\sqrt{1 + x^2}]. \tag{2.5}$$

Defining $R = (\ln\ 2)/2$, the $n = 3$ Padé approximation [see (8.8)] may be used to approximate $\exp r$ over the range $-R \le r \le R$, to about 9 decimal digits or better; this approximation may be written

$$\exp r \approx \frac{120 + 60\, r + 12\, r^2 + r^3}{120 - 60\, r + 12\, r^2 - r^3}. \tag{2.6a}$$

To extend the range over which we can accurately apply this formula, we define $x = a + r$ for a convenient choice of a that keeps the residual r in the range $-R \le r \le R$, noting that $\exp(a + r) = \exp(a)\, \exp(r)$. In particular, we take $a = k\, \ln 2$ for the appropriate integer k that keeps r in this range by taking

$$k = \lfloor 0.5 + x/\ln 2 \rfloor, \quad r = x - k\, \ln 2 \quad \Rightarrow \quad \exp x = \exp(a + r) = \exp(k\, \ln 2)\, \exp r = 2^k \exp r. \tag{2.6b}$$

If $\exp r$ is stored as a floating point representation $(-1)^s \times 2^{e - e_{\text{off}}} \times 1.f$ (see §1.1.4), where, e.g., e is an 8 bit representation of the exponent in single precision, then calculating 2^k times $\exp(r)$ involves simply adding k to e, in binary, while leaving the sign bit s and fractional part f unchanged. This can be done (when programmed appropriately) quickly and without any loss of precision.

For $1 \le r < 2$, define y as follows, noting that $\ln r$ may be expressed conveniently [see (B.84)] as

$$y = \frac{r - 1}{r + 1} \quad \leftrightarrow \quad r = \frac{1 + y}{1 - y} \quad \Rightarrow \quad \ln r = \ln \frac{1 + y}{1 - y} = 2 \sum_{i=0}^{\infty} y^{2i+1}/(2i + 1). \tag{2.7a}$$

Assume now that some $x > 0$ is stored as a floating point representation $x = r\,2^n$, where $n = e - e_{\text{off}}$, $r = 1.f$ (and thus $1 \le r < 2$), and $x > 0$ (and thus the sign bit used is $s = 0$; see §1.1.4). Note again that, e.g., for single precision arithmetic, e is an 8 bit representation of the exponent. It follows from $x = 2^n\,r$ and (2.7a) that

$$\ln x = n \ln 2 + \ln r \approx n \ln 2 + 2 \sum_{i=0}^{p} y^{2i+1}/(2i+1) \quad \text{for} \quad y = \frac{r-1}{r+1}; \tag{2.7b}$$

with $p = 6$ terms retained in this sum, accuracy to 8 decimal digits is obtained (note that constants like $\ln 2$ above, and $\ln 10$ below, may be precomputed and stored). Again, like the approximation of $\exp(x)$ in (2.6), the efficient approximation of $\ln(x)$ in (2.7) directly leverages the floating point representation that modern computers use to express real numbers. Approximation of $\log$ functions with other bases is straightforward via

$$\log_a x = (\log_c x)/(\log_c a), \quad \Rightarrow \quad \log_{10} x = (\ln x)/(\ln 10), \quad \log_2 x = (\ln x)/(\ln 2), \quad \ldots \tag{2.8}$$

Simple Matlab codes are available in §2 of the RR repository that demonstrate the use of the several approximations above for calculating the most common special functions of interest, specifically (2.1) for $\cos$, (2.2) for $\sin$, (2.3) for $\tan$, (2.4) for atan and[1] atan2, (2.5) for acos and asin, (2.6) for $\exp$, (2.7) for $\ln$, and (2.8) for $\log_{10}$, $\log_2$, etc. Any practical (fast) embedded application must rewrite such algorithms efficiently in C. Accurate and compact approximations of many other special functions are also available.

2.7 Pseudorandom number generators (PRNGs)

To provide cryptographic security, one can build a **true random number generator** (**TRNG**) that generates random numbers from a physical (e.g., thermodynamic) process with entropy (see §1.5.3.4). However, MPUs and MCUs are useful in part because their behavior is entirely predictable, so it is not obvious at first how to use an MPU or MCU appropriately to produce an adequately "random" sequence for a given application[2].

The development of a deterministic **pseudo random number generator** (**PRNG**) capable of producing sequences that are "effectively random" in application thus requires significant care. PRNGs generally operate as iterative algorithms (see, e.g., §2.7.1 and 2.7.2 below) that generate very long sequences of unsigned integers (usually, 32-bit or 64-bit) that eventually repeat. PRNGs can be initialized randomly, for example, by using the number of microseconds since some epoch on the system clock when the code is started, and can easily be postprocessed[3] to generate the following three useful types of "effectively random" sequences:

A) real numbers with uniform distribution between two limits a and b,

B) real numbers with gaussian distribution, with mean μ and variance σ^2, and

C) integers with equal likelihood between two limits a and b.

[1] Noting that $\text{atan}\, x \in [0, \pi/2)$ if $x \ge 0$, the following definition is often useful to remove ambiguity:

$$\text{atan2}(b,a) \triangleq \begin{cases} \text{atan}\,|b/a| \cdot \text{sgn}\ b & \text{if } b \neq 0,\ a > 0, \\ \pi/2 \cdot \text{sgn}\ b & \text{if } b \neq 0,\ a = 0, \\ (\pi - \text{atan}\,|b/a|) \cdot \text{sgn}\ b & \text{if } b \neq 0,\ a < 0, \\ 0 & \text{if } b = 0,\ a \ge 0, \\ \pi & \text{if } b = 0,\ a < 0, \end{cases} \quad \text{where} \quad \text{sgn}\ b = \begin{cases} -1 & \text{if } b < 0, \\ 0 & \text{if } b = 0, \\ 1 & \text{if } b > 0; \end{cases} \tag{2.9}$$

note that $\text{atan2}(b,a) \in (-\pi, \pi]$. The definition of $\text{atan2}(b,a)$ computes $\text{atan}\,(b/a)$, where b is vertical distance from the origin and a is horizontal distance, while placing the angle in the correct quadrant based on the signs of a and b.

[2] Indeed, it is often said that *The definition of insanity is doing the same thing over and over again and expecting different results...*

[3] Let y_i be a sequence of unsigned integers generated by a PRNG, with the max integer in the sequence denoted y_{max}. Sequences of type A may be generated by computing $x_i = a + (b-a)\,y_i/(y_{max}+1)$. Sequences of type B may be generated by applying a Box Muller transform, taking $z_i = \sqrt{-2\ln x_i}\,\cos(2\pi\,x_{i+1})$ and $z_{i+1} = \sqrt{-2\ln x_i}\,\sin(2\pi\,x_{i+1})$ for odd i, where $\{x_i, x_{i+1}\}$ are generated as in type A with $a = 0$ and $b = 1$. Sequences of type C may be generated by rounding down sequences of type A.

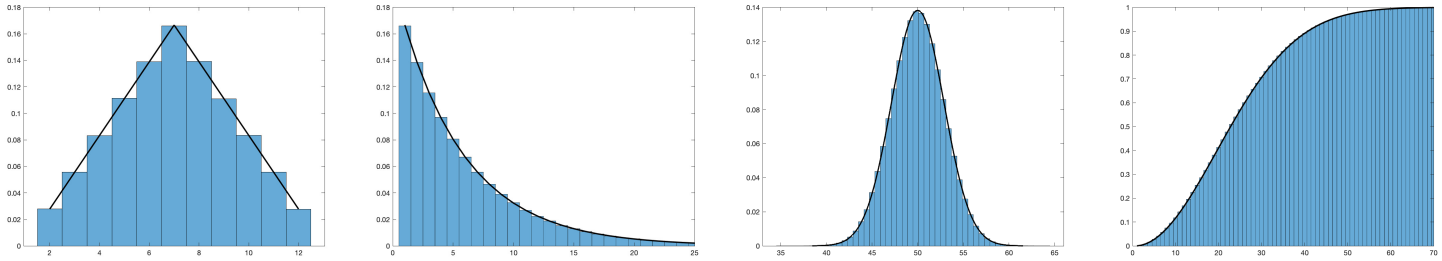


Figure 2.1: (a) The **triangular** probability distribution function (**PDF**) generated by a model of the sum of two fair 6-sided dice (see RR_Fair_Dice_Test). (b) The **exponential** PDF generated by a model of the minimum $j > 0$ for which $x_i = x_{i+j}$ in the repeated rolling of a single 6-sided die (see RR_Repeat_Symbol_Test). (c) The **gaussian** PDF generated by a model of the sum of 100 real numbers uniformly distributed between 0 and 1 (see RR_Overlapping_Sums_Test). (d) The **gamma** cumulative distribution function (**CDF**) generated by a model of the Birthday Problem, indicating that in a random grouping of only 23 people, there is over a 50% chance that at least two have the same birthday, and with 50 people, there is a 97% chance (see RR_Birthday_Problem_Test).

Good PRNGs produce unsigned integer sequences that, primarily[4]:
1) are characterized by **good statistical properties**, largely mimicking those of a thermodynamic process,
2) have a **very large period**, so in application they do not exhibit a repeating pattern, and
3) are **fast to compute**, in a small memory footprint, when coded in a low-level language like C, C++, or Rust.

A fair 6-sided die (cf. loaded or shaved dice) rolls $\{1, 2, 3, 4, 5, 6\}$ with equal probability. The sum of two such dice will give a total of $\{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$ with probability $\{1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1\}/36$. A good PRNG, adjusted to give integers between 1 and 6 for each die with equal probability (type C above), should mimic this **triangular** distribution over millions of trials (see Figure 2.1a) without a noticeable repeating pattern.

When determining x_i by rolling a single fair n-sided die, the odds that the next number rolled, x_{i+1}, is the same as x_i is $p(1) = 1/n$. Thus, defining $c = (n-1)/n$, the odds that the minimum $j > 0$ for which $x_i = x_{i+j}$ is equal to j is just $p(j) = c^{j-1}/n$, generating what is called the **exponential** distribution (see §6.2.2 of *NR*). A good PRNG (again, of type C above) measured in this manner should mimic this exponential distribution over millions of trials (see Figure 2.1b), again without a noticeable repeating pattern.

Consider next a PRNG adjusted to give a real number x between $a = 0$ and $b = 1$ (type A above); this **uniform** distribution (see §6.2.2 of *NR*) is said to have a mean of $\bar{x} \triangleq \mu = (a+b)/2 = 1/2$ and a variance of $\overline{(x-\mu)^2} \triangleq \sigma^2 = (b-a)^2/12 = 1/12$. Any sum of $n = 100$ consecutive real numbers x_i so generated, denoted $y_i = \sum_{j=0}^{n-1} x_{i+j}$, should total about $y_i \approx n\,\mu = 50$, but will sometimes be a bit higher, and sometimes a bit lower. It is a remarkable consequence of the Central Limit Theorem (§6.2.3 of *NR*) that a histogram of the computed values of this sum will tend towards a **gaussian** distribution (see §6.2.1 of *NR*), with a mean of $n\,\mu = 50$ and a variance of $n\,\sigma^2 = 8.333$. A good PRNG measured in this manner should mimic this gaussian distribution over millions of trials (see Figure 2.1c), again without a noticeable repeating pattern.

Finally, the odds that 2 people selected at random do not have the same birthday is $364/365$. By the same logic, the odds that, in a random grouping of n people, at least 2 do NOT have the same birthday is $p_{\text{NOT}}(n) = \prod_{k=1}^{n-1}(365-k)/365$. The odds this is false (that is, in a random group of n people, at least 2 DO have the same birthday) is $p_{\text{DO}}(n) = 1 - p_{\text{NOT}}(n)$. This distribution, known as the **Birthday Problem**, is a cumulative distribution function (CDF) of a gamma distribution (see §6.2.2 of *NR*). A good PRNG measured in this manner should mimic this type of distribution over millions of trials (see Figure 2.1d),

The Diehard and Dieharder (Yippee-Ki-Yay John McClane!) and TestU01 and PractRand suites of statistical tests for PRNGs, which quantify property 1 above, evolved from over 100 pages of original analysis of the subject by Knuth [1]. The many statistical experiments incorporated into such test suites are similar to those described in the previous four paragraphs. Unfortunately, all such "randomness" tests are only statistical in nature.

[4] **Difficulty to predict** is sometimes also mentioned a fourth desired property, though none of the PRNGs surveyed in §2.7 is cryptographically secure, and most of the modern PRNGs summarized in §2.7.2-2.7.4 have indeed already been "cracked".

2.7.1 Linear Congruential Generators (LCGs)

Linear congruential generators (LCGs) are the essential starting point. LCGs are PRNGs defined by a simple recurrance of the form[5]

$$x_n = (a \cdot x_{n-1} + c) \bmod m, \tag{2.10}$$

where the multiplier a, increment c, modulus m, and states $x_n, x_{n-1}, \ldots$ are unsigned integers. An LCG with $c = 0$ is often called a **multiplicative congruential generator** (MCG) or **Lehmer generator**. The evolution of an LCG is determined by the value of x at the previous iteration, x_{n-1}, together with the $\{a, c, m\}$ constants.

Two types of LCGs are of particular interest: (a) those with prime m, which have the best statistics, and (b) those with $m = 2^b$ and odd c, where b is the number of bits in the binary representation of the integers being used, which are even faster to compute when implemented in a language that wraps on integer overflow. When using either type of LCG, the trick is to select a well for a given m. Most choices of this parameter result in bad PRNGs, with short periods and/or bad statistics. Some choices, though, give fairly "good" PRNGs (in terms of properties 1, 2, and 3 itemized on the previous page) given the simplicity of (2.10). A starting point to find a good value for a for the $m = 2^b$ case, known as the Hull-Dobell Theorem, is to take $\text{mod}(a, 8) = 5$ [i.e., $a = m \cdot 8 + 5$ for some m]; though this choice generates PRNG sequences with **full period** (i.e., which repeat only after m elements), most values of a so generated in fact still do not have good statistics. Parameters leading to good LCGs must be searched for exhaustively, and are well tabulated in the literature [6].

As a (very small) example, take $a = 19 \cdot 8 + 5 = 157$, $c = 47$, and $m = 2^8 = 256$ in (2.10). Starting from $x = 0$, this LCG generates every integer, once, from 0 to $m - 1 = 255$, then repeats:

```
  0   47    2  105  148  243   54   77  104  247  170  113  124   59   94  213  208
191   82  121  100  131  134   93   56  135  250  129   76  203  174  229  160   79
162  137   52   19  214  109    8   23   74  145   28   91  254  245  112  223  242
153    4  163   38  125  216  167  154  161  236  235   78    5   64  111   66  169
212   51  118  141  168   55  234  177  188  123  158   21   16  255  146  185  164
195  198  157  120  199   58  193  140   11  238   37  224  143  226  201  116   83
 22  173   72   87  138  209   92  155   62   53  176   31   50  217   68  227  102
189   24  231  218  225   44   43  142   69  128  175  130  233   20  115  182  205
232  119   42  241  252  187  222   85   80   63  210  249  228    3    6  221  184
  7  122    1  204   75   46  101   32  207   34    9  180  147   86  237  136  151
202   17  156  219  126  117  240   95  114   25  132   35  166  253   88   39   26
 33  108  107  206  133  192  239  194   41   84  179  246   13   40  183  106   49
 60  251   30  149  144  127   18   57   36   67   70   29  248   71  186   65   12
139  110  165   96   15   98   73  244  211  150   45  200  215   10   81  220   27
190  181   48  159  178   89  196   99  230   61  152  103   90   97  172  171   14
197    0   47    2  105  148  ...
```

The above PRNG sequence was generated with the following simple line of Matlab code:

```
clear, a=157, c=47, m=256, x(1)=0, for i=2:m+5, x(i)=mod(a*x(i-1)+c,m); end, x
```

In the above line of code, replacing $a = 157$ with $a^* = 181$ and replacing $c = 47$ with $c^* = 197$ generates the same sequence of integers but in reverse order (try it!); that is, such LCGs are **reversible**, simply by replacing a with a^* and c with c^*, where $\text{mod}(a \cdot a^*, m) = 1$ [that is, $m \cdot e + a \cdot a^* = 1$ for some integer e; see §A.7.1] and $c^* = mod(c \cdot (m - a^*), m)$. Replacing c with any other odd integer produces a different PRNG sequence that is qualitatively similar. The first 9 entries of the above PRNG sequence may be written in binary as

```
0  00101111 00000010  01101001  10010100  11110011  00110110  01001101  01101000
```

Note that the least significant bits (LSBs) alternate between 0 and 1 (that is, odd entries in the sequence are even integers, and even entries in the sequence are odd integers). The next significant bit follows the sequence

[5] LCGs with $c \neq 0$ are *affine*, not linear, in the state x, though the misnomer "LCG" is used broadly in the literature.

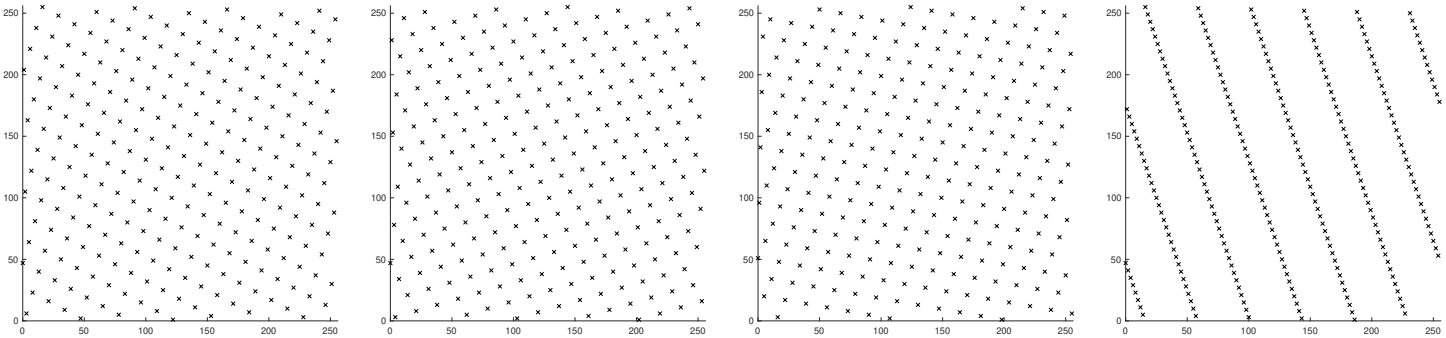


Figure 2.2: Coordinates of adjacent pairs of random numbers generated by an LCG with $m = 256$ and (a) $\{a, c\} = \{157, 47\}$, (b) $\{a, c\} = \{181, 47\}$, (c) $\{a, c\} = \{45, 51\}$, (d) $\{a, c\} = \{125, 47\}$. All four of these full-period LGCs repeat after m terms; however, the distribution of the pairs so generated is much less "uniform" (that is, more "clustered") in case (d) than it is in cases (a), (b), and (c).

0,1,1,0,0,1,1,0,0,1,1, ... That is, the statistics of the lower bits in an $m = 2^b$ LCG follow a very noticeable pattern, as the lower bits of such an LCG affect the evolution of the higher bits, but the higher bits do not affect the evolution of the lower bits. However, when m is large, several of these lower bits can easily be suppressed when outputting the result of the PRNG subroutine; it is effectively the "mod m" part of a (deterministic) LCG that, when a is a substantial fraction of m, makes the higher bits of the LCG appear to be "more random".

Another way to quantify the quality of a PRNG is to examine the coordinates of all adjacent pairs or triplets or, more generally, k-tuples, of the random numbers generated, when plotted in R^k. A good PRNG reveals a uniform distribution of the coordinates so generated. In the output of an LCG, even when $\{a, c, m\}$ are selected such that the resulting PRNG sequence has a relatively large period, the coordinates generated by this process sometimes cluster in a relatively small number of lines (for $k = 2$), planes (for $k = 3$), or hyperplanes (for $k > 3$), as illustrated for the $k = 2$, $m = 256$ case in Figure 2.2d; this clustering is undesirable.

The period of the $\{a, c, m\} = \{157, 47, 256\}$ LCG discussed above is far too short to be useful in most applications. Good compilers (like C/C++ and Rust, but unlike Matlab[6]) that "wrap" on integer overflow (i.e., that ignore any bits generated that are larger than b) implement the mod $m = 2^b$ in (2.10), for $b = 32$ or 64, automatically. Good full-period LCGs of this form are given (see [6]) by taking any odd c and taking, e.g., a = 2,891,336,453 for $m = 2^{32}$ (using uint32), or a = 3,935,559,000,370,003,845 for $m = 2^{64}$ (using uint64).

2.7.1.1 k-dimensional equidistribution

Now reëxamine the coordinates of adjacent pairs, triplets, or more generally k-tuples, in a sequence of integers generated by a simple LCG. Focusing to begin with on the pairs illustrated in Figure 2.2a-c (taking $k = 2$), it is seen that LCGs with "good" uniformity of pairs still only generate m pairs, of a total of m^2 possible pairs. In some specific situations (e.g., generating lottery numbers with a PRNG), it may be desired that a PRNG might ultimately (in theory, at least) generate *every* possible k-tuple, for some k, with equal probability.

We can achieve the desired "equidistribution" of all k-tuples, when plotted or considered in k-dimensional space, simply by running k such independent LCGs (each referred to as a "stream") in parallel, with each stream having the same (power of two) m but different ("good") values of a and c, noting the second paragraph of §2.7.1. As suggested by [8], to get this to work properly, for $j = 1, \ldots, (k - 1)$, each time the j'th stream is updated to some special value (say, $x_j = 0$), a sort of "carry" (as in integer addition) occurs, and the $(j + 1)$'th stream is incremented an extra time, by adding $2c$ instead of c in (2.10). Implementation of this approach using simple LCGs is illustrated in Figure 2.3, and illustrates some unfortunate flaws.

[6] Matlab's default behavior is to saturate (not wrap) on integer overflow. To get around this, as of this writing, you have to purchase Matlab's Fixed-Point Designer, which you might otherwise have no need for, or use, e.g., the RR_uint64.m class, which is free.

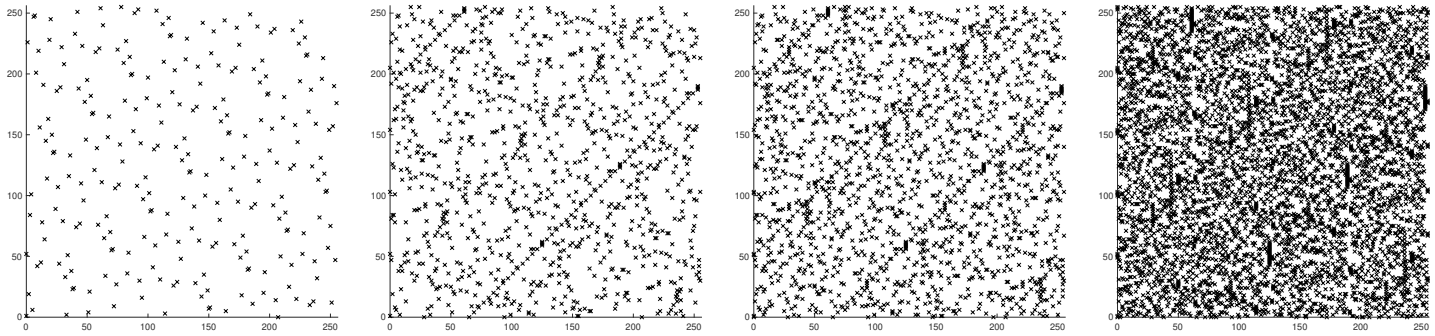


Figure 2.3: Coordinates of ordered pairs of integers generated by $k = 2$ very small LCG streams (one with $\{a, c, m\} = \{157, 47, 256\}$, the other with $\{a, c, m\} = \{45, 51, 256\}$) to form a combined LCG with period $2^{kb} = (256)^2$, which generates each of the m^k possible integer k-tuples (e.g., pairs), once, before repeating. Plots (aka **randograms**) show the points generated by this approach after (left to right) m, $4m$, $7m$, and $20m$ iterations; unfortunately, significant structure (long diagonals and short vertical clusters) is evident in such early iterations following this simplistic LCG-based approach. When implemented with significantly larger m, and in particular by incorporating PRNG output functions which cleverly permute the internal LCG state, the structure that appears in early iterations of such randograms is greatly reduced.

2.7.1.2 Improving PRNGs beyond LCGs and MCGs

Good PRNGs are hard to find. Much effort has been put into the search for good PRNGs. A candidate PRNG may be relatively (a) fast to calculate, with (b) small memory footprint and (c) small code size and (d) long period, and may (e) satisfy many statistical randomness tests (see, e.g., Figures 2.1 and 2.2), only to fail some other randomness test. Though increasing the size of the internal state is certainly valuable, when restricted to using 32-bit or 64-bit arithmetic, LCGs alone have proven to be insufficient for many applications.

Over the years, *many* PRNGs have been developed and implemented, some of which are fairly good statistically but unnecessarily complex in terms of both space usage and code size, including the Mersenne Twister[7] and stream ciphers like RC4 and its modern successor ChaCha20, and many of which are simpler but with inferior statistical properties, including IBM's once pervasive yet "truly horrible" RANDU, Numerical Recipe's RanQ1, and unix's drand48, rand, and random implementations. In the following, we briefly survey three modern families of PRNGs, which effectively supersede all of the PRNGs mentioned above:

• **Permuted Congruential Generators** PRNGs, specifically O'Neill's PCG32 and PCG64 algorithms, propagate an internal LCG state with $n = 64$ or 128 bits, and output integers of size $n/2$ bits (that is, 32 or 64 bits), which are generated via clever permutations (shifts/XOR/OR) of the internal LCG state.

• **XOR/shift**, **XOR/shift/rotate**, and **XOR/rotate/shift/rotate** PRNGs, specifically Marsaglia's XorShift*64/32 and XorShift*128/64 algorithms, and Vigna's xoshiro256++, xoshiro256**, xoroshiro128++, and xoroshiro128** algorithms, propagate by taking an XOR of bit-shifted versions of the internal state (with n bits), and output (via various additional shift/rotate permutations) 32-bit or 64-bit integers of size $n/2$ or $n/4$ bits.

• **Multiply With Carry** PRNGs, such as Kaitchuck's MWC128XXA32 and MWC256XXA64 algorithms, use an internal state totaling $n = 128$ or 256 bits, propagated as 4 separate $n/4$ bit integers, and output integers of size $n/4$ bits, again by implementing permutations (XOR/XOR/Add, thus the "XXA" in the names) of the state.

The pros and cons of these three modern families of high-quality (fast, small, and statistically great) PRNGs have been debated extensively online by their authors and others; from the typical user's perspective, in the present author's opinion, their differences amount, effectively, to much ado about fairly minor properties.

[7] The Mersenne Twister is based on the Mersenne prime $M_{19937} = 2^{19,937} - 1$. Mersenne primes are prime numbers that may be written as $M_n = 2^n - 1$ for some integer n. The largest known prime numbers are all Mersenne primes; as of this writing, the largest of these is $2^{82,589,933} - 1$, which when written in base 10 has 24,862,048 digits.

```
// *Really* minimal PCG32 code / (c) 2014 M.E. O'Neill / pcg-random.org
// Licensed under Apache License 2.0 (NO WARRANTY, etc. see website)
typedef struct { uint64_t state;  uint64_t inc; } pcg32_random_t;
uint32_t pcg32_random_r(pcg32_random_t* rng)
{
    uint64_t oldstate = rng->state;
    // Advance internal state
    rng->state = oldstate * 6364136223846793005ULL + (rng->inc|1);
    // Calculate output function (XSH RR), uses old state for max ILP
    uint32_t xorshifted = ((oldstate >> 18u) ^ oldstate) >> 27u;
    uint32_t rot = oldstate >> 59u;
    return (xorshifted >> rot) | (xorshifted << ((-rot) & 31));
}
```

Figure 2.4: O'Neill's minimal (single-stream, no skip) PCG code, implemented in C using uint64 arithmetic and generating a uint32 output; a separate code is used to randomly initialize the rng struct (based, e.g., on the state of the system clock). Two 64-bit unsigned integers, rng->state and rng->inc, are maintained in memory to advance the PRNG stream. Efficient low-level coding like this is essential for fast execution of core functions; to improve readability, most algorithms in this textbook are instead provided in Matlab syntax (see, e.g., RR_PCG32.m), ultimately leaving it to the user to convert to the user's low-level language of choice (C, C++, Rust, ...).

2.7.2 Permuted Congruential Generators (PCGs)

O'Neill's PCG32 and PCG64 Permuted Congruential Generators (PCGs; see [8] for an illuminating discussion) provide substantial improvements over using LCGs alone. These PRNGs use LCGs as the underlying algorithm to propagate their (64-bit or 128-bit[8]) internal state (taking $a = 6364136223846793005$ in the former case, and $a = a_{\text{high}} \cdot 2^{64} + a_{\text{low}}$ where $a_{\text{high}} = 2549297995355413924$ and $a_{\text{low}} = 4865540595714422341$ in the latter), and simply perform some targeted bit permutations on the LCG state to generate their (32-bit or 64-bit) output, substantially improving the overall PRNG behavior. By so doing, PCGs build upon simple LCGs to improve the statistics of their output, while inheriting many useful features of the LCGs upon which they are based, including the generation of multiple streams, the formation of combined LCGs with period 2^{kb} providing k-dimensional equidistribution, and fast skipping forward/backward in any given random number stream.

Figure 2.4 presents O'Neill's minimal (single-stream, no skip) C implementation of PCG32, with a 64-bit internal state and increment and a 32-bit output. Note that the line after the "Advance internal state" comment is simply the underlying LCG algorithm. The precise logic for the specific bit permutations given in the last few lines of O'Neill's code is described in [8]; note that

- a >> k denotes a rightshift of the bitwise representation of a by k bits[9],
- a << k denotes a leftshift of the bitwise representation of a by k bits,
- a & b denotes a logical AND of the bitwise representations of a and b (see Table 1.3),
- a ^ b denotes a logical XOR of the bitwise representations of a and b,
- a | b denotes a logical OR of the bitwise representations of a and b,
- - a denotes a two's complement representation of the negation of a (see §1.1.3).

In short, these bit permutations (which execute much faster in C, C++, or Rust than Matlab!) reduce the uint64 LCG variable oldstate to the uint32 output of this PCG in a manner that provides significantly improved statistical characteristics of the PRNG. PCG64 is structurally similar. A full-featured pedagogical Matlab implementation of the PCG32 algorithm (including multiple streams, and fast skipping forward/backward within a stream) is provided in RR_PCG32.m.

[8] Smaller PCGs by O'Neill are also available, but given the prevalence today of fast 32-bit MCUs for modern robotics applications, such smaller PCGs are likely of limited practical benefit.

[9] Note that the bit positions that are vacated by this shift are filled with zeros.

2.7.3 XOR/shift, XOR/shift/rotate, and XOR/rotate/shift/rotate PRNGs

Marsaglia's XorShift*64/32 and XorShift*128/64 PRNGs...
xoshiro256++, xoshiro256**,
xoroshiro128++, and xoroshiro128**

2.7.4 Multiply With Carry (MWC) PRNGs

...

Example 2.1 "Smart" shuffling of jokes and song playlists. To conclude §2.7, consider the following:

Sequence **A**: 82AF87C1E08A9759EAF1818B2C6E4031F01030A833150974BD4875DFA2195482

Sequence **B**: 02C8F653742ACD813940FBE673C85109AE4B28CDF03AE9285B601F9475DC0A6E

Sequence **A** was generated using the Mersenne Twister PRNG, as implemented by Matlab's built-in command **randi** ([0,15],1,64) (see §A.1), which does well on the four batteries of statistical tests mentioned previously.

You might aspire to build a device that tells a joke or plays a song at "random", from a not-very-large list of (in this example) 16 known jokes or songs, enumerated 0 through F. Note that Sequence **A** contains subsequences like "01030" (that is, within of 5 characters, 1 symbol appears 3 times). While such behavior is indeed anticipated for a long sequence generated by an "effectively random" process[10], repeating "joke 0" or "song 0" three out of five times may seem "insufficiently random" to the user. Sequence **B** above was also generated with the **randi** command, but rejecting each number so generated that duplicates one of the last 8 numbers generated (see RR_Dad_Jokes.m). A modified (albeit, "less random") randomizing behavior like this is likely to be preferred if you are listening to songs on "shuffle mode", or are attempting robotic comedy (is that even a thing?). △

[10] For example, within the first 1000 digits of π written in decimal notation, generating a seemingly random (?) sequence of digits, the subsequence "999999" appears!

Chapter 3

Sensors, actuators, and interfaces

Contents

This chapter surveys some of the most common components that attach to SBCs to form useful **cyber-physical** (aka **electro-mechanical**) systems. These components generally come in three broad categories: **sensors**, **actuators**, and **user interfaces**. Though not at all exhaustive, we discuss many representative examples of components in each of these categories in turn in this chapter; a brief discussion of how some such components may be put together to make larger systems is deferred to §16.

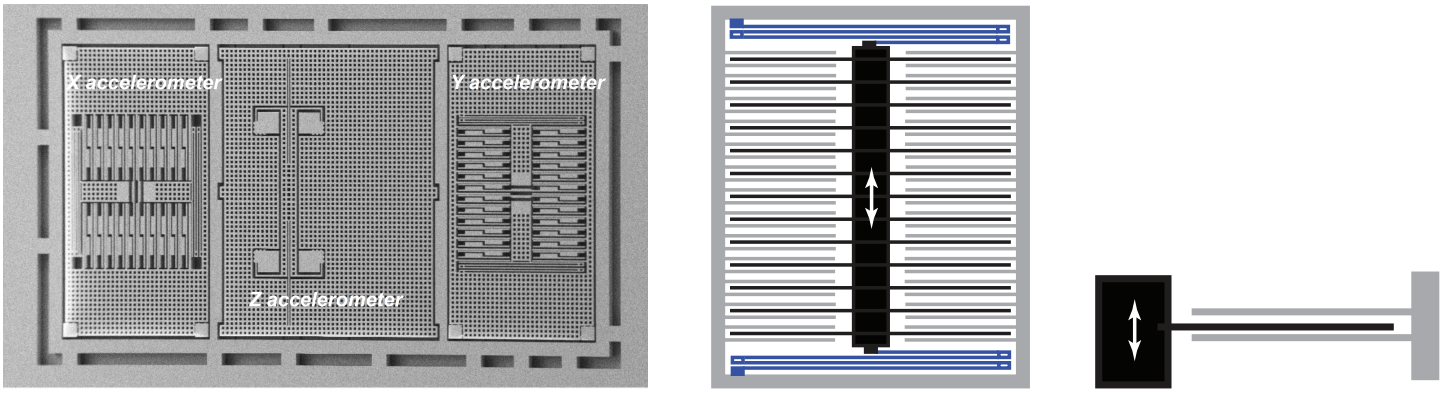


Figure 3.1: A prototype 3-axis MEMS accelerometer by ST: (a) photograph, (b) schematic of the y-axis component, and (c) principle of operation. The schematic highlights (grey) the "fixed" outer body, including the "fixed" outer plates, (black) the "moveable" proof mass, including the attached "moveable" inner plates, and (blue) the flexible beams attaching the "moveable" proof mass to the "fixed" outer body. Note that the notions of "fixed" and "moveable" here are w.r.t. the vehicle to which the device is rigidly attached, which itself moves. Note also, at right, that each moveable inner plate forms two capacitors, one with the fixed outer plate directly above, and one with the fixed outer plate directly below. As the proof mass moves w.r.t. the outer body, the distance between the plates in these two capacitors changes (one increases, the other decreases). Sensitive electronics packaged with the MEMS accelerometer measures this difference in capacitance (averaged appropriately over all the plates), thereby inferring the component of the acceleration of the vehicle in each coordinate direction.

3.1 Sensors for obtaining situational awareness

3.1.1 Inertial Measurement Units (IMUs): accels and gyros

A mobile robot needs a certain degree of "situational awareness" to detect its own motion, and to respond appropriately to changes in its environment. A starting point to achieve such situational awareness of a vehicle's movement is to estimate its linear and angular acceleration in an inertial frame using a modern MEMS[1] 3-axis accelerometer, as illustrated in Figure 3.1, and 3-axis gyroscope, built similarly.

Typical accuracy of a modern MEMS accel is better than 10^{-3} g (that is, 1/1000 of the acceleration due to gravity), and typical accuracy of a modern MEMS gyro is better than 0.1 deg/s/s. Such convenient sensors may thus be seen as very good (certainly as compared to a decade ago), but still quite insufficient to integrate over a long period of time to determine linear and angular velocity as compared to some initial state, a process referred to as **dead reckoning**. Other sensors providing absolute position, orientation, linear velocity, and/or angular velocity (see §3.1.2) are thus also needed to supplement the data provided by the MEMS accels and gyros.

Some MEMS sensors have inherent dynamics that may be significant at the frequencies of interest in a given system. Note in Figure 3.1 that a MEMS accelerometer is, effectively, a small floating proof mass supported by a spring, and therefore has a response magnitude that is inherently a function of the forcing frequency. An expanded dynamic range of such devices may generally be obtained by active electrostatic **force rebalancing**; that is, by closing a feedback control loop around the sensor itself, applying an electrostatic force (using some of the plates indicated in Figure 3.1) that is just sufficient to keep the proof mass from moving w.r.t. the outer

[1] A **Micro-Electro-Mechanical-System** is a very small physical system made using the same mask/expose/etch technology used to manufacture silicon chips. Today, this **COTS** (**commercial off-the-shelf**) technology is very mature, and several types of MEMS sensors are mass produced on a very large scale. For example, MEMS accelerometers are used in airbag deployment systems in automobiles, video game controllers & smartphones, and hard disk drives. MEMS gyros are also mass produced, albeit on a somewhat smaller scale, for use in video game controllers & smartphones.

body, then measuring the electrostatic force required to "rebalance" the floating mass within the device in order to determine the acceleration applied to the entire system. This essentially supplants the mechanical time constant of the device, $\sqrt{m/k}$, with the electrical time constant of the sensor control circuit, RC, which may generally be made much faster. The most accurate MEMS accels and gyros available today all incorporate such electrostatic rebalancing feedback.

3.1.2 Inclinometers, magnetometers, barometers, and GNSS/GPS systems

Supplemental sensors may be used by a vehicle to provide data, albeit approximate, regarding absolute position, orientation, linear velocity, and angular velocity.

If a vehicle is near the Earth's surface and is relatively motionless in the lab frame, then a MEMS accelerometer may be used as an **inclinometer** to estimate the vehicle's angle (pitch and roll) w.r.t. the gravity vector (directed towards the center of the Earth), which is often useful. An inclinometer can also be used, more simply, to determine if a box-shaped object like a cellphone (that is, a rectangular cuboid) is oriented, approximately, (1) face up, (2) face down, (3) right side down, (4) left side down, (5) lower side down, or (6) upper side down; this problem is commonly referred to as **6D orientation** determination.

There is substantial natural variability in the Earth's magnetic field; this variability can be especially pronounced in human-built environments. However, obtaining even a rough approximation ($\pm 10\%$) of magnetic north in the local environment is often a useful starting point when attempting to maintain orientation. MEMS **magnetometers** provide precisely this functionality. Remarkably, the Earth's magnetic field may even be measured underground or underwater, where it is otherwise quite difficult to get useful orientation information.

Atmospheric pressure decreases at a rate of 11.3 Pa per meter increase in altitude at sea level; this variation is remarkably consistent, even in fairly windy conditions. Thus, a change of about 1 Pa, which a modern **MEMS barometer** can easily resolve, corresponds to a change of about 8.8 cm in altitude. When attempting to hold a particular altitude with a drone, feedback based on barometer measurements is quite helpful.

On Earth, the gold standard of absolute position and time information today is obtained with a **global navigation satellite system** (aka **GNSS**), which include the US Space Force's **GPS** system as well as several others, including EU's **Galileo** system, China's **BeiDou** system, Russia's **GLONASS** system, India's **NavIC** system, and Japan's **QZSS** system. Amongst these, GPS (with 31 satellites) is the most extensive and advanced; indeed, the abbreviation GPS is commonly used synonymously with the more general term GNSS, even when using GNSS systems that monitor signals from mutiple satellite constellations. GPS systems typically achieve 5 meter accuracy; those using the new L5 band, which is scheduled to be fully operational in 2027, achieve an accuracy of about 30 centimeters. Small units incorporating Differential GPS (aka DGPS, which compare the GPS signals received locally with those at a known reference station) are now readily available and, quite remarkably, can achieve relative position accuracies of 1 to 3 centimeters.

Given all of the available techniques discussed above to estimate position and orientation, and the rate of change thereof, the question of which technique(s) to use in a given application is subtle, and varies from problem to problem. The best answer is usually to perform some sort of **sensor fusion**, using several of these techniques *at the same time*, blending them together to get the best estimate possible. This problem is delicate, as each of these different sensor technologies, as well as those discussed in §3.1.3, measure different quantities, have different types of uncertainties, and provide information at different rates. Essential algorithms that are used when performing sensor fusion include simply blending the low-frequency information from some sensors with high-frequency information from other sensors, known as **complementary filtering** and discussed further in §8.5.3, and leveraging directly the equations modeling the dynamics of the physical system itself, tother with Bayes' rule at each measurement update to minimize the estimate uncertainty, known as **Kalman filtering** and discussed further in *NR*.

3.1.3 Optical flow and Simultaneous Localization and Mapping (SLAM)

Depth imaging
RGB-D
2D and 3D Lidar
Example: Big box store
Camera-based systems

3.1.4 Ground truth via Motion Capture (MoCap) using triangulation & trilateration

Passive and active markers.
visual & RF beacons and microphone arrays
Broadband high-frequency RF MoCap
3-10 GHz, IndoTraq

3.1.5 Other sensors

Strain gauges. Piezoelectric effect
Liquid level sensors and **flow meters**
Thermocouples and **Ph meters**

3.2 Transferring power and signals to rotating components

It is often necessary to transfer power and/or signals on or off of rotating shafts that coordinate the motion of wheels, linkages, and other moving parts. Applications in which this need arises, among many, include:

- low-cost BDC *motors* (see §3.4),
- analog devices, called *resolvers*, used to measure shaft rotation (see §3.3.1),
- *robot arms* capable of large ranges of motion (see Figure 3.4c and §16.9.3),
- rotating sets of LEDs leveraging *persistance of vision* (PoV) effect to make images (see §3.8), etc.

There are four main methods of accomplishing such transfers of power and/or signals to rotating frames:

(a) brushes and commutators (Figure 3.2, discussed in §3.2.1),
(b) rotary transformers (Figure 3.3, discussed in §3.2.2),
(c) local COTS wireless communication protocols (§4.4) like bluetooth[2] (§4.4.2), and
(d) flexible wires with careful cable routing[3], in applications for which the total rotation is limited (Figure 3.4).

3.2.1 Brushes and commutators

The cheapest method to transfer power and/or signals to a rotating frame is to use stationary spring-loaded carbon graphite (sometimes with copper added) **brushes**, mounted to rub against (and, to make reliable electrical contact with) rotating conductive copper rings called **commutators** (see Figure 3.2). This approach can efficiently pass both low-bandwidth logical signals, as well as both alternating current (AC) and direct current (DC) power, from the stationary frame to the rotating frame, but often introduces significant electrical noise, motivating the use of some low-pass filtering (see §8 and §9) to remove. Commutators can either be the **continuous-ring** type, which provide continuous electrical contact as the shaft turns, or the **split-ring** type which, periodically, mechanically break the electrical connection when the shaft turns past a certain angle, and later reëstablish the electrical connection, with the opposite polarity, as the shaft turns further. Note that the clever use of split-ring commutators forms an essential component of the operation of brushed DC (BDC) motors, discussed in §3.4. Either way (using continuous-ring or split-ring commutation), due to mechanical wear, the brushes will eventually wear out. Most small BDC motors are designed be disposed of when the brushes wear out; in some older/larger BDC motors, the brushes may be replaceable by the user. Most newer large motors are brushless, which are more efficient and require less maintenance.

3.2.2 Rotary transformers

A more durable method for transferring AC power or AC signal to or from rotating frames is to use **rotary transformers**. With this approach, magnetically-coupled electromagnets are placed near to each other, one on the rotating shaft, and the other on its stationary housing. One of these electromagnets is driven by an AC input (typically oscillating at a frequency ω_e that is fast w.r.t. the shaft rotation); by **magnetic induction**, a concomitant AC current is picked up by the other electromagnet. Typical configurations are illustrated in Figure 3.3. This approach is more durable than using brushes and commutators, but can only handle AC signals and, depending on the strength of the magnetic coupling in the design, typically suffers 30% or higher power loss. This power loss is turned directly into waste heat, which means that rotary transformers are generally not well suited for power transfer in high-current applications.

[2] COTS bluetooth solutions are low latency, low power, and inexpensive, and are often preferred in high-bandwidth applications.
[3] Note that such wires must be routed very carefully, in order to not tug or foul during operation, or to fatigue too quickly.

Figure 3.2: Brush & commutator systems for transferring both DC and AC power and signals between rotating (green) and non-rotating parts. (a) Side (cut-away) view of a **continuous-ring brush/commutator system**; a spring is used to keep the (relatively soft) carbon graphite brush (gray), which is stationary, in constant contact with the copper commutator (orange) mounted to the rotating component (green) attached to the shaft (blue). (b) Top view of **split-ring brush/commutator system** for coordinated excitation of the rotating electromagnets of a BDC motor, with brushes {D,E} and commutators {F,G,H} labelled as in Figure 3.9. Images are schematic representations only; for notational clarity, the housings and bearings that keep the parts aligned and spinning freely are not shown.

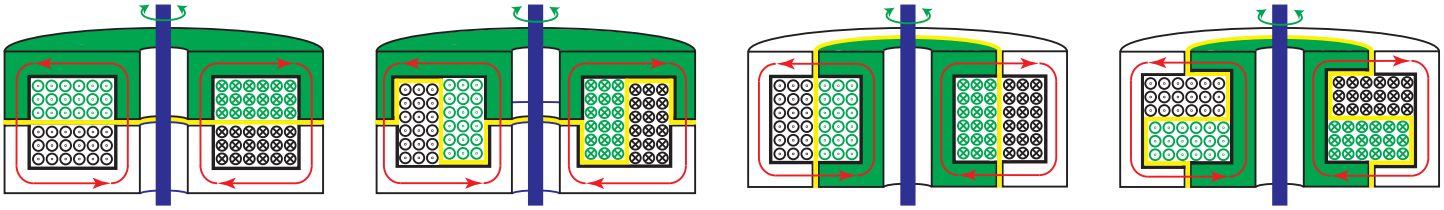


Figure 3.3: Four different configurations of **rotary transformers** for magnetically transferring AC (only) signals between rotating (green) and non-rotating parts. (a) & (b) are **flat-face** (aka **pot core**), (c) & (d) are **axial**; the stationary (black) and rotating (green) windings in (a) & (d) are **adjacent**, those in (b) & (c) are **coaxial**. A small **air gap** (yellow) separates the rotating and non-rotating components; minimizing the size of this gap leads to better magnetic coupling and thus higher efficiency. Magnetic flux lines are illustrated in red. Images are schematic representations only; for notational clarity, the housings and bearings that keep the parts aligned and spinning freely are not shown. A circle with a dot indicates the windings comes out of the page, and a circle with a cross indicates the windings go into the page; note that the individual windings are much smaller gauge wires than suggested by the circles shown here.

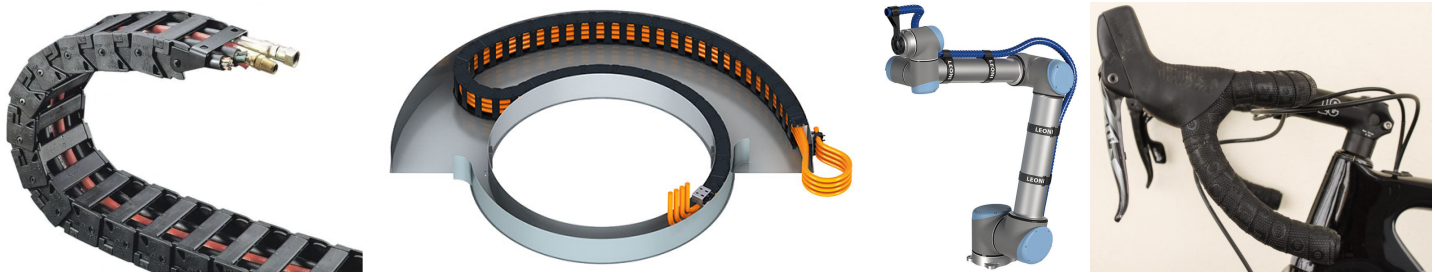


Figure 3.4: Example cable routings facilitating limited ranges of movement. (a) Linear drag chain (each link of the chain pivots in only one direction, and a finite amount, thus ensuring a prespecified minimum radius of curvature of the cables lying in the channel running through its center), (b) rotary drag chain (that is, a linear drag chain lying in a channel between two concentric walls which rotate wrt each other), and (c), (d) less-structured solutions that allow multiple degrees of freedom of movement. Cables manufactured using highly flexible silicone rubber insulation are particularly well suited for such applications.

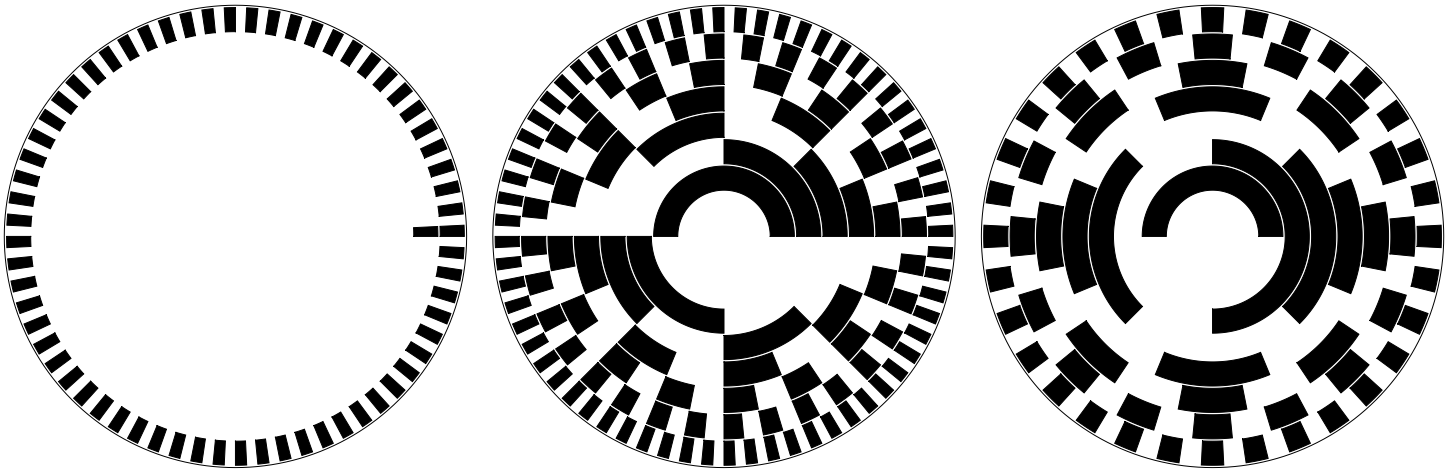


Figure 3.5: Common encoder disk slit patterns: (a) ABZ, (b) binary, (c) Gray. A flexible code for accurately producing such patterns is available at RR_encoders.m]

3.3 Sensors for measuring shaft rotation

Resolvers and **synchros** are *analog* devices that may be used to measure the rotation of a shaft, as discussed briefly in §3.3.1.

More commonly used today, however, are **rotary encoders**, which are *digital* devices used to measure the rotation of shafts. Rotary encoders (often, just called encoders) thus form the focus of our attention in this chapter; common constructions are optical, mechanical, and magnetic.

An **optical** encoder consists of one or more LEDs illuminating one side of a circular **encoder disk** (mounted to the rotor shaft) with one or more rings (that is, annular rows) of slits (see Figure 3.5a-c), and one or more photodiodes per ring of slits on the opposite side. As a slit passes in front of each photodiode, light from an LED passes through this slit, and the wire attached to that photodiode transmits a logical **pulse** (transitioning from 0 to 1, and shortly later back to 0). Reflective optical encoders also exist, in which the LED(s) and the phototdiode(s) are mounted on the same side of the encoder disk.

A **mechanical** encoder energizes metal patches on a rotating disk, arranged in the same patterns as shown in Figures 3.5b or c (power is transferred to the disk using brushes and commutators; seen §3.2). Stationary brushes slide on and off these metal patches (also acting as commutators), again sending signals down the wires attached to these brushes. Due to friction and wear issues, mechanical encoders are only appropriate for shafts that are turned infrequently, and at low speed, such as the rotary selector input on a voltmeter or the volume knob on a stereo. In other applications, optical or magnetic encoders are preferred.

A **magnetic** encoder replaces the slits in a disk with magnets, and the photodiodes with magnetic (Hall effect) sensors, but otherwise again operate according to similar principles. Magnetic encoders are particularly convenient for use in brushless motors (see §3.4.2), which already have a series of permanent magnets mounted, with alternating directions of polarity, to the rotor shaft.

There are two principal types of rotary encoders:

- **incremental** (see §3.3.2), which only count changes to the rotation angle of a shaft from its prior state, and
- **absolute** (see §3.3.3), which directly indicate the absolute phase angle of a shaft, regardless of its prior state.

Optical and magnetic encoders may be of either type; mechanical encoders are generally of the absolute type. Note that **hybrid** ABZ encoders (also discussed in §3.3.2) may be considered as a third category, hybrid encoders are incremental over much of their range, with occasional resets at a known angle.

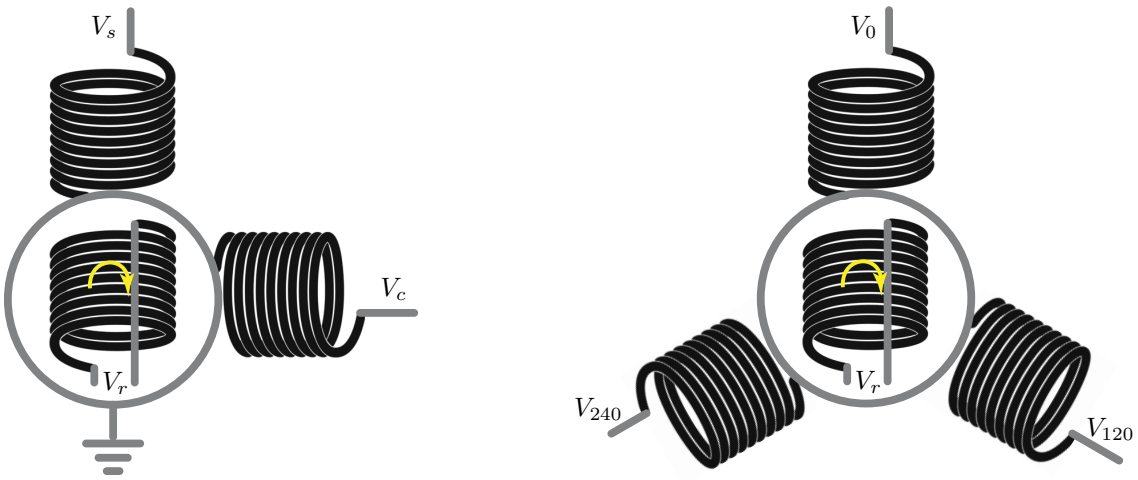


Figure 3.6: Analog devices for determining the angle of a rotating shaft: (a) a **resolver**, and (b) a **synchro**. Both have a (rotating) **reference** winding; a resolver has two stationary identical windings nearby, mounted $90°$ apart, and a resolver has three stationary identical windings nearby, mounted $120°$ apart. In **control transmitter** mode, the (rotating) reference winding is electrically excited sinusoidally, and the induced voltages in the other windings are measured. In **control transformer** mode, the other windings are excited sinusoidally ($90°$ or $120°$ out phase from each other), and the induced voltage in the (rotating) reference winding is measured. In either case, an AC signal V_r, carried over two wires, must be transmitted either to or from the rotating shaft (this is the driving signal in control transmitter mode, and the measured signal in control transformer mode). This may be done using brushes and commutators, or using a rotary transformer, as discussed in §3.2.

3.3.1 Resolvers and synchros

Resolvers and synchros (see Figure 3.6) are analog devices capable of determining the angle of a rotating shaft.

In **control transmitter** mode, the rotating reference winding is driven by $V_r = \overline{V}\cos\phi$ where $\phi(t) = \omega_e\, t$.
• In a resolver, this generates voltages in the two stationary windings of $V_s = \eta\, V_r\, \sin\theta$ and $V_c = \eta\, V_r\, \cos\theta$ for some efficiency metric η. Regardless of the value of η, the shaft angle θ is then given simply from the instantaneous output voltages as $\theta(t) = \text{atan2}(V_s,\, V_c)$, where atan2 is defined in (B.1).
• In a synchro, this generates voltages in the three stationary windings of $V_0 = \alpha\, V_r\, \sin 0° + \beta\, V_r\, \cos 0° + \gamma\, V_r$, $V_{120} = \alpha\, V_r\, \sin 120° + \beta\, V_r\, \cos 120° + \gamma\, V_r$, and $V_{240} = \alpha\, V_r\, \sin 240° + \beta\, V_r\, \cos 240° + \gamma\, V_r$. Solving these equations to determine $\{\alpha, \beta, \gamma\}$ from $\{V_0, V_{120}, V_{240}\}$, it works out that $\gamma \approx 0$ if the stationary coils are nearly identical and $120°$ apart, and we can rewrite the result as $V_0 = c\, V_r\, \sin\theta$ for the shaft angle $\theta(t) = \text{atan2}(\beta, \alpha)$.

In **control transformer** mode, again defining $\phi(t) = \omega_e t$, the stationary windings are driven:
• in a resolver, such that $V_s = \overline{V}\, \sin\phi$ and $V_c = \overline{V}\, \cos\phi$, and
• in a synchro, such that $V_0 = \overline{V}\, \sin\phi$, $V_{120} = \overline{V}\, \sin(\phi + 120°)$, and $V_{240} = \overline{V}\, \sin(\phi + 240°)$.
In either case, this generates a voltage $V_r = \eta\, \overline{V}\, \sin(\phi - \theta)$ in the (rotating) reference winding for some efficiency metric η, where $\theta(t)$ denotes the angle of the shaft from some appropriately-defined reference angle θ_0, and $\omega_s(t) = d\theta(t)/dt$ denotes the rate of rotation of the shaft. Noting that $\dot{V}_r = \eta\, \overline{V}\, [\omega_e \cos(\phi - \theta) - \omega_s \cos(\phi - \theta)]$ and assuming that the $\omega_e \gg \omega_s$ [i.e., that the frequency ω_e of the electrical excitation is much higher than the rate of rotation of the shaft ω_s], it follows, regardless of the value of η, that $\theta \approx \phi - \text{atan2}(\omega_e V_r(t),\, \dot{V}_r(t))$.

Resolvers and synchros are often set up in **torque chains**, in which the (2 or 3) outputs from a "primary" (resolver or synchro), set up in control transmitter mode, are used to drive the (2 or 3) inputs to a "secondary" (resolver or synchro), set up in control transformer mode. If the rotation of the primary device is driven by some process, and the secondary device can rotate freely, as with an analog instrument, and if the reference winding of the secondary device is excited by the same voltage used to excite the reference winding of the primary device, then a torque is generated by this electrical connection to the secondary device that tends to drive the angle of the shaft of the secondary device to match the angle of the shaft of the primary device.

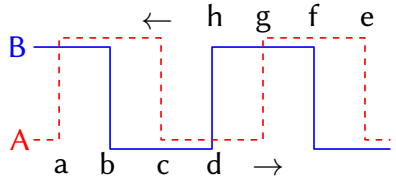


Figure 3.7: Encoder signals from a quadrature encoder, indicating the (dashed) A and (solid) B signals. These signals are generated by a pair of photodiodes placed about $90(1+4i)/n$ degrees apart (for integer i) near an encoder disk with a single ring of n slits. The $\{a, b, c, d\}$ *transitions* (from left to right in the image) indicate one direction of shaft rotation, and the $\{e, f, g, h\}$ transitions (from right to left in the image) indicate the other direction of shaft rotation. For a given n, i may be selected to make the encoder easy to manufacture (e.g., physically placing the two photodiodes around $45°$ to $90°$ apart around the shaft).

3.3.2 Incremental encoders: unidirectional, quadrature, and ABZ

The simplest incremental encoder, called a **unidirectional encoder**, consists of just a single ring of slits (see Figure 3.5a, without the single slit in the second row) and a single LED/photodiode pair. Every time a slit passes in front of the photodiode, a pulse is generated on the wire attached to the photodiode. A counter unit (see §1.5.5) on the MCU increments its counter in response to the rising edge, the falling edge, or both, of these pulses. This approach is only useful for shafts designed to spin in one direction, but such applications are common (in conveyer belts, assembly lines, etc.). The number of slits per revolution in the encoder disk defines the resolution of the encoder, and should be selected based on the maximum speed of rotation of the shaft and the maximum reliable rate of the counter unit. Note that motors (which often operate most efficiently at speeds much higher than needed in a particular application) often have speed-reducing (and, thus, torque-increasing) gearboxes attached. Attaching the encoder disk before (or, after) such a gearbox increases (or, decreases) the effective resolution of the encoder for a given number of slits in the encoder disk. Also, the number of slits per revolution of the encoder disk is limited by practical issues; if this number is made too high, the slits become too narrow to manufacture precisely, the state transitions of the signals from the photodiodes become noisy, and counting them becomes difficult. Thus, depending on the application specifics, mounting the encoder disk before or after the gearbox (if one is present) may be preferred.

We next consider a **quadrature encoder**, built with a single ring of n slits, and *two* photodiodes, denoted A and B. If the two photodiodes are placed $360\,i/n$ degrees apart from each other around the shaft (for integer i), the signals that they generate will be exactly in phase with each other, and the second photodiode will provide no useful new information. If the two photodiodes are placed $180(1+2i)/n$ degrees apart from each other around the shaft, the signals that they generate will be of opposite phase, and again the second photodiode will provide no useful new information. However, if the two photodiodes are placed about $90(1+4i)/n$ degrees apart from each other around the shaft, the signals that they generate will be about $90°$ out of phase (see Figure 3.7a), which as described below is quite useful. Note that, for a given n, i may be selected to make the encoder easy to manufacture (e.g., physically placing the two photodiodes around $45°$ to $90°$ apart around the shaft).

With two signals A and B that are about $90°$ out of phase, we can actually infer the *direction* of rotation by looking at state *transitions* (low-to-high, high-to-low, or both) of one or both logical states, while monitoring the other state, leading to a **bidirectional** incremental encoder. Transitions $\{a, b, c, d\}$ happen only when time flows from left to right in Figure 3.7 (indicating, say, "clockwise" shaft rotation), and transitions $\{e, f, g, h\}$ happen only when time flows from right to left (indicating "anticlockwise" rotation):

a) A transitions from low to high when B is high,
b) B transitions from high to low when A is high,
c) A transitions from high to low when B is low,
d) B transitions from low to high when A is low,
e) A transitions from low to high when B is low,
f) B transitions from low to high when A is high,
g) A transitions from high to low when B is high,
h) B transitions from high to low when A is low.

Such a bidirectional encoder can be set up to watch for 1, 2, or all 4 of the transitions in each of the above two groups (and, to increment or decrement its counter as appropriate when these transitions are detected), referred to as **1x**, **2x**, and **4x** modes. The capability of using 4x mode (that is, of updating the counter $4n$ times per full wheel revolution) lends this sensor its common name as a quadrature encoder.

An **ABZ encoder** combines a (bidirectional) quadrature encoder as described above with a third photodiode mounted to detect the passage of a single narrow slit on a second ring in the encoder disk, as shown in Figure 3.5a. This third signal gives an absolute reference point on the shaft angle during each full rotation, and can be useful to initialize an absolute reference angle when restarting the system, and to correct for possible missed encoder counts during normal operation. The ABZ encoder is thus actually a hybrid between incremental and absolute encoder designs.

3.3.3 Absolute encoders: binary, Gray, and commutation

In order determine with high resolution the absolute rotation angle of a shaft, consider again the use of an optical encoder, but now with an encoder disk with multiple rings, and a radially-aligned row of photodiodes to read the binary values (slits) in each ring. With 7 rows of slits, it is straightforward to see that $2^7 = 128$ distinct positions can be read off directly; conveniently, the sensed signal will immediately be in simple binary order if a **binary encoder** pattern, such as that illustrated in Figure 3.5b, is used.

However, there is a significant problem with the above approach. Due to manufacturing inaccuracies (specifically, minor misalignments in the row of photodiodes), during the transition from a single binary number to another (say, between the state 1111111 and the state 0000000) as the encoder the encoder disk turns, some bits will inevitably change slightly before the others, making the rotation of the shaft appear, for a moment or two, to be in a vastly different state than it actually is. These errors can be quite problematical. Note further that the misalignment of the photodiodes can become significantly more severe in a system as it ages, and inevitably receives a few substantial bumps and knocks. As a result, *binary encoders should never actually be used*.

The solution to the above problem is to use a **Gray encoder**. Consider the following reversible transformation: start from an n-bit binary number $b(1)\, b(2)\, \ldots b(n)$, where $b(1)$ denotes the most significant bit (msb) and $b(n)$ denotes the least significant bit (lsb), and define another n-bit binary sequence $g(1)\, g(2)\, \ldots g(n)$, dubbed a **Gray code**, as follows:

$$g(1) = b(1), \quad \text{for} \quad i = 1 : n-1, \quad g(i+1) = b(i) \text{ xor } b(i+1), \quad \text{end} \tag{3.1a}$$

where xor denotes exclusive or[4] (see Table 1.3). The inverse of this operation, it is easy to prove[5], is given by

$$\bar{b}(1) = g(1), \quad \text{for} \quad i = 1 : n-1, \quad \bar{b}(i+1) = \bar{b}(i) \text{ xor } g(i+1), \quad \text{end} \tag{3.1b}$$

The transformation of the first $2^5 = 32$ binary numbers (starting from 0) to Gray code and back, by the above reversible transformations, is listed in Table 3.1. Reading an n-bit Gray code sequence with an optical encoder, rather than reading an n-bit binary sequence, completely eliminates the problem mentioned in the previous

[4] The operation a xor b is equal to 1 (true) if its arguments differ, and is equal to 0 (false) if they are the same.

[5] That is, the reconstructed $\bar{b}(i)$ in (3.1) equals the original $b(i)$ for all i. This may be established with a **proof by induction**: assume, for some i, that $\bar{b}(i) = b(i)$. To show that it follows, for this i, that $\bar{b}(i+1) = b(i+1)$, consider the four possible cases:

(a) If $b(i) = 0$ and $b(i+1) = 0$, then $g(i+1) = b(i)$ xor $b(i+1) = 0$, and $\bar{b}(i+1) = \bar{b}(i)$ xor $g(i+1) = 0$.
(b) If $b(i) = 0$ and $b(i+1) = 1$, then $g(i+1) = b(i)$ xor $b(i+1) = 1$, and $\bar{b}(i+1) = \bar{b}(i)$ xor $g(i+1) = 1$.
(c) If $b(i) = 1$ and $b(i+1) = 0$, then $g(i+1) = b(i)$ xor $b(i+1) = 1$, and $\bar{b}(i+1) = \bar{b}(i)$ xor $g(i+1) = 0$.
(d) If $b(i) = 1$ and $b(i+1) = 1$, then $g(i+1) = b(i)$ xor $b(i+1) = 0$, and $\bar{b}(i+1) = \bar{b}(i)$ xor $g(i+1) = 1$.

In all four cases, $\bar{b}(i+1) = b(i+1)$; i.e., (3.1b) inverts the transformation done by (3.1a). This applies identically for the "base case" $i = 1$ [$b(1) = g(1) = \bar{b}(1)$], and thus, successively, for the case with $i = 2$, and then for $i = 3$, etc., ultimately for all integer $i \geq 1$.

Gray	00000	00001	00011	00010	00110	00111	00101	00100	01100	01101	01111	01110	01010	01011	01001	01000	11000	11001	11011	11010	11110	11111	11101	11100	10100	10101	10111	10110	10010	10011	10001	10000
binary	00000	00001	00010	00011	00100	00101	00110	00111	01000	01001	01010	01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101	10110	10111	11000	11001	11010	11011	11100	11101	11110	11111

Table 3.1: The numbers 0 to $2^5 - 1 = 31$, in binary and Gray code. [Codes: RR_bin2gray.m, RR_gray2bin.m]

paragraph, of some bits changing slightly before other bits when the photodiodes are slightly out of whack. This is a result of the remarkable fact that *only one bit changes at a time when counting through the numbers 0 to* $2^n - 1$*, and looping back to 0, when the numbers are represented using an* n*-bit Gray code*, as illustrated for the $n = 5$ case in Table 3.1. Further, the conversion from Gray code to a corresponding binary representation can be done remarkably quickly using the above algorithm.

Finally, one added benefit of a Gray code encoder is that, even though n bits are still needed to represent all numbers from 0 to $2^n - 1$, the *changes in the lsb happens at half the rate* as the changes in the lsb of the corresponding binary sequence (again, see Table 3.1). As mentioned previously, if the slits become too narrow to manufacture precisely, the transitions of the signals from the photodiodes become noisy, and, practically, sensing them becomes difficult. The severity of this problem is reduced in a Gray code encoder, as, at a given resolution n, the smallest slits are twice as wide as for the corresponding binary encoder, as readily apparent in the 7-bit Gray code encoder disk illustrated in Figure 3.5c.

Finally, consider a **commutation encoder**, built with n pairs of N-S poles mounted (with alternating polarity) to a rotating shaft, and *three* stationary magnetic (Hall effect) sensors, denoted $\{U, V, W\}$, mounted nearby. As illustrated in Figure 3.8, there are two distinct variants on this idea:

• if the sensors are placed about $60(1 + 6i)/n$ degrees apart from each other around the shaft, for integer i, the signals they generate will be about 60° out of phase, and
• if the sensors are placed about $120(1 + 3i)/n$ degrees apart from each other around the shaft, for integer i, the signals they generate will be about 120° out of phase.

Either way, six valid phases of rotation can be uniquely detected[6], denoted $\{a, b, c, d, e, f\}$. The inputs to the three sets of electromagnets of a three=phase BLDC motor (see §3.4.2) may then be synchronized to these six phases by the commutation logic (see §3.4.3) driving these sets of electromagnets in order to coordinate the motor's efficient application of torque on the shaft in the clockwise and anticlockwise directions. As a result of its utility in the commutation of BLDC motors, such an encoder is commonly referred to as a commutation encoder. Note that, since a commutation encoder indicates which of six phases of rotation that a motor shaft is in, without reference to where the shaft was previously, it is classified as an absolute encoder.

[6]The states 101 and 010 are not valid in the 60° variant, and the states 111 and 000 are not valid in the 120° variant.

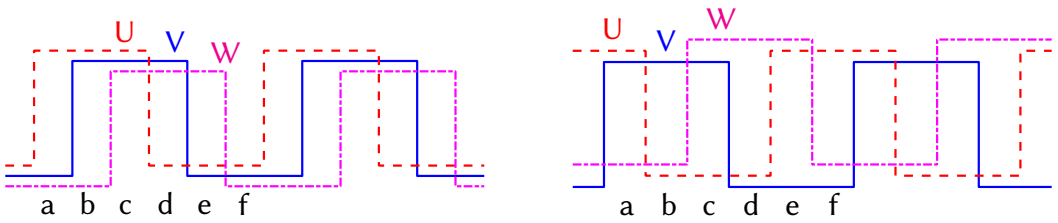


Figure 3.8: Encoder signals from (left) the $60°$ variant, and (right) the $120°$ variant, of a commutation encoder, indicating the (dashed) U, (solid) V, and (dot-dashed) W channels. These signals are picked up by a triplet of magnetic (Hall effect) sensors placed $60(1+6i)/n$ degrees apart (in the $60°$ variant), or $120(1+3i)/n$ degrees apart (in the $120°$ variant), near the $2n$ permanent magnets, of alternating polarity, mounted to the rotating shaft of a BLDC motor. The $\{\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d},\mathsf{e},\mathsf{f}\}$ *phases* (note: not transitions) may be used by six-state logic driving the motor (see §3.4.3) to coordinate the application of torque on the shaft in the clockwise or anticlockwise directions. Again, for a given n, i may be selected to make the encoder easy to manufacture (e.g., physically placing the two photodiodes around $30°$ to $60°$ apart around the shaft).

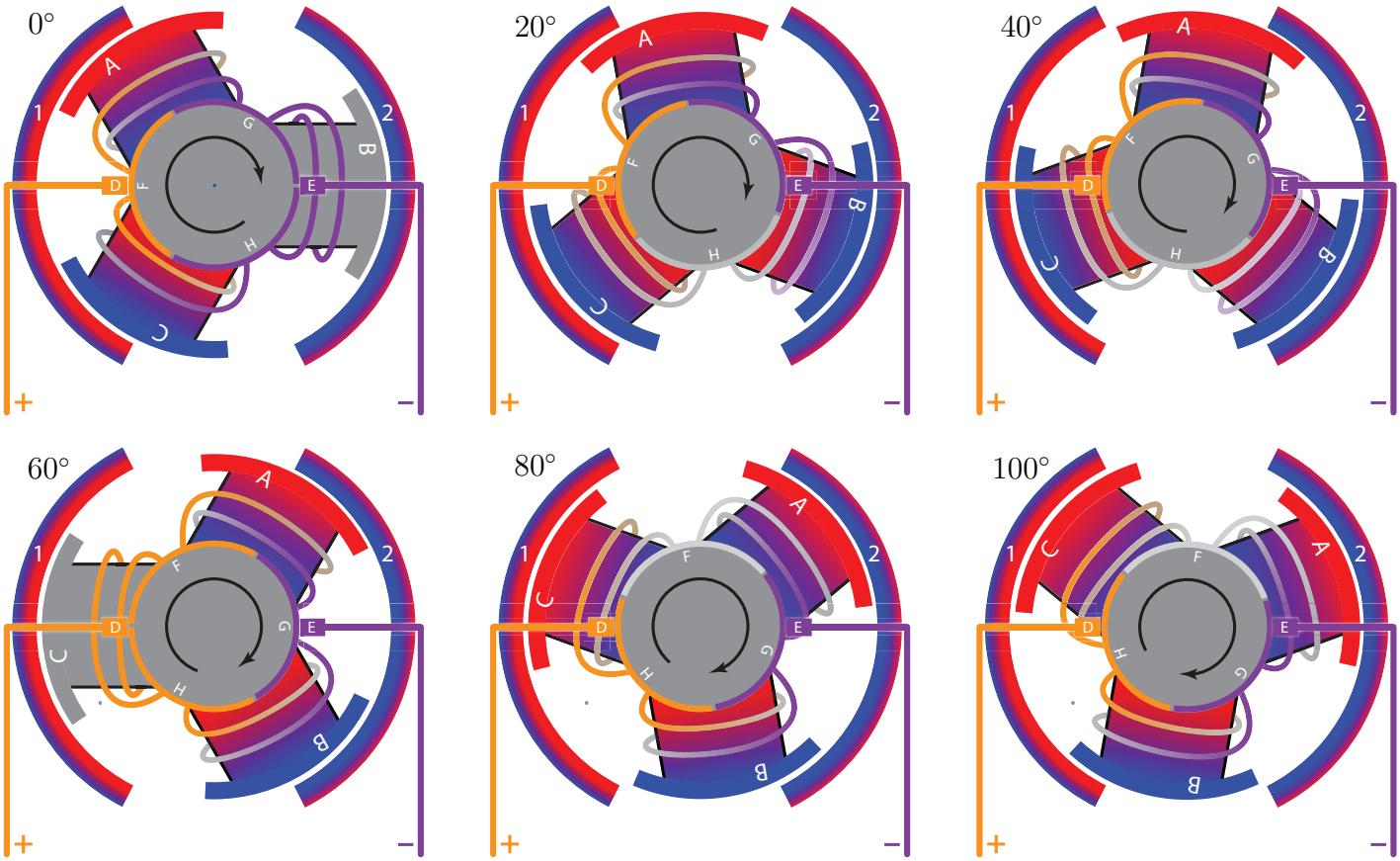


Figure 3.9: Principle of operation of a **brushed DC** (**BDC**) motor, in which the **electromagnets {A,B,C} rotate** and the **permanent magnets {1,2} are stationary** (cf. Figure 3.10); the pattern illustrated in this case repeats after 120° of rotation. The appropriate coordination of the energizing of the coils driving the electromagnets with the rotation angle of the shaft is achieved via carbon brushes {D,E} dragging against split-ring commutators {F,G,H}; see Figure 3.2. The outer ends of the electromagnets that are red (north) are repelled from the inner surfaces of the permanent magnets that are red (north) and are attracted to the inner surfaces of the permanent magnets that are blue (south); vice-versa for the outer ends of the electromagnets that are blue. At any instant, all of the energized electromagnets tend to torque the shaft in the clockwise direction.

3.4 Brushed DC (BDC) and Brushless DC (BLDC) Motors

Though there are many different types of electric motors (see also §3.6), the two main paradigms that drive small robotic systems today are radial-flux brushed DC (BDC) and brushless DC (BLDC) motors. Their essential principles of operation are illustrated in Figure 3.9 and Figures 3.10-3.11, respectively; both use **electromagnets** to generate alternating magnetic fields in the presence of **permanent magnets**[7], thereby generating torques which tend to rotate a shaft. Note that, in electromagnets (aka inductors), when energizing (running a current) through a coil wrapped around a soft iron core, a magnetic field is generated in the core in the direction indicated by the **right-hand** rule (when current is flowing from + to − in coils wrapped around the core in the direction of your fingers, the north pole of the resulting magnetic field is generated in the direction of your thumb). The key to the proper operation of both BDC and BLDC motors is the appropriate coordination of the energizing of the coils that generate such magnetic fields with the rotation angle of the shaft, known as **commutation**, combined with compatible orientation of nearby permanent magnets to effectively produce torque on the shaft.

[7] Collectively, these two paradigms are thus sometimes referred to as permanent-magnet DC (PMDC) motors.

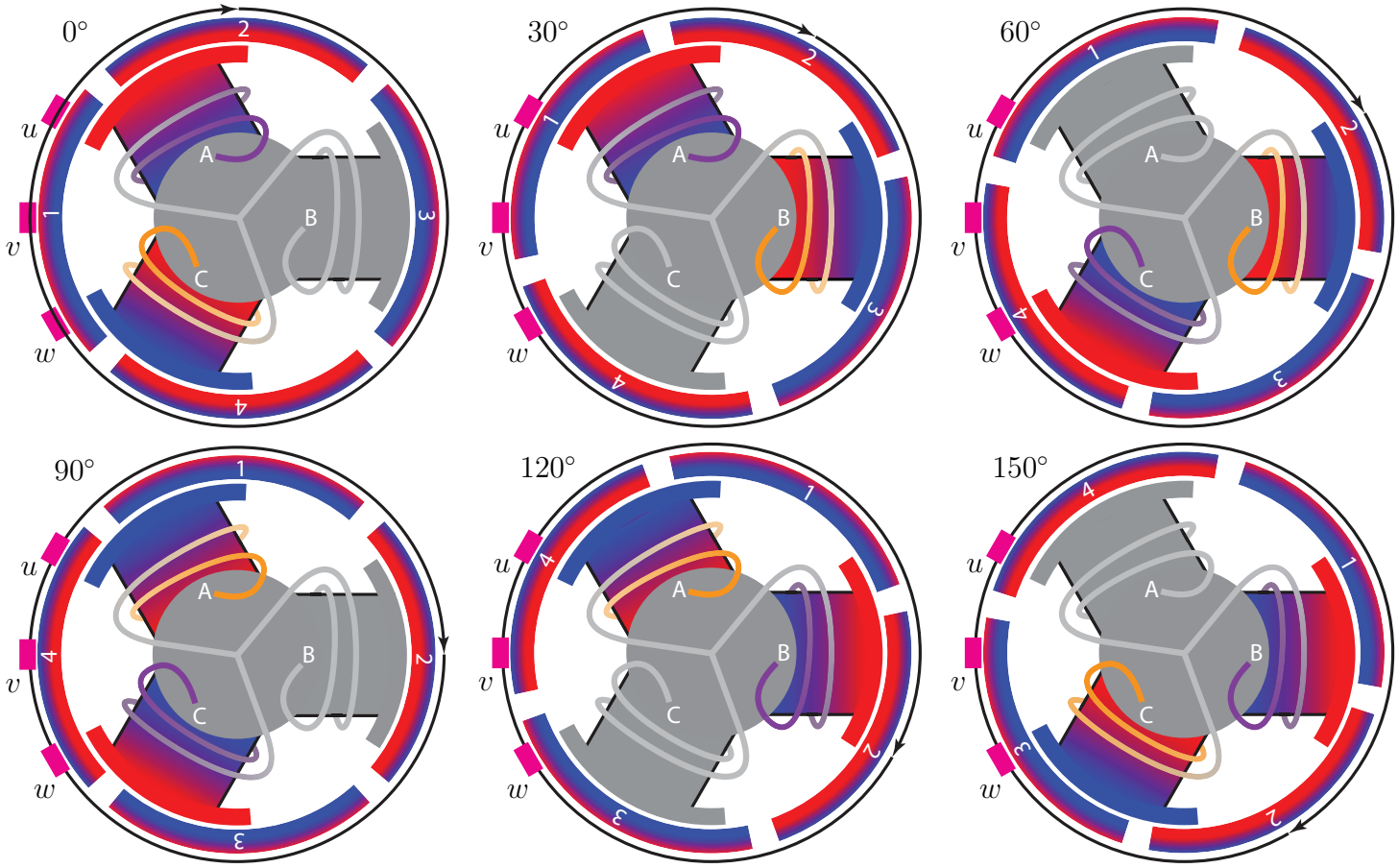


Figure 3.10: Principle of operation of a **brushless DC** (**BLDC**) motor, in which the **permanent magnets {1,2,3,4} rotate** and the **electromagnets {A,B,C} are stationary** (cf. Figure 3.9). The pattern illustrated in this case repeats after $180°$ of rotation. Implementing the appropriate feedback, commutation is achieved electronically.

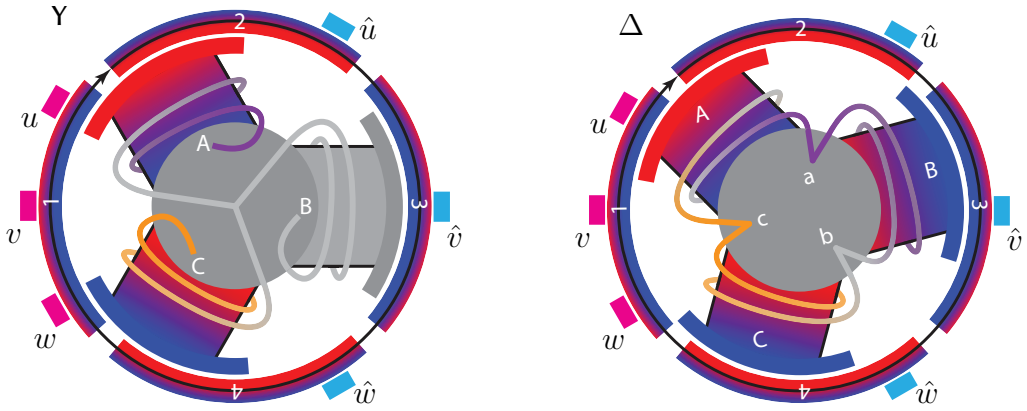


Figure 3.11: Illustration of the Y (Wye) and Δ (Delta) winding connections of BLDC motors. For a given overall configuration, windings, and power supply, the Y winding connection provides more low-speed torque, and the Δ winding connection provides a higher speed. [In theory, the Y and Δ winding connections of BLDC motors provide essentially the same performance; Δ has lower effective resistance, lower K_V, and higher K_T.] At the instants shown, in the Y case, electromagnet A is repelling magnet 2, electromagnet C is attracting magnet 4, and electromagnet B is off, whereas in the Δ case, electromagnet A is attracting magnet 1 and repelling magnet 2, electromagnet B is repelling magnet 3, and electromagnet C is attracting magnet 4, all of which tend to torque the shaft in the clockwise direction. Commutation using the appropriate feedback, using either the $u/v/w$ hall sensors or the $\hat{u}/\hat{v}/\hat{w}$ hall sensors, is discussed in §3.4.3.

The first **BDC** motors were developed by Thomas Davenport (and, others) around 1834, and rapidly became the dominant mechanism to convert electric energy into mechanical motion (and, when run backwards as a **generator**, to convert mechanical motion into electrical energy). With this paradigm, the **electromagnets rotate** and the **permanent magnets are stationary**. As illustrated in Figure 3.9, commutation in BDC motors is achieved mechanically, simply by dragging carbon brushes against split-ring commutators. In the figures shown, note that the brushes are wide enough that, at certain times, they overlap two commutators at a time; if there are more commutators than brushes in a given ring, this does not result in a short circuit. Unfortunately, the carbon brushes upon which the BDC paradigm is based wear out over time.

With the advent of low-cost, fast microcontrollers in the early 21st century, **BLDC** motors are quickly replacing BDC motors in almost all small-scale robotics applications except low-cost toys. With this paradigm, the **permanent magnets rotate** and the **electromagnets are stationary**. As illustrated in Figure 3.10, commutation in BDC motors is achieved electronically, based on feedback (see §3.4.3) indicating the relative rotational position of the electromagnet array with respect to the permanent magnet array. In addition to superior wear characteristics, BLDC motors are much easier to make weatherproof or waterproof; as the electromagnets are stationary in the BLDC paradigm, they can easily be sealed in plastic, while the rotating components can be exposed to the elements, or even submerged in water.

3.4.1 Dynamic modeling of BDC and BLDC motors

To model (in SI units; see §9.1.1-9.1.2) the dynamics of both BDC and BLDC motors, we first consider the following functions of time, averaged over each shaft rotation:

- $V(t)$ is the **voltage** applied to the motor [measured in volts],
- $I(t)$ is the **current** through the motor [measured in amps],
- $\tau(t)$ is the **torque** applied by the motor to the mechanical load [measured in N·m],
- $\omega(t)$ is the **rate of rotation** of the motor shaft [measured in rad/s],

An approximate model of the voltage and torque balances, respectively, of such motors is then given by

$$\textbf{electrical behavior:} \quad V = R\,I + L\,dI/dt + \omega/K_V, \tag{3.2a}$$

$$\textbf{mechanical behavior:} \quad \tau = K_T\,I = J\,d\omega/dt + b\,\omega + C\,\mathrm{sgn}(\omega), \tag{3.2b}$$

where the three RHS terms in (3.2b) model the rotational inertia, viscous friction, and dry friction (see Example 6.3) of the full system (that is, the motor together with its attached load), and where (again, in SI units)

- R is the motor **resistance** [measured in ohms],
- L is the motor **inductance** [measured in henries],
- K_V is the **speed constant** of the motor [measured in (rad/s)/V, with 1 rpm/V $= 2\pi/60$ (rad/s)/V],
- K_T is the **torque constant** of the motor [measured in N·m/A],
- J is the **rotational inertia** of the full system [measured in kg·m^2, with 1 g·cm^2 $= 10^{-7}$ kg·m^2],
- b is the **viscous friction coefficient** of the full system [measured in N·m·s], and
- C is the **dry friction coefficient** of the full system [measured in N·m; see (6.6c)].

The term ω/K_V in (3.2a) is called the motor's **back emf** (electromotive force). Key relations to start with are:

A. at steady conditions (d/dt term zero) with negligible R, $\omega \approx K_V\,V$ (speed is proportional to voltage),
B. $\tau = K_T\,I$ (torque is proportional to current), and
C. when written in SI units, the essential link between (3.2a) and (3.2b) is that $K_V = 1/K_T$.

In general, the dynamic relationship between the applied voltage $V(t)$ and the resulting motor current $I(t)$, torque $\tau(t)$, and rotation rate $\omega(t)$ of the shaft is a result of the coupled equations (3.2a)-(3.2b), and should be modeled as such when doing model-based control of systems driven by BDC and BLDC motors.

	Values at nominal voltage		
1	Nominal voltage	V	6
2	No load speed	rpm	9630
3	No load current	mA	29.5
4	Nominal speed	rpm	7390
5	Nominal torque (max. continuous torque)	mNm	4.81
6	Nominal current (max. continuous current)	A	0.84
7	Stall torque	mNm	20.1
8	Stall current	A	3.42
9	Max. efficiency	%	83

	Characteristics		
10	Terminal resistance	Ω	1.76
11	Terminal inductance	mH	0.106
12	Torque constant	mNm/A	5.9
13	Speed constant	rpm/V	1620
14	Speed/torque gradient	rpm/mNm	482
15	Mechanical time constant	ms	20.5
16	Rotor inertia	gcm²	4.07

Figure 3.12: Specifications of the Maxon 110117 BDC motor.

For any BDC or BLDC motor, the unknown parameters in the model (3.2a)-(3.2b), as discussed above, may be determined using a home-built **dynamometer**; that is, by repeatedly accelerating and decelerating a flywheel of known rotational inertia (carefully attached to the motor shaft with low-friction load bearings) while monitoring the rate of rotation of the shaft with encoders, then performing a least squares fit.

More simply, the parameters in (3.2a)-(3.2b) may be estimated from some values that may be either be measured or (for motors made by high-end motor manufacturers) referenced in the corresponding device datasheet. As a typical example, consider the representative motor specifications in Figure 3.12, which indicate the following three conditions when operating the motor at $V_o = 6$ V:

- a "stall" condition of $\omega_{\text{stall}} = 0$ in which $I_{\text{stall}} = 3.42$ A and $\tau_{\text{stall}} = 0.0201$ N·m;
- a "nominal" speed of $\omega_{\text{nom}} = 7390 \cdot (2\pi/60) = 774$ rad/s when $I_{\text{nom}} = 0.84$ A and $\tau_{\text{nom}} = 0.00481$ N·m;
- a "no-load" speed of $\omega_{\text{no-load}} = 9630 \cdot (2\pi/60) = 1008$ rad/s when $I_{\text{no-load}} = 0.0295$ A and $\tau_{\text{no-load}} = b\,\omega + C$.

This datasheet also lists an efficiency of 83% at the "nominal" conditions, a terminal resistance of $R = 1.76$ ohms, a terminal inductance of $L = 1.06 \cdot 10^{-4}$ henries, and a rotor inertia (assuming no applied load) of $J = 4.07 \cdot 10^{-7}$ kg· m^2. From these values, we can compute the following characteristics:

- $K_{T,\text{stall}} = \dfrac{\tau_{\text{stall}}}{I_{\text{stall}}} = 5.88$ mN·m/A and $K_{V,\text{stall}} = \dfrac{1}{K_{T,\text{stall}}} = \dfrac{170.15}{2\pi/60} = 1625$ rpm/V at $\omega_{\text{stall}} = 0$;
- $K_{T,\text{nom}} = \dfrac{\tau_{\text{nom}}}{I_{\text{nom}}} = 5.73$ mN·m/A and $K_{V,\text{nom}} = \dfrac{1}{K_{T,\text{nom}}} = \dfrac{174.64}{2\pi/60} = 1668$ rpm/V at $\omega_{\text{nom}} = 7390$ rpm;
- $K_{V,\text{no-load}} = \dfrac{\omega_{\text{no-load}}}{V_o - R \cdot I_{\text{no-load}}} = \dfrac{169.47}{2\pi/60} = 1618$ rpm/V at $\omega_{\text{no-load}} = 9630$ rpm.

Note that these computations are largely consistent with characteristics 12 and 13 reported in Figure 3.12, and indicate that K_T and K_V are remarkably constant over the entire range $0 \le \omega \le \omega_{\text{no-load}}$.

The C parameter, which can not be determined accurately from the information provided in Figure 3.12, is essentially insignificant for large ω. Note that dry friction can lead to a troublesome **stick-slip** behavior at low speeds (see, e.g., Example 6.3), which may be effectively overcome in DC motors by driving them with an input voltage incorporationg **pulse-width modulation** (PWM; see Example **??**). Note also that the information provided, at $\omega_{\text{nom}} = 774$ rad/s $= 7390$ rpm, is consistent with $b = \tau_{\text{nom}}/\omega_{\text{nom}} \approx 6.2 \cdot 10^{-6}$ N·m·s.

The speed/torque gradient is simply ω/τ at equilibrium conditions, which for small ω is approximately equal to $(V/I) \cdot K_V/K_T = R \cdot K_V/K_T$. For the motor specified in Figure 3.12, this equals $1.76 \cdot 1620/5.9 \approx 483$ rpm/(mN·m), as reported as characteristic 14. The speed/torque gradient reflects, in a way, the overall "motor strength"; that is, it quantifies how sensitive (or, for small speed/torque gradient, how insensitive) the equilibrium motor speed ω is to variations in the equilibrium load torque τ.

The electrical time constant related to (3.2a) is $t_e = L/R$, and reflects how quickly the current in the motor increases in response to a step change in the input voltage, neglecting any changes in ω. For the motor in Figure 3.12, $t_e = (1.06 \cdot 10^{-4})/1.76 \approx 0.06$ ms, and is essentially negligible.

The mechanical time constant is the time it takes for the motor to accelerate from $\omega = 0$ to $\omega = 0.63 \cdot \omega_{\text{no-load}}$ after a step increase in the input voltage from $V = 0$ to $V = V_o$. It may be approximated as $t_m = J \cdot R/K_T^2$. For the motor in Figure 3.12, $t_m = (4.07 \cdot 10^{-7})(1.76)/(5.9 \cdot 10^{-3})^2 \approx 20.6$ ms, as reported as characteristic 15.

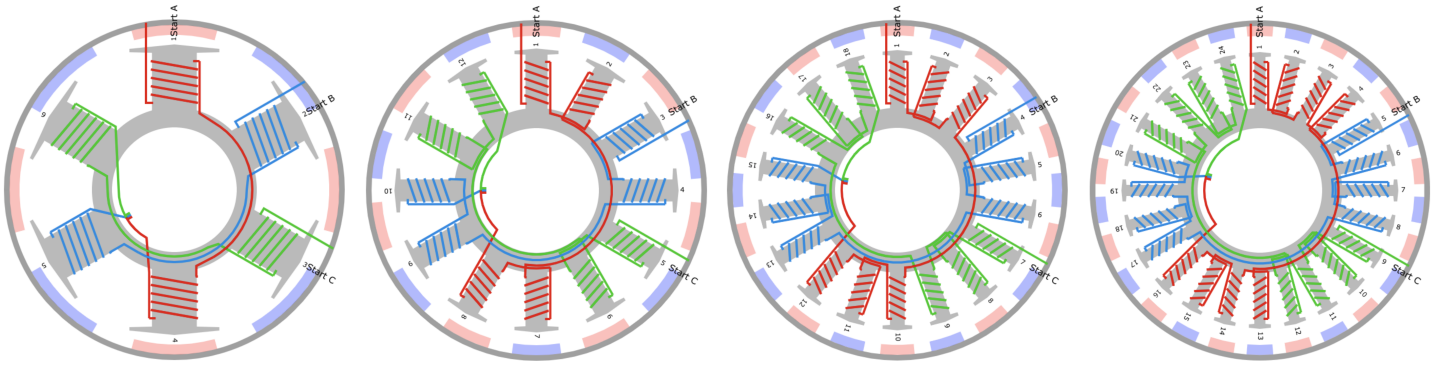


Figure 3.13: Some efficient outrunner BLDC (brushless) motor configurations (6s8p, 12s14p, 18s20p, 24s26p), shown with Y winding connections. In BLDC motors, the electromagnets are stationary, and the permanent magnets rotate with the shaft. Commutation (that is, coordination of the electrical signals causing the motor to turn) is thus achieved electronically rather than mechanically. The windings of these particular three-phase motors are given by $(ABC)^2$, AabBCcaABbcC, $(AaABbBCcC)^2$, and AaAabBbBCcCcaAaABbBbcCcC (see text). Images generated via the convenient online tool available at http://www.bavaria-direct.co.za/.

3.4.2 BLDC motor design

A BLDC motor consists of n_s stationary[8] electromagnets, and n_p permanent magnets[9] (of alternating polarity) attached to a rotating shaft (see Figure 3.10). Both sets of magnets are placed in a circular arrangement around the shaft, with a small air gap between them. The (rotating) permanent magnets may be placed to the inside [called an **inrunner** configuration] or to the outside [called an **outrunner** configuration] of the (stationary) electromagnets. BLDC motor designs are commonly denoted n_s s n_p p; a few examples are given in Figure 3.13.

As the electromagnets are stationary in brushless (BLDC) motors, they are commutated electronically; this is in contrast with brushed (BDC) motors, in which the electromagnets rotate and the permanent magnets are stationary, and for which mechanical commutation (with brushes and split-ring commutators) is used instead (see Figure 3.9). **Three-phase** commutation of the electromagnets, in which the n_s stationary electromagnets are electrically arranged into 3 equal-sized groups that are powered 120° out of phase from each other, are by far the most common, and form the focus of our study[10]. The algorithms used to coordinate the electronic commutation of three-phase BLDC motors with the rotation of the shaft is discussed in §3.4.3.

Recall from Figure 3.11 that the ends of the three sets of electromagnets may be tied together in a Y (aka wye) configuration, as also done in Figure 3.13, or arranged in a Δ (aka delta) configuration. For a given set of windings, the Y configuration has $\sqrt{3}$ times more torque per amp, and the Δ configuration has a higher top speed; if a motor with a Y configuration is rewound (with more turns of thinner wire), the performance of the Δ configuration can be made to be roughly equivalent.

The number of "slots" n_s (a multiple of 3), and the number of "poles" n_p (a multiple of 2) are important in BLDC motor design[11]. We focus next on selecting n_s and n_p appropriately, and optimizing the corresponding winding pattern of the three phases, both of which require closer consideration.

[8] The maximum number of electromagnets may be determined by counting the **slots** through which the wires are threaded when winding the electromagnets; n_s is thus often said to indicate the number of slots through which these wires are thread.

[9] Rare Earth magnets made from Neodymium (Nd) or Samarium Cobalt (SmCo) are most commonly used in BLDC motors.

[10] Note that five-phase (and, even, seven-phase and nine-phase) BLDC motor designs are also possible, with improved efficiency and fault tolerance, but with substantially more complex coordination circuitry.

[11] We focus the present discussion on the case in which every adjacent pair of "slots" is used to wrap a coil, thus making an electromagnet out of the (iron-core) post in between each pair of slots (see Figure 3.13). This is sometimes said to be a "two-layer" configuration in which, looking down in any individual slot, two coils are visible: those going around the post on one side, and those going around the post on the other. This is in contrast to the "one-layer" configurations sometimes encountered, in which, looking down in any individual slot, only one coil is visible, with alternating posts not coiled to form electromagnets.

To make maximum use of the volume that the motor occupies for torque production, n_s and n_p should be close to the same, but not equal. If $n_s = n_p$, then at certain rotation angles all the electromagnets and permanent magnets are aligned, and the motor can not self start; this configuration should thus be avoided.

For a given n_s and n_p for a three-phase motor, the following characteristics are commonly defined:

$$\text{slot/pole ratio:} \qquad q = n_s/(3n_p), \tag{3.3a}$$

$$\text{pole units:} \qquad u = \gcd(n_s, n_p), \tag{3.3b}$$

$$\text{coils per phase per pole unit:} \qquad z = n_s/(3u) \quad \Leftrightarrow \quad 3\,u\,z = n_s, \tag{3.3c}$$

$$\text{cogging steps:} \qquad c = \text{lcm}(n_s, n_p), \tag{3.3d}$$

where gcd denotes the greatest common divisor, and lcm denotes the least common multiple.

To keep n_s close to but not equal to n_p for efficient use of the motor volume, as suggested previously, a **slot/pole ratio** in the range $0.25 \leq q \leq 0.5$ is generally advised.

The number of **pole units** u is the maximum number of electromagnets that align with a permanent magnet at any moment. The loading on the motor is said to be **symmetric** when $u > 1$; configurations that are not symmetric should be avoided, as the torque produced by the electromagnets would cause the motor to wobble. The quantity z measures the number of **coils per phase per pole unit**. The windings are **balanced** when z is an integer; configurations which are not balanced should also be avoided. The values of u and z for several recommended $\{n_s,\ n_p\}$ combinations is given in Table 3.2, directly eliminating from consideration: (a) those with $n_s = n_p$, (b) those regions outside the recommended slot/pole ratio range of $0.25 \leq q \leq 0.5$, (c) those without symmetry, and (d) those that are out of balance. Also listed in Table 3.2 is the electrical excitation frequency $f_e = f_s n_p/2$ associated with shaft rotation at $f_s = 100\,\text{Hz} = 6000\,\text{rpm}$.

To illustrate, the four designs in Figure 3.13 each have $u = 2$ pole units (and are thus symmetric), and the values of z for these four designs are 1, 2, 3, and 4 respectively (and each are, thus, balanced).

The quantity c counts the number of times that permanent magnets align directly with electromagnets during an entire revolution of the shaft (as mentioned previously, u permanent magnets simultaneously align with electromagnets each time this happens). This quantity also indicates the number of times that the torque of the BLDC reaches a minimum (and, a maximum) during one complete revolution of the shaft, dubbed the **cogging steps** of the motor. The larger c is, the smaller the amplitude of this **torque ripple**. The worst torque ripple is given for an equal number of slots and poles, for which $c = n_s = n_p$, again suggesting that this condition should be avoided. Values of c for several recommended $\{n_s,\ n_p\}$ combinations is given in Table 3.3.

We next consider how the individual electromagnets of a brushless motor should be wound. Denote the three electrical phases driving the BLDC motor as $\{\text{A}, \text{B}, \text{C}\}$, and let $\{\text{A}, \text{a}, \text{B}, \text{b}, \text{C}, \text{c}\}$ denote windings of these phases, where uppercase denotes clockwise (CW) windings, and lowercase denotes anticlockwise (ACW) windings. A BLDC motor configuration with 6 clockwise windings (with $u = 2$ and $z = 1$) may thus be denoted ABCABC, or more compactly as $(\text{ABC})^2$. Defining the (dLRK) winding as AabBCcaABbcC, and also defining

(i) as ABbcaABCcabBCAabcC,
(ii) as $(\text{AaABbBCcC})^2$,
(iii) as $(\text{AabBCcaABbcC})^2$ = $(\text{dLRK})^2$,
(iv) as AaAabBbBCcCcaAaABbBbcCcC,
(v) as $(\text{AaABbBCcC})^2$,
(vi) as ABCcabcaABCABbcabcCABCAabcabBC,
(vii) as AaABbcCcaABbBCcaAabBCcCAabBbcC,
(viii) as $(\text{AaAaABbBbBCcCcC})^2$,

recommended winding patterns that maximize the torque output for several recommended $\{n_s,\ n_p\}$ combinations are given in Table 3.4. Note in particular that, for those $\{n_s, n_p\}$ combination with $z = 1$, the symmetric answer must simply be $(\text{ABC})^u$, by inspection. Further, $(\text{ACB})^u$ is just $(\text{ABC})^u$ operating in reverse.

The remaining winding patterns listed above may be motivated fairly simply via symmetry arguments on a case-by-case basis, and are not nearly as mysterious as they might at first look. For example, consider first the 24s26p design, rotated just a couple of degrees from the configuration shown at right in Figure 3.13, such

$n_p \rightarrow$ / $n_s \searrow$	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34
6	2, 1		**2, 1**													
9		3, 1			3, 1											
12			4, 1	2, 2		**2, 2**	4, 1						$q<0.25$			
15				5, 1					5, 1							
18					6, 1	2, 3	2, 3		**2, 3**	2, 3	6, 1					
21						7, 1							7, 1			
24							8, 1		4, 2	2, 4		**2, 4**	4, 2		8, 1	
27								9, 1			3			3		
30		$q>0.5$							10, 1	5		2, 5	2, 5		**2, 5**	2, 5
33										11, 1						
36											12, 1	2, 6	4, 3	6, 2	4, 3	2, 6
f_e	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700

Table 3.2: Pole units and coils per phase per pole unit, $\{u,\ z\}$, of 3-phase motors with recommended combinations of [left] the number of slots n_s and [top] the number of poles n_p. Non-recommended regions include:
those with $n_s = n_p$, which are not self starting, and have relatively poor cogging numbers,
those with a slot/pole ratio $q = n_s/(3n_p)$ outside the range $0.25 \leq q \leq 0.5$, which are inefficient,
those without symmetry (that is, those with $u = 1$), which wobble, and
those that are electrically out of balance, with more windings on some phases than others.
Also listed is the electrical excitation frequency f_e associated with shaft rotation at $f_s = 100\,\text{Hz} = 6000\,\text{rpm}$.
Code to generate and extend Tables 3.2, 3.3, 3.5: RR_BLDC_design.m.

$n_p \rightarrow$ / $n_s \searrow$	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34
6	12		**24**													
9		18			36											
12			24	60		**84**	48									
15				30					60							
18					36	126	144		**180**	198	72					
21						42							84			
24							48		120	264		**312**	168		96	
27								54			216			270		
30									60	330		390	420		**480**	510
33										66						
36											72	468	252	180	288	612

Table 3.3: Cogging steps c (larger is smoother) of 3-phase motors for recommended $\{n_s,\ n_p\}$ combinations.

$n_p \rightarrow$ / $n_s \searrow$	4	6	8	10	12	14	16	18	20	22	24	26	28
6	$(ABC)^2$		$(ABC)^2$										
9		$(ABC)^3$			$(ABC)^3$								
12			$(ABC)^4$	(dLRK)		(dLRK)	$(ABC)^4$						
15				$(ABC)^5$							$(ABC)^5$		
18					$(ABC)^6$	(i)	(ii)				(ii)	(i)	$(ABC)^6$
21						$(ABC)^7$							$(ABC)^7$
24							$(ABC)^8$				(iii)	(iv)		(iv)	(iii)
27								$(ABC)^9$				(v)	
30											$(ABC)^{10}$	(vi)		(vii)	(viii)

Table 3.4: Recommended winding configurations of 3-phase motors for recommended $\{n_s,\ n_p\}$ combinations. $(ABC)^2$ denotes ABCABC, etc., where {A,B,C,a,b,c} denote (uppercase) CW and (lowercase) ACW winds of phases A, B, and C. Winding (dLRK) denotes AabBCcaABbcC; windings (i) through (viii) defined in the text.

$n_p \rightarrow$ / $n_s \searrow$	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34
6	0.866		**0.866**													
9		0.866			0.866											
12			0.866	0.933		**0.933**	0.866									
15				0.866					0.866							
18					0.866	0.902	0.945		**0.945**	0.902	0.866					
21						0.866							0.866			
24							0.866		0.933	0.949		**0.949**	0.933		0.866	
27								0.866			0.945			0.945		
30									0.866	0.874		0.936	0.951		**0.951**	0.936
33										0.866						
36											0.866	0.867	0.902	0.933	0.945	0.953

Table 3.5: Fundamental winding factor k_{w1} (larger is more powerful) for recommended n_s and n_p combinations.

that there is (pink) permanent magnet exactly halfway between the (CW-wound) electromagnet 7, and the (ACW-wound) electromagnet 6. If the B phase is sinusoidal, and is phased such that the magnetic field at electromagnets 6 and 7 (which are of opposite polarity) peak at this moment, then a maximum force will be exerted on the (pink) permanent magnet halfway in between (ditto on the blue permanent magnet on the opposite side). Looking around the circumference at the other permanent magnets that, at this instant, are closest to being halfway between two electromagnets suggests that the electromagnet pairs {5,6} and {7,8} should also peak at roughly the same time (and, again, be of opposite handedness); again, ditto on the opposite side. By symmetry, this suggests that, for the 24s26p design, the $z = 4$ electromagnets of each phase, in each of the $u = 2$ pole units, should be placed next to each other, and with opposite handedness. The handedness at the intersections between the different sets of phases in each pole unit (that is, ab, BC, ca, AB, bc, CA) is then selected in the sense that continues to apply torque in the same direction, finally arriving at winding (iv). The other winding patterns listed above (and, those for larger $\{n_s, n_p\}$ configurations) may be reasoned similarly.

A more detailed discussion of the electromagnetic forces within a BLDC motor is beyond the scope of the present discussion. A definitive text on this subject is Hendershot & Miller (2010); we will simply state here the primary definitions that arise in this analysis, leaving derivation and detailed discussion to this text:

$$\text{slot-pitch angle:} \quad \gamma_s = \pi n_p / n_s = \pi/(3q), \tag{3.4a}$$

$$\text{coil-span angle:} \quad \epsilon = \pi - \gamma_s, \tag{3.4b}$$

$$\text{pitch factor, aka coil-span factor:} \quad k_{pn} = \cos(n\epsilon/2), \tag{3.4c}$$

$$\text{distribution factor:} \quad k_{dn} = \sin(n\sigma/2)/[z\sin(n\sigma/(2z))] \quad \text{where} \quad \sigma = \pi/3, \tag{3.4d}$$

$$\text{winding factor:} \quad k_{wn} = k_{dn} k_{pn}. \tag{3.4e}$$

As discussed in the above-mentioned text, the **fundamental winding factor** k_{w1} (that is, k_{wn} with $n = 1$), as defined by (3.3)-(3.4), is proportional to the total torque that a BLDC motor can generate. Generally, the higher k_{w1} is the better; note that $0.866 \leq k_{w1} < 1$ for the recommended BLDC motor designs listed in Table 3.5.

In summary, all of the configurations listed in Tables 3.2-3.5 are viable three-phase BLDC designs, and may be reached by the motor designer for different applications, depending on the relative importance placed on:

- simplicity of construction, for which small n_s and small n_p are preferred,
- efficiency, for which a large fundamental winding factor k_{w1} is preferred, and
- smoothness of operation, for which a large number of cogging steps c is preferred.

As highlighted in Tables 3.2-3.5, some particularly good tradeoffs between simplicity of construction, efficiency, and smoothness of operation include the **6s8p**, **12s14p**, **18s20p**, **24s26p**, and **30s32p** designs, with windings of $(ABC)^2$, (dLRK), (ii), (iv), and (viii), respectively, as defined above (see Figure 3.13).

mechanical rotation angle	center range	$0°$ $-15° - 15°$	$30°$ $15° - 45°$	$60°$ $45° - 75°$	$90°$ $75° - 105°$	$120°$ $105° - 135°$	$150°$ $135° - 165°$
electrical rotation angle	center range	$0°$ $-30° - 30°$	$60°$ $30° - 90°$	$120°$ $90° - 150°$	$180°$ $150° - 210°$	$240°$ $210° - 270°$	$300°$ $270° - 330°$
signals from Hall sensors	$u/v/w$ $\hat{u}/\hat{v}/\hat{w}$	$1/1/1$ $0/1/0$	$1/1/0$ $0/1/1$	$1/0/0$ $0/0/1$	$0/0/0$ $1/0/1$	$0/0/1$ $1/0/0$	$0/1/1$ $1/1/0$
A/B/C or a/b/c		L/Z/H	L/H/Z	Z/H/L	H/Z/L	H/L/Z	Z/L/H
Field at edge of electromagnets	Y Δ	N/-/S N/s/s	N/S/- n/S/n	-/S/N s/s/N	S/-/N S/n/n	S/N/- s/N/s	-/N/S n/n/S

Table 3.6: Caption goes here...

3.4.3 Commutation in BLDC motors

Hall sensors... (see Table 3.6).

Trapezoidal ("six-state") commutation... simple...
Sinusoidal commutation. minimizes torque ripple
Third harmonic injection, higher torque.
Field oriented control (FOC).

3.5 Servos and Electronic Speed Controllers (ESCs)

3.6 Other types of actuators

many, many

3.6.1 Axial BLDC "pancake" motors

3.6.2 AC motors

Induction motors

3.6.3 Linear actuators and solonoids

3.6.4 Hydraulic and pneumatic actuators

3.6.5 Artificial muscle and electroactive polymers

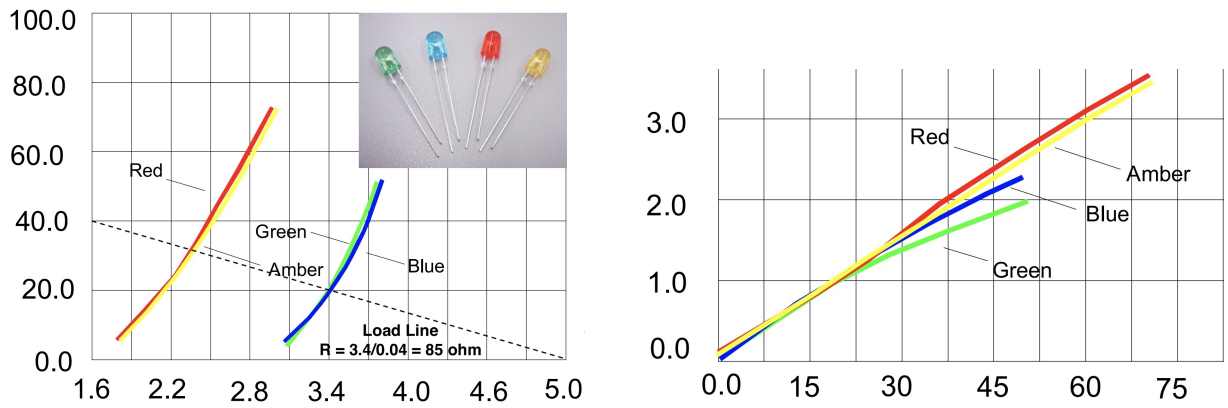


Figure 3.14: (a) Current (in mA) vs applied voltage, and (b) luminous intensity (normalized by value at 20 mA) vs current, of typical small signaling LEDs (see inset) of different colors in the Cree C566D-RFF series.

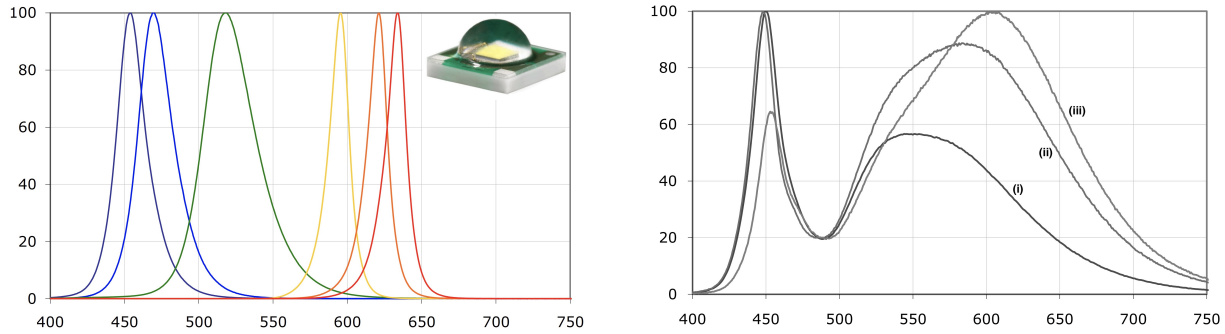


Figure 3.15: Luminous intensity (normalized by peak value) versus wavelength of (a, from left to right) royal blue, blue, green, amber, red orange, and red LEDs, and (b) "white" LEDs [at color temperatures of (i) $5000°$K, (ii) $3700°$K, and (iii) $2600°$K] of typical high-intensity LEDs (see inset) in the Cree XLamp XP-E series.

3.7 Light Emitting Diodes (LEDs), buttons, and touchscreens

The current-voltage relationship of diodes is nonlinear, as shown (and discussed in detail) for general diodes in Figure 9.14, and as highlighted for typical small Light Emitting Diodes (LEDs) in Figure 3.14a. In both cases, the current through the diode is nearly zero until a certain "cut-in" voltage across the device is reached (in the case of LEDs, usually somewhere between 1.5 and 3.5 volts, depending on the chemistry of its construction). To operate properly without burning out[12], an LED is generally driven by an operating voltage higher than its cut-in voltage, and is placed in series with a current-limiting resistor. This series resistor is responsible for regulating the current through the LED, which is approximately proportional its brightness as illustrated in Figure 3.14b. Typical safe operating currents for many small signaling LEDs are in the range of 5 to 40 mA.

To determine the appropriate value for the resistor in order to set the desired current passing through the LED + current-limiting resistor pair, a convenient load line may be drawn on the current-voltage plot of the LED. This loadline starts at zero for an applied voltage of V_{cc} on the horizontal axis, and (since $V = IR$) slopes upward to the left at a slope of $I/V = 1/R$. For the loadline shown in Figure 3.14a, (0.040 A)/(3.4 V)=1/(85 Ω) and thus, for $V_{cc} = 5\,V$ and $R = 85\,\Omega$, the current through a blue or green LED would be about 20 mA, whereas the current through a red or amber LED would be about 32 mA, where the loadline shown intersects the corresponding curves. The LED current for other resistor values is determined similarly.

[12] Beware: if you attach an LED between power and ground without inserting a current-limiting resistor in series, you will burn it out almost immediately; double check the datasheet of the LED you actually select to ascertain its range of safe operating current.

Figure 3.16: Use of tri-state (H/L/Z) logic plus PWM to independently drive two LEDs with a single GPIO.

3.7.1 Tri-state (H/L/Z) logic

As depicted in Figure 3.16, consider now attaching a GPIO pin (see §4.2.1) through both a green LED through R_1 to power, and (in parallel) a red LED through R_2 to ground, taking $R_1 = 150$ ohm and $R_2 = 85$ ohm to run both LEDs (from the Cree C566D-RFF series) at about 20 mA (see Figure 3.14a).
• If the GPIO is set (as an output) to H (that is, to power), then the red LED will illuminate.
• If the GPIO is set (as an output) to L (that is, to ground), then the green LED will illuminate.
If, on the other hand, the GPIO is set as a high-impedance input (which need not be monitored by the MCU), then its value **floats** (i.e., it is determined by the rest of the circuit hooked to this GPIO). In this setting, we refer to this third GPIO state as Z; the circuit then connects, in series between power and ground, the green and red LEDs and two resistors (totaling 85+150=235 ohm); since the cut-in voltage of the red LED is around 1.7 V, and that of the green LED is around 2.9 V, the current through the circuit will be about (5-1.7-2.9 V)/(235 ohm) ≈ 1.7 mA; the LEDs will glow so dimly at this low current that they will essentially appear to be off, thus providing a means to effectively control two LEDs independently with a single GPIO.

Such clever use of Z as a third GPIO state is referred to as **tri-state** (aka three-state) logic. **Pulse width modulation** (**PWM**) can extend even further what is possible using such logic. In the setting described above, we can send a square wave at a frequency greater than 30 Hz that sets the GPIO as H 50% of the time, and as L 50% of the time. The result is that both LEDs blink on and off so quickly that, to our eyes (which effectively average over a timescale of about 1/30 of a second), both LEDs appear to be on 100% of the time, at about 50% of the intensity than they would otherwise appear with the particular current-limiting resistors selected.

3.7.2 Arrays of LEDs or buttons using x/y multiplexing or crossplexing

When building arrays of LEDs or buttons (keys on a keyboard, resistive or capacitive touchscreens, etc.) it is desirable to make the maximum use of a limited number of GPIOs, leveraging the speed of the computer or microcontroller in comparison to the speed of the human with whom it is interacting.

A simple way to build such an array from n GPIO channels, referred to as **x/y multiplexing**, is to use $n_1 = \lfloor n/2 \rfloor$ GPIOs on the left edge and $n_2 = \lceil n/2 \rceil$ GPIOs on the lower edge of a (nearly) square array (e.g., if $n = 15$, taking $n_1 = 7$ and $n_2 = 8$), leading to $n_1 \cdot n_2 = \lfloor n^2/4 \rfloor$ wire intersections at which LEDs or buttons may be placed, as shown in Figure 3.17a. We may then successively (one at a time) energize (as either H or L) each of the n_1 GPIOs controlling the rows, as inputs to the array, and simultaneously either:
- control the state at the other n_2 GPIOs (treated as MCU outputs) to turn on the desired LEDs on that row, or
- read the state at the other n_2 GPIOs (treated as MCU inputs) to determine which buttons are pressed.

An alternative way to build an array from n GPIO channels, referred to as **crossplexing** (aka Charlieplexing), is effectively to rotate the "chessboard" of a large x/y multiplexing array by $45°$, attaching the wires along the rows and columns of this rotated chessboard to the n GPIOs along the left edge of the array, and connecting the nearby wires of rows and columns of the chessboard along the top and bottom edges of the array, as shown in Figure 3.17b. We may then cycle quickly through all n of the GPIOs, energizing (as either H or L) one of them at a time as an input to the array, and controlling (for LEDs) or monitoring (for buttons) all $n - 1$ other GPIOs, similar to before. This approach connects each of the n GPIOs to each of the $n_2 = n - 1$ other GPIOs at exactly 1 wire intersection, leading to $n(n - 1)/2$ intersections at which an LED or button may be placed.

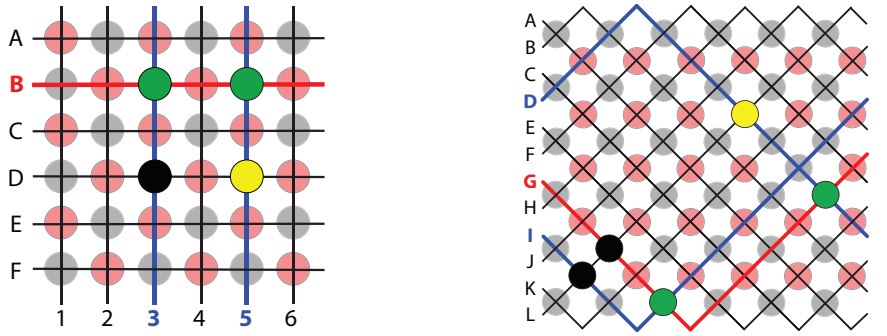


Figure 3.17: Interconnection of $n = 12$ GPIOs as (a) an **x/y multiplexing array**, generating $(n/2)^2 = 36$ intersection points (shown as pink and gray circles), and (b) a **crossplexing array**, generating $n(n-1)/2 = 66$ intersection points. The coordinates of the intersection points are more directly enumerated using an x/y multiplexing array, but crossplexing generates almost twice as many intersection points for large n.

Given an x/y multiplexing or crossplexing array of n GPIOs, as discussed above and illustrated in Figures 3.17a-b, there are several useful things that can be done with each intersection:

a. Put a single LED, together with a current-limiting resistor, between the two wires at each intersection.
   - If using an x/y multiplexing array, to activate any number of LEDs, cycle quickly and repeatedly through the input GPIOs {A,B,C,...} to the array, setting one at a time to **H** while keeping the rest Z or L; at the same time, coordinate the output GPIOs {1,2,3,...} to be **L** for those LEDs corresponding to the rows with the intersections that you want to turn **on**, and leave as Z or H those GPIOs on the rows corresponding to the intersections that you want to stay off[13].
   - If using a crossplexing array, to activate any number of LEDs, cycle quickly through the GPIOs {A,B,C,...}, setting one channel at a time to **H**, while keeping the rest Z; at the same time, coordinate the remaining GPIOs to be **L** for the LEDs at the intersections that you want to turn **on**.

b. Put *two* LEDs (one facing each direction, each with its own current-limiting resistor), between the two wires at each intersection (using either an x/y multiplexing or crossplexing array). The first of the pair at each intersection is illuminated exactly as described above; the second is illuminated by energizing the channels using precisely the opposite logic (swap **H** and **L**).

c. Put a single *button*, in series with a diode, between the two wires at each intersection, again setting one channel at a time to **H**, and monitoring the n_2 other channels to determine which button has been pressed.

The reason that the diode is needed in case c above is to prevent a phenomenon known as **ghosting**, which arises when multiple buttons are pressed at the same time. For example, in Figures 3.17a-b, consider what happens if the 2 green and 1 yellow buttons are pressed at the same time:

• In the case of the x/y multiplexing array in Figure 3.17a, when energizing (as **H**) channel D, both channels 5 and 3 are pulled high; the former because the yellow button is pressed, as expected, but the latter because current flows from the yellow button along channel 5, through button **B5** to channel B, and through button **B3** to channel 3, thus making it appear as if button **D3** is pressed, even if it isn't.
• In the case of the crossplexing array in Figure 3.17b, when energizing (as **H**) channel L, channels D, G, and I are all pulled high; the former because the yellow button is pressed, as expected, but the latter because current flows from the yellow button along channel D, through button **DG** to channel G, and through button **GI** to channel I, thus making it appear as if buttons **LG** and **LI** are pressed, even if they aren't.

[13] Note that, in this case, the use of tri-state logic is in fact unnecessary. Also, in this case, the current-limiting resistors can all be moved adjacent to the output GPIOs {1,2,3,...} from the array.

Simply inserting a diode in series with each of the buttons at each of the intersections prevents all of these spurious effects from arising.

3.8 Displays and other interfaces

TFT, LCD, OLED
- resistive or capacitive touch screens
- eInk
- Persistance of Vision (POV)

Chapter 4

Communication

Contents

4.1 Computer networks

4.1.1 Network protocol stacks

A **protocol stack** is an implementation of a **protocol suite**, which defines the precise set of rules by which computers communicate over a network. Network protocols are modeled as a stack of **layers**, each designed for a specific purpose. There are two primary models of network protocols in common use. The **internet** (aka TCP/IP) model consists of four layers:

4. **Application layer**
3. **Transport layer**
2. **Internet layer**
1. **Link layer**

The **OSI** (Open Source Interconnection) model consists of seven layers:

7. **Application layer**
6. **Presentation layer**
5. **Session layer**
4. **Transport layer**
3. **Network layer**
   a) Subnetwork Access
   b) Subnetwork Dependent Convergence
   c) Subnetwork Independent Convergence
2. **Data link layer**
   - Logical link control (LLC) sublayer
   - Medium access control (MAC) sublayer
1. **Physical layer**
   physical signaling sublayer
   - Physical coding

4.1.2 Wired network topologies

The **topology** of a "wired" computer network (interconnected by electrical cables or optical fibers) is a design criterion that may be tailored to best suit the specific applications for which the system is intended, such as:

- internet connectivity of general-purpose computers,
- centralized fault-tolerant coordination of machines,
- distributed computation of multidimensional PDEs (via spectral, finite difference, or finite element methods),
- weather and climate forecasting [using ensemble methods for forecast uncertainty quantification (UQ)],
- navigation and manipulation of transactional databases (airline ticket sales, large-scale search, ...),
- deep learning for medical diagnostics,
- DNA sequencing, etc.

Links between computers (aka nodes) in any such network can be half-duplex (able to maintain communication in one direction at a time only) or full-duplex (able to maintain communication in both directions simultaneously), with the connecting cables attaching to the nodes via a network interface controller (NIC). If links are physically too long to reliably deliver a signal, one or more repeaters can be used. Possible logical network topologies[1] include the following:

A) The prototypical example of a physically-dedicated **point-to-point** link is a tin-can telephone. Circuit-switching technology, as used in conventional telephony, allows temporary dedicated point-to-point electrical connections to be set up when needed in settings incorporating many nodes.

B) Each pair of nodes in a **daisy chain** is connected via a cable with a NIC at each end; if a received message is not intended for that node, it is simply retransmitted down the chain. A **linear** (with two ends) or **ring** configuration can be used for such a chain; a ring provides, at modest additional cost, an alternate direction to pass any given message, which is useful if the other direction is substantially longer, broken, or busy.

C) Each node may be connected (via a NIC) to a single central **bus** (aka backbone or trunk), and all data carried on the bus can simultaneously be received (or, ignored) by any connected node. A **linear bus** has two endpoints; a **distributed bus** has branches, and thus multiple endpoints. All endpoints of a bus must be terminated (see §??) to prevent reflections.

D) The sending of signals out to all nodes in a **star** configuration is coordinated by a central hub [in which an input signal on one port is repeated as an output signal on all other ports] or switch [in which an input signal (say, on port A) is routed only towards its specific destination (say, on port B), allowing simultaneous communication traffic between the various other ports as needed (say, from port C to D, etc.)].

E) At least some nodes in a **mesh** network have more than two NICs, and thus can themselves act simultaneously as both nodes and switches (note that certain nodes at high-traffic junctions may be replaced by dedicated switches). This topological class is very versatile. A small network with n nodes can be:

E.1) **fully connected**, with a point-to-point link between every pair of nodes [that is, with $n-1$ NICs per node, and $(n-1)!$ total links...which quickly becomes unmanageable for fairly small n], or

E.2) a single d-dimensional (aka dD) **hypercube**, with $n = 2^d$ nodes, d NICs per node, and $d\,2^{d-1}$ total links; with this paradigm, all nodes are within d "hops" from any starting node in the network.

For larger n, a mesh forms some sort of (partially connected) d-dimensional interconnect **grid**, such as:

E.3) A d-dimensional **cartesian grid**, usually with periodic connections in each coordinate direction (thus dubbed a dD **torus**), with $n = n_1 n_2 \cdots n_d$ nodes, and 2^d NICs per node. At the cost of more NICs per node, an dD grid has two important performance advantages over a 1D ring. First, for unstructured data flow, a key performance metric is how many additional nodes are reached per "hop" from any starting

[1] Of course, physically, such networks are usually laid out quite differently (leveraging standardized server racks, etc.).

node in the network; after $r \gg 1$ hops, the number of new nodes reached with one more hop is roughly proportional to r^{d-1} (that is, a dD grid spreads data faster with increased d). Second, with its more numerous, well-structured pathways, an dD grid can more quickly "transpose" a large multidimensional grid that is distributed over the cluster[2]. Note also that circuit switching (see topology A) can be used at times to temporarily partition a single periodically-connected grid into a number of smaller periodically-connected grids to better run smaller jobs.

E.4) An d-dimensional **noncartesian grid**, also with periodic connections, formed by a rare sphere packing (see *RP*), with fewer NICs per node for a given dimension d than a cartesian grid[3]. Examples include the 2D uniform hexagonal tiling, with 3 NICs per node, the 3D diamond packing D_3^+, with 4 NICs per node, and the D_d^+ **hyperdiamond** packing (a d-dimensional generalization), with $d+1$ NICs per node.

E.5) An **unstructured 2D grid** interconnecting many computers that are sparsely separated over a large physical area, such as a factory floor.

Networks may also be arranged logically as a **hybrid** combination of the above basic topologies, such as:

E.6) An **extended star**, given by a star (topology D) connecting to additional stars.

E.7) A **tree**, given by a high-speed bus (aka trunk; topology C) connected to stars (aka branches).

E.8) A **spoked ring**, given by a ring (topology B) with a central switch connected (as in a star) to a sparse subset of nodes distributed around the ring. The ring facilitates fast simultaneous nearest-neighbor communications (in 1D), and the spokes facilitate fast communication farther over the network.

E.9) A **spoked grid**, generalizing the spoked ring to a d-dimensional grid (topology E.3, E.4, or E.5), with a central switch connected to a sparse subset of nodes distributed over the grid, to facilitate both fast simultaneous nearest-neighbor communications (in dD), and fast communication over longer distances.

Illustrations of several such logical network topologies are given in Figure 4.1. Note that topologies E.6 through E.9 are especially well suited for algorithms that have a coordinating "central" node that needs especially fast access to all other "compute" or "machine coordination" nodes.

A well-designed structured grid network (topology E.3 or E.4 above) with $2h$ NICs per node and $n \gg 1$ nodes sometimes has embedded within it h **nonoverlapping Hamiltonian circuits**; that is, h entirely nonoverlapping pathways that reach every other node in the network; a couple of examples are illustrated in Figure 4.1, which can be useful for certain data sharing tasks within a network. Consider, for example, a difficult **all-to-all** data transfer problem, in which each node has a certain (large) amount of data that needs to be transferred to all other nodes. Splitting the data to be transferred on each node into h equal-sized pieces and directing each piece along one of the Hamiltonian circuits (from each node simultaneously) gets all of the data where it needs to go in exactly n hops, utilizing each communication link in the network with maximum efficiency.

Once one of the several above network topologies is selected, and set up correctly, the medium access control (MAC) sublayer of the network protocol stack (see §4.1.1) handles all of the low-level rules for determining which links to use to actually route packets across the network in any given situation. This significantly streamlines the coding task for the embedded programmer, requiring simply the calling of the appropriate one-to-one, one-

[2]Consider, for example, a high-resolution x-y-z discretization of a 3D field defined over a cube using a 2D cartesian network topology. Each node in the network can contain the discretized values of the field at all z gridpoints (for a certain range of x and y gridpoints), which substantially accelerates numerical algorithms involving implicit solves or FFTs in the z coordinate (only). Transferring data in one set of directions over the 2D network allows one to quickly perform a sort of matrix transpose, putting onto each node the discretized values of the field at all y gridpoints (for a certain range of x and z gridpoints), thus facilitating fast implicit solves or FFTs in the y coordinate; transferring data in the other set of directions puts onto each node the discretized values of the field at all x gridpoints (for a certain range of y and z gridpoints), thus facilitating fast implicit solves or FFTs in the x coordinate.

[3]The cost of mesh network is proportional to the number of NICs per node. Note that 4 NICs per node form a 2D cartesian or 3D diamond grid, and 6 NICs per node form a 3D cartesian or 5D hyperdiamond grid, clearly favoring noncartesian grid topologies for certain applications (particularly, those with unstructured message passing and a very large number of nodes).

?

Figure 4.1: Illustration of several logical network topologies.

to-all, or all-to-all data transfers in the numerical code, and leaving it to the protocol stack to sort out which links to use to actually complete the requested data transfer.

4.1.3 Ad hoc wireless networks

4.2 Short-range wired communication protocols

4.2.1 Signaling ("bitbanging") with GPIOs, and pullup/pulldown resistors

General-purpose input/outputs (GPIOs)

4.2.2 Encoders (ENC)

4.2.3 Pulse Width Modulation (PWM)

PWM (Pulse Width Modulation) can be used for both driving H-Bridges directly, and signaling to Servomotors (Servos) and Electronic Speed Controllers (ESCs).

A good way to generate a PWM signal of specified frequency and duty cycle is discussed in Example 9.32.

4.2.4 I2C / I3C

4.2.5 SPI / QSPI

4.2.6 UART / USART

4.2.7 USB

4.3 Long-range wired communication protocols

4.3.1 RS485

4.3.2 CAN

4.3.3 Ethernet

(medium range)

4.4 Wireless communication protocols

review

4.4.1 RFID / NFC

4.4.2 Bluetooth / BLE

generation	standard	adoption	frequency band(s)	MIMO	max datarate
Wi-Fi 1	**802.11b**	1999	2.4 GHz		11 Mbps
Wi-Fi 2	**802.11a**	1999	5 GHz		54 Mbps
Wi-Fi 3	**802.11g**	2003	2.4 GHz		54 Mbps
Wi-Fi 4	**802.11n**	2009	2.4 and 5 GHz	✓	600 Mbps
Wi-Fi 5	**802.11ac**	2014	5 GHz	✓	6.933 Gbps
Wi-Fi 6 / 6E	**802.11ax**	2019	2.4 and 5 GHz and 6 GHz	✓	9.607 Gbps

Table 4.1: Commonly used variants of the Wi-Fi standard.

4.4.3 Wi-Fi

Commonly used variants of the Wi-Fi standard are listed in Table 4.1; note that the maximum practical throughput that an application can expect to achieve is about 53% of the max data rate using TCP, and about 64% of the max data rate using UDP.

Wall-powered Wi-Fi routers operating at 2.4 GHz are typically effective up to about 46 m indoors and 92 m outdoors, whereas routers operating at 5 GHz are typically effective over only about a third of these distances, though they can be pushed to significantly higher data rates. Unfortunately, many other household products operate in the 2.4 GHz band, including Bluetooth (see §4.4.2), microwave ovens, and baby monitors. Due to such (often, frustrating) interference issues on the 2.4 GHz band, Wi-Fi 4 and later protocols do not rely exclusively on the 2.4 GHz band. Notably, Wi-Fi 6 and 6E specifically address channel congestion and interference issues, as well as significantly reducing the power required by client devices.

4.4.4 3G/4G/5G cellular

4.4.5 Satellite

4.4.6 Zigbee / Zwave

4.4.7 LoRa / SigFox

LoRaWAN, Symphony Link

4.4.8 LPWAN / NBIOT / LTE-M

4.5 Connector standards

"The nice thing about standards is that you have so many to choose from." - Andrew Tanenbaum

Standard *protocols* for wired communication between "hosts" (e.g., SBCs) and "clients" (sensor modules, actuators, ...) are surveyed briefly in §4.2 and §4.3. What many of these protocols leave unaddressed, however, is the standardization of *connectors* on host and client devices that implement these protocols.

Venerable connector standards like USB (e.g. Types A, C, and micro-B), HDMI (e.g. standard and micro, aka Types A and D) and Ethernet (RJ45) are both relatively compact and ubiquitous in modern laptop computers and SBCs; their use thus requires no further discussion here. Also well standardized are 3.5mm TRRS A/V jacks (aka TRS or "stereo mini" jacks, unfortunately with certain incompatibilities), and MIPI DSI display ports and CSI camera ports (though the MIPI specs are not themselves publicly released). However, as addressed in this chapter, other commonly-needed connectors for wiring together host and client devices are currently much less standardized, sometimes leading to device incompatibilities and often requiring fragile (and, easily misrouted) custom wire harnesses to address.

4.5.1 Existing I2C, SPI, and UART connector standards

Many attempts have been made to standardize powered wire harnesses for various short-range comm protocols (see §4.2), prescribing both the connectors and the corresponding pin order to be used, including:

- standard I2C, with data and clock lines {SDA, SCL},
- extended I2C, adding {INT/SMBA, RES/SMBS} lines to standard I2C,
- SPI, with {MOSI, MISO, SCK, SS}, often with multiple SS (slave select) lines to support multiple devices,
- extended SPI, adding {INT, RES} lines to standard SPI,
- simplex or half duplex UART, using Tx or Rx only, or a single combined Tx/Rx line,
- full duplex UART, with separate transmit and receive lines {Tx, Rx},
- UART with Hardware Flow Control (HFC), adding {CTS, RTS} lines to avoid channel contention,
- (synchronous) USART, adding a clock line SCK to UART to synchronize the receiver and transmitter, and
- other analog or digital signals, such as GPIOs, PWMs, encoder signals, clocks, etc.

Several manufacturers have proposed standardized powered[4] connection protocols to address this need, and marketed a variety of host and client devices mounted on small PCBs using the proposed protocols, including:

- PMOD, a standard by Digilent for 6- and 12-wire harnesses mated with 0.1" pin headers, including:
  - for I2C channels, a 1x6 connector with pin order[5,6] {INT, RESET, SCL, SDA, GND, Vcc},
  - for UART channels, a 1x6 connector with pin order {CTS, Tx, Rx, RTS, GND, Vcc},
  - for SPI channels, a 1x6 connector with pin order {SS, MOSI, MISO, SCK, GND, Vcc},
  - etc. (a handful of other 1x6 and 2x6 connectors are also defined; see the PMOD spec for details);
- Grove, a standard by Seeed for 4-wire harnesses with 2 mm pitch proprietary connectors (see here) with:
  - for I2C connections, a pin order[7] of {SCL, SDA, Vcc, GND},
  - for UART connections, a pin order of {Rx, Tx, Vcc, GND},
  - for other digital and PWM-driven devices, a pin order denoted {D0, D1, Vcc, GND}, and
  - for analog devices, a pin order denoted {A0, A1, Vcc, GND};

[4] Most devices that implement these standards require 3.3V PWR, some use 5V, and some can use either. Check the specs!
[5] Many PMOD I2C connectors on clients are 2x6, with identical columns, facilitating easy daisy-chain wiring of I2C devices.
[6] An older PMOD spec excluded the {INT, RESET} lines from I2C wire harnesses, using 1x4 (or, by Footnote 5, 2x4) connectors.
[7] Note that Seeed sells branch cables to facilitate multiple I2C devices hooked to a single I2C channel.

- STEMMA, a standard[8] by Adafruit for 3- and 4-wire harnesses with 2 mm pitch JST-PH connectors, with:
  - 4-pin connectors, designed for I2C only, with pin order {SCL, SDA, Vcc, GND}, and
  - 3-pin connectors, designed for analog, digital, and PWM-driven devices, with pin order {GND, Vcc, Signal};
- Gravity, a standard by DFRobot for 3- and 4-wire harnesses with 2 mm pitch JST-PH connectors, with:
  - for I2C or UART channels, a 4-pin connector with pin order[9] {Vcc, GND, SCL, SDA} or {Vcc, GND, Rx, Tx},
  - for analog, digital, and PWM-driven devices, a 3-pin connector with pin order {GND, Vcc, Signal};
- Qwiic, a standard by SparkFun for 4-wire harnesses[10] with 1 mm pitch JST-SH connectors, with:
  - 4-pin connectors, designed for I2C only, with pin order {GND, Vcc, SDA, SCL};
- STEMMA-QT, a standard by Adafruit for 4-wire I2C harnesses with JST-SH connectors compatible with Qwiic.

Also noteworthy with regard to connector standardization (or, the glaring lack thereof...) in the industry are:

- the several 1 mm pitch JST-SH connectors used by the Beaglebone Blue, including:
  - for I2C channels, a 4-pin connector with pin order {GND, 3.3V, SCL, SDA},
  - for UART channels, a 4-pin connector with pin order {GND, 3.3V, Rx, Tx},
  - for SPI channels, a 6-pin connector with pin order {GND, 3.3V, MOSI, MISO, SCK, SS},
  - etc. (a handful of other JST-SH connectors are also incorporated; see, e.g., here for pin order), and
- the 3-pin 1.5 mm pitch JST-ZH connector[11], with pin order {3.3V, GND, Rx}, used by the DSM radio receivers.

Digilent's 0.1" pitch PMOD, Seeed's 2 mm pitch Grove, DFRobot's 2 mm pitch Gravity, Adafruit's 2 mm pitch STEMMA and 1 mm pitch STEMMA-QT, and SparkFun's 1 mm pitch Qwiic standards all have their pros and cons. The substantial benefit that they share is the large catalog (from each respective manufacturer) of ready-to-use devices, preassembled on PCBs incorporating the necessary passives, and sold with suitable wiring harnesses and connectors. However 0.1" (2.54 mm) pitch pin headers and 2 mm pitch JST-PH (and similar) connectors are unnecessarily large when considering the current requirements of most devices in these catalogs (connector size becomes an essential limiting factor when designing space-constrained logic boards), whereas 1 mm pitch JST-SH, 1.25 mm pitch JST-GH, and similar connectors are only available as SMD, which are fragile (these connectors often rip off a host PCB if used extensively). Further, the general lack of flexibility in existing standards, in terms of optional additional pins, presents a significant downside for many applications.

4.5.2 Recon: an extensible JST-ZH powered connector standard

The smallest broadly-available, low-cost, 1A-rated connector standard with durable PTH shrouded headers for mounting on a PCB is the 1.5 mm pitch JST-ZH standard, connectors for which are nonreversible (as with all JST standards, but not with bare 0.1" pitch pin headers), as the pins are displaced from the centerline of the connector shroud. Conveniently, JST-ZH wire housings with M pins can also fit into JST-ZH shrouded headers (on a PCB) with N pins so long as $M \leq N$. This leads to the possibility of creating a uniquely *extensible* standard using this type of connector that, for each comm protocol, picks up Vcc and GND on the first 2 pins, then all essential pins for a given comm protocol, followed by a flexible number of optional pins for that comm protocol, all in a predefined order. Following this approach, hosts may be used to drive clients directly using a given comm protocol following this new standard (without incorporating custom wire harnesses that reorder the pins) so long as the shrouded header on the host incorporates at least as many optional pins as the wire harness from the client. Such hosts may include multiple connectors of a given comm protocol (like UART), some with fewer optional pins and some with more, to more efficiently support a rich variety of auxiliary devices.

[8] 4-pin STEMMA and Grove I2C devices are interoperable, and (if powered by 3.3V) STEMMA-QT and Qwiic are interoperable.
[9] Despite published claims to the contrary (1, 2), 4-pin Gravity (with Vcc on pin 1) is not pin compatible with Grove or STEMMA.
[10] SparkFun's Qwiic modules each incorporate a pair of 4-pin JST-SH connectors to facilitate easy daisy-chain wiring of I2C devices.
[11] This 3-pin JST-ZH connector is also included on the BeagleBone Blue, as well as several flight control boards meant for UAVs.

The new extensible JST-ZH based open connector standard proposed here is dubbed **Recon**, and comes in four main types (underlined pins are required, non-underlined are optional):

pin # →	1	2	3	4	5	6	7	8	9	...
Recon Basic	[Vcc,	GND,	S0,	S1,	S2,	S3,	S4,	S5,	S6,	...]
Recon I2C	[Vcc,	GND,	SDA,	SCL,	INT/G2,	RES/G3,	G4,	G5,	G6,	...]
Recon SPI	[Vcc,	GND,	MOSI,	MISO,	SCK,	SSa,	INT/SSb,	RES/SSc,	SSd,	...]
Recon UART-T	[Vcc,	GND,	Tx,	Rx/G1,	SCK/Vbat/G2,	CTS/G3,	RTS/G4,	G5,	G6,	...]
Recon UART-R	[Vcc,	GND,	Rx,	Tx/G1,	SCK/Vbat/G2,	RTS/G3,	CTS/G4,	G5,	G6,	...]

(4.1)

To accelerate the adoption of the Recon standard, ready-made wire harnesses that convert directly from Recon hosts (like the Berets discussed in §5) to PMOD, Grove/STEMMA, Gravity, and Qwiic/STEMMA-QT clients are available, thus enabling such hosts to connect directly (without requiring user-made custom wire harnesses) to *all* of the large catalogs of available client devices incorporating these current competing standards.

The most common voltage used by currently-available client devices is 3.3V; many 5V devices, and an increasing number of 1.8V devices, are also available. The Recon standard thus requires that hosts provide Vcc = 3.3V, and use 3.3V TTL logic, by default on all I2C, UART, and SPI connectors. Other voltages (5V and 1.8V in particular) may be selectable on individual connectors on the host, in order to support an even larger range of client devices. If Vcc = 5V is selectable on a given connector, its (3.3V TTL) digital pins must simply be 5V tolerant. If, Vcc = 1.8V is selectable on a given connector, on the other hand, all of its digital signals must be level shifted (on the host) to the value of Vcc selected (using, e.g., a TI TxB0108 level shifter).

We now discuss some details related to each of the four main types of Recon connectors.

4.5.2.1 Recon Basic and its variants, including PWM and ENC

The simplest Recon connector is a 2-pin power connector, {Vcc, GND}. From this starting point, the Recon Basic standard shares power and a set of one or more generic numbered signals, denoted {S0, S1, S2, S3, ...}, which is useful for signals that do not follow one of the three main short-range digital comm protocols {I2C, UART, SPI} discussed in the following three subsections. For signals that are intended for more specific purposes, different one- or two-character identifiers, plus a sequencing number, may be used; for example, instead of using the name **Basic** and the generic signal names {S0, S1, S2, S3, ...}, one may substitute as follows:

- **GPIO**s may be denoted {G0, G1, G2, G3, ...},
- **PWM** based signals (usually, as outputs from the host) may be denoted {P0, P1, P2, P3, ...},
- **Encoder** (**ENC**) signals (usually, as inputs to the host) may be denoted[12] {E0a, E0b, E1a, E1b, ...},
- **Clock** signals for general-purpose applications may be denoted {CK0, CK1, CK2, ...},
- **Analog** signals may be denoted {A0, A1, A2, A3, ...}, and
- **Digital** comm signals not following the I2C, UART, or SPI standards may be denoted {D0, D1, D2, ...},

thus defining the **Recon GPIO**, **Recon PWM**, **Recon ENC**, **Recon Clock**, **Recon Analog**, & **Recon Digital**[13] variants of the **Recon Basic** standard. Other specific signal names and enumerations may also be proposed and used when necessary, if appropriately documented in the corresponding device datasheet; the Recon Basic

[12] Most modern encoders are quadrature encoders, the outputs of which are attached to the host a pair at a time in order to discern both the speed and direction of rotation of the shaft to which they are attached. For clarity, such pins should thus be enumerated a pair at a time; this modified enumeration of the signals of the Recon Basic standard should not present any confusion.

[13] Other digital comm approaches are typically **bit-banged** from the MCU, which requires the host CPU to manage; the advantage of the I2C, UART, and SPI standards, and their common extensions, is that they may typically be handled by dedicated subunits on modern MCUs, offloading the computational burden of controlling these channels from the available CPU core(s) on the host.

standard itself is meant to be extensible and flexible[14].

GPIOs may be appended to the Recon Basic variants described above, and to the Recon I2C, SPI, and UART standards discussed below; e.g., the connector [Vcc, GND, CK0, CK1, CK2, G3] fully conforms to the Recon Clock spec. A logical numbering for these optional GPIOs, consistent with the Recon Basic spec, is proposed in (4.1); other short/descriptive names for these GPIOs, appropriately documented, should instead be used on clients to identify the functions of the GPIOs that they include, noting that the Recon spec calls for all required and (if included) optional signals listed in (4.1) to remain in the order specified for the corresponding connector.

4.5.2.2 Recon I2C

Perhaps the most widely adopted standard for low-speed short-distance serial communication between a host (aka "master") and multiple clients (aka "slaves") is I2C. The I2C standard facilitates half-duplex[15] communication rates up to 400 kbps, with extensions to 1 Mbps and, via additional logic and clock stretching, to 3.4 Mbps. Standard I2C requires just two digital signals, data and clock {SDA, SCL} (pins 3 and 4 of the Recon I2C standard). An I2C master may communicate individually with up to 112 slaves at addresses x08 to x77 using simple 7-bit device addresses (or up to 1024 slaves, at addresses x000 to x3FF, using 10-bit device addresses). Traditionally, communication via standard I2C requires all transmissions to be initiated by a single master; however, some newer I2C devices implement a multimaster protocol in which different devices (each of which implement the multimaster protocol) can take over the master role on a single I2C bus at different times. Common extensions of the I2C standard add the following (optional pins 5 and 6 of the Recon I2C standard):

- INT, an active low open drain output from the slaves(s) meant to alert the master of new data to report, and

- RES, an active low "reset" or "suspend" output from the master meant both to drive the slaves(s) into a low-power "sleep" state if available, and to re-initialize certain settings on the slaves(s) once released.

The SMBus and related PMBus standards are based closely on I2C; all three types of devices may generally be mixed on a single bus. Amongst other refinements, SMBus standardizes the behavior of the optional INT (aka SMBALERT#) and RES (aka SMBSUS#) pins in a useful way; if these standardized behaviors on the optional INT and/or RES pins are available on a given host or client, they are (for brevity) to be denoted in the Recon I2C standard as SMBA and SMBS, respectively, on the corresponding device.

The newer I3C standard might well reshape how hosts communicate with multiple low-power clients in the coming decade. I3C is also based on the I2C standard, and is compatible with older I2C devices, while allowing much faster comm with other I3C devices over the same {SDA, SCL} pins. The I3C standard facilitates communication rates up to 12.5 Mbps, with extensions to 33 Mbps. One of the new features of I3C is in-band interrupts, which provide an efficient way for slaves to alert the host of new data to report without using a separate INT/SMBA pin, or swapping out which device plays the role of master (the logic of which can get complicated). The Recon I2C connector standard is, of course, compatible with I3C; if/when I3C becomes widely adopted, the name of the Recon I2C standard might well need to be updated to reflect this compatibility.

4.5.2.3 Recon SPI

Another common protocol for short-distance serial communication between one master and multiple slaves is SPI. The SPI approach, which does not have any formal standard, facilitates fast full-duplex[16] synchronous[17]

[14] Note that, e.g., H-bridge outputs for driving brushed DC motors and steppers are generally not considered to be part of the Recon standard, as they are not "powered" connectors with {Vcc, GND}.

[15] Half duplex means that communication in one direction at a time only is allowed.

[16] Full duplex means that simultaneous communication in both directions is possible.

[17] Synchronous means that there is a shared serial clock signal from the master, denoted SCK, upon which both the transmit and receive signals at both ends of the comm channel are coordinated.

communication, often at rates exceeding 10 Mbps. Typical SPI implementations (aka "4-wire SPI") use four signals: master-out-slave-in, master-in-slave-out, serial clock, and slave select, denoted {MOSI, MISO, SCK, SS} (all four signals are required on SPI hosts by the Recon SPI standard), where SS is active low. When multiple slaves are attached to a single SPI master, a different slave select signal {SSa, SSb, SSc, ...} is connected to each attached device; custom wire harnesses are thus generally required.

A few common simplifications of standard SPI (full-duplex, with {MOSI, MISO, SCK, SS} signals) exist:

- In simplex[18] mode, either the MOSI or the MISO wire is simply dropped.

- In half-duplex (aka "3-wire SPI") mode, a single SDIO signal is used for both input and output at different times[19]. On some MCUs, 3-wire SPI mode can simply be selected in software when needed (i.e., to communicate with a 3-wire SPI slave), making SDIO available directly on the host's SPI MOSI pin. Selecting this feature in software on a host allows both 3-wire SPI comm to certain slaves at some times, and 4-wire SPI comm to other slaves at other times. On other MCUs, to facilitate 3-wire SPI, the MOSI pin on the host must be connected via a resistor (on the PCB, likely as a DNP[20]) to the MISO pin, and the modified MISO line subsequently connected to the SDIO pin of the slave. Unfortunately, this hard-wired approach to combining MOSI and MISO on a host would likely interfere with the communication with other 4-wire SPI slaves on the same SPI channel.

Note that, even if there is only one slave device driven by a given SPI channel, the corresponding SS pin on the slave can usually not simply be tied off to GND and the SS signal eliminated, as state transitions on the SS pin are often (but not always) used by the slave to detect the beginning and end of each data transmission.

As with I2C, communication via standard SPI requires all transmissions to be initiated by a single master[21]. Thus, common extensions of the SPI protocol add (software-controlled) INT and RES signals (optional pins 7 and 8 of the Recon SPI standard), the functionality of which is defined as for I2C channels (see §4.5.2.2).

4.5.2.4 Recon UART

A UART is a ubiquitous MCU subunit for asynchronous full-duplex short-distance serial communication, nominally point-to-point (between two devices). UART communication speed is configurable (and, measured on the fly at the opposite end of each wire, rather than being synced via a shared clock), with rates up to 5 Mbps realistically achievable, and 20 Mbps possible under ideal conditions, though many devices top out at 115.2 kbps or less. Unlike I2C and SPI, there is no concept of master or slave in UART; either device can initiate a transmission. Like I2C, standard UART requires just two digital signals, transmit and receive {Tx, Rx} (pins 3 and 4 of the Recon UART standard). Unlike I2C and SPI, the connection of these two signals need to be crossed between one end of the wire harness and the other (i.e., Tx connects to Rx, and Rx connects to Tx); this is accomplished in the Recon UART standard (4.1) by defining a UART-T pin order, usually implemented on hosts, and a UART-R pin order, usually implemented on clients, thus obviating the need for crossing wires within harnesses that connect UART-T connectors to UART-R connectors.

Notable extensions to the UART standard include the following.

- SCK (optional pin 5 of the Recon UART standard) is a serial clock signal used, in a modern yet still somewhat uncommon extension of UART dubbed USART, to synchronize the transmit and receive signals at both ends of the comm channel and thereby facilitate faster communication rates, as done in SPI (see §4.5.2.3). In fact,

[18] Simplex means that communication in one direction only is possible.

[19] Warning: in 3-wire SPI, a resistor is generally needed somewhere along the communication path between the master and the slave, to prevent a possible (though, temporary) direct connection between a driven pin on the master and a driven pin on the slave at the opposite logic state, as the master and slave nodes are not generally synchronized.

[20] DNP means Do Not Populate, or Do Not Place, a given component during the board assembly process, but instead leave an open solder pad at this location, for a component to be added later by the user if desired.

[21] Though rarely used, some hosts do implement a multimaster SPI protocol, though multimaster SPI is restricted to operate between two compatible devices only; unfortunately, this approach does not readily extend to SPI channels with additional slaves on it.

some flexible USART subunits on MCUs can also support an SPI (master or slave) mode of operation; the Recon UART pin order is designed specifically to support this.

- Vbat (optional use of pin 5 in the Recon UART spec) is a secondary (low-current) standby voltage source, which is required by some UART clients (e.g., GPS modules) for efficient operation[22]. If implemented, it is anticipated that Vbat would usually be made available on pin 5 of a Recon UART connector via a PCB solder jumper.

- {CTS, RTS} (optional pins 6 and 7 of the Recon UART spec) are used for Hardware Flow Control (HFC), which is today also somewhat uncommon in clients. The names and functions of these signals are derived from the (once-ubiquitous, but now mostly legacy) full RS232 standard[23]; note that {CTS, RTS} are crossed between the client and the host, like {Tx, Rx}, as again facilitated by the distinct UART-T and UART-R pin orders.

A few simplifications of standard UART (full-duplex, with {Tx, Rx} signals) are also quite common (and thus permitted on both hosts and clients by the Recon UART-T and UART-R connector standards):

- In simplex mode, either the Tx or the Rx wire from the host is simply dropped. Notable common examples include seven segment display drivers, which are transmit only from the host, and DSM receivers, which are receive only at the host (e.g., a mobile robot or drone).

- In half-duplex (aka "single-wire" or "1-Wire") mode, a single signal is again used for both input and output. On some MCUs, this mode (on the Tx line) can simply be selected in software when needed. Usually, a pull-up resistor is needed somewhere along this single wire; to facilitate this mode, it is thus suggested that DNP pads be left for such a pull-up resistor on the host. If a hardware single-wire mode is not available in the UART subunit on the host MCU, and the UART transmit module is (or can be configured as) open drain, the Tx and Rx pins may simply be connected to enable single-wire functionality. Unfortunately, most UART transmit modules are push/pull, thus requiring extra circuitry to convert them to open drain behavior before connecting the Tx and Rx lines to enable single-wire functionality, as discussed further here.

Creative switching strategies and nonstandard ring connections are occasionally proposed to interconnect multiple UART devices. This gets complicated and inefficient (requiring substantial intervention by the CPU) in a hurry; if multiple devices need to be interconnected, the authors thus recommend instead using standard I2C (§4.5.2.2) or SPI (§4.5.2.3) for short-range connections, or CAN or RS485 (§4.5.4.3) for longer-range connections.

4.5.2.5 Reasoning for the Recon pin order

The logic for the pin order adopted across the entire Recon standard is as follows:

- Vcc and GND, which by definition are required on all powered connectors, come first. Following the uniquely extensible Recon standard, JST-ZH wire housings with M pins will often be fit into JST-ZH shrouded headers with N pins, where M<N. It is thus important that Vcc be located on the very first pin, as this prevents Vcc from accidentally being sent directly to any other pin on the client if the connector is inserted incorrectly (not engaging the first pin), thus minimizing the possibility of damaging the client device.

[22] To provide such standby power from the host to SPI or I2C clients, or to USART clients which make use of the SCK pin, while maintaining maximum flexibility and extensibility according to the Recon spec, it is recommended to use a separate 2-pin secondary power connector of the Recon Basic type (see §4.5.2.1).

[23] There are 6 control pins on the common DE9 connector used in this once-ubiquitous standard: {CD, CTS, RTS, DSR, DTR, RI}, standing for Carrier Detect, Clear To Send, Request To Send, Data Set Ready, Data Terminal Ready, Ring Indicator. Of these, only {CTS, RTS} are still in significant use today. If the need arises to support all 6 of the control signals on DE9 connectors (primarily, to support legacy equipment), the Recon UART-T and UART-R standards may be augmented as follows (with typical outputs, inputs specified, noting that DTE originally stood for Data Terminal Equipment, and DCE stood for Data Circuit-terminating Equipment):

Recon RS232-T [Vcc, GND, Tx, Rx, CD, CTS, RTS, DSR, DTR, RI, ...] ← host (aka DTE)
Recon RS232-R [Vcc, GND, Rx, Tx, CD, RTS, CTS, DTR, DSR, RI, ...] ← client (aka DCE) (4.2)

- Data lines come next, with priority given to output from the master in cases that data is carried over 2 wires,

- Clock comes next, followed by slave select(s).

- Optional coordinating signals come last, after all of the required signals, and ordered by frequency of use; these optional signals notably include INT/RES, CTS/RTS, extra SS lines, and extra GPIOs.

It is hoped that, following the logic presented here, new host and client devices following the space-efficient (1.5 mm pitch), secure, durable, extensible, and inexpensive JST-ZH based standard outlined in (4.1) will be developed by various manufacturers. Several conversations advocating for this new standard are already underway; if interested, please contact the author.

4.5.2.6 Recon compatibility, and incompatibility, with pin muxing on current devices

The Recon pin order is compatible with some essential pin muxing design decisions already made for a number of currently-available market-leading host and client devices, including the following:

1. Pin multiplexing {SDA, SCL} on I2C lines with, respectively, {Tx, Rx} on UART lines, as suggested by the UART-T standard, is consistent with the approach taken on several host MCUs, including the Broadcomm BCM2711 in the RPi4 (Table 5.9), and the TI C2000 and MSP432. Recon I2C connectors on such hosts can be converted directly into Recon UART-T connectors (at least, on these primary 2 signals) via a switch in software.

2. Alternate pin functions of STM32 USART modules between UART and SPI modes are consistent with the Recon standard; that is, USART modules on STM32-based hosts can be converted from Recon UART-T connectors (including {SCK, CTS, RTS}) to Recon SPI connectors (including {SCK, SSa, SSb}) via a switch in software. Further, on certain (host) STM32 ICs, {MOSI,MISO} of at least some dedicated SPI modules align with {Tx,Rx} of other dedicated UART modules, consistent with the Recon UART-T standard (at least, on these 2 signals).

3. DSM2/DSMX receivers, which happen to be available already with JST-ZH connectors, are compatible with the Recon UART-R standard. Further, STM32 USART, UART, and LPUART modules, when operating in half-duplex mode and not transmitting, can perform Rx functions on the Tx pin. Thus, the DSM receiver pinout is also compatible with the Recon UART-T standard when using an STM32 host operating in half-duplex mode.

Unfortunately, many pin multiplexing decisions made for currently-available host and client devices do *not* allow for simple software conversion between different comm protocols with consistent pin orders on a given connector (especially on the optional additional pins); indeed, some of these pin multiplexing decisions seem to have been made almost at random. For example:

A. On the STM32, the pin multiplexing between {SDA, SCL} and {Tx, Rx} matches the Recon UART-T order on some channels, but the Recon UART-R order on other channels.

B. On the RPi4, the pin multiplexing between SPI and UART with (optional) HFC, as shown in Table 5.9, does not follow any easily discernible reasoning [cf. §4.5.2.5].

It is suggested that broadly adopting a logical pin muxing standard, consistent with (4.1), might help both IC and PCB manufacturers, of both host and client devices, to market more capable and interoperable products with fewer pins. This may be made possible by deploying *reconfigurable* comm ports that may easily be switched between different comm protocols (e.g., via solder jumpers on clients, or via software on hosts), without having to change the wiring between the host and client devices, thus reducing both IC package size and board and wiring complexity, ultimately reducing manufacturing costs. If the idea of standardizing [to (4.1)] both pin multiplexing and (on connectors) pin order becomes well adopted, such ports could be named as, e.g., a **Recon I2C/UART-R** port (on a client), or a **Recon I2C/SPI/UART-T** port (on a host), thereby indicating both the various comm protocols available on those ports as well as the standardized Recon pin order used.

4.5.2.7 Extended Recon

It is at times useful to provide multiple regulated voltages over a connector. Perhaps the most common need for this is to provide a (low-current) standby voltage to GPS modules to facilitate warm starts; this need is addressed with the optional Vbat pin function in the Recon UART standard, as discussed in §4.5.2.4. However, other situations are anticipated which might also call for multiple regulated voltages to be provided over a single connector. To facilitate this, if appropriately documented (and, if possible, clearly called out on the silkscreen on the PCB itself), **Extended Recon** connector standards may be proposed, implementing one or more extra (optional) regulated voltages provided on pins placed *before* (to the left of) the Vcc = 3.3V pin appearing in the Recon standard given in (4.1). **Warning**: accidentally plugging into these extra voltage pins with a standard Recon connector will likely damage or destroy the host and/or client device; to reduce the likelihood of such a consequential mistake, small dummy plugs should be used to block these pins, thus safely reducing an Extended Recon connector to a standard Recon connector as defined in (4.1).

4.5.2.8 Stackable Recon

The Recon standard is designed to compactly, securely, and extensibly connect PCB hosts to nearby client devices elsewhere on the same mobile robot or electromechanical machine. As motivated in the first two paragraphs of §4.5.2, the Recon standard calls for JST-ZH connectors to be used.

At times when building a mobile robot or electromechanical machine, however, the SBC controlling the machine (and/or its COTS motor control board, such as those described in §5) does not quite have all of the necessary control or filter electronics implemented, and some addition custom circuits are required. In such situations, it is necessary for the user to design and use a custom daughterboard, to connect this custom daughterboard to one or more of the analog or digital comm (I2C, SPI, UART) channels on the SBC or the COTS motor control board, and to securely mount this custom electronics somewhere nearby.

For this task, the use of multiple single-row 0.1" pitch female headers laid out on a 0.1" grid, as popularized by Arduino, is quite convenient. Such an arrangement provides both electrical connectivity to the necessary channels as well as secure physical mounting of the custom daughterboard itself. Also, with such an arrangement and the use of stackable headers, two or more custom daughterboards may be stacked.

Stackable Recon and **Stackable Extended Recon** standards are thus defined that follow the same pin order as the Recon and Extended Recon standards defined above, but using single-row 0.1" pitch female headers (with 0.025" square pins) instead of JST-ZH connectors. The SPI and I2C Headers defined in Table 5.2 are examples of Stackable Extended Recon SPI and Stackable Extended Recon I2C connectors.

Note that small **servos** and **ESCs** ubiquitously come with 1x3 female jacks which mate with 0.1" pitch male header pins on the host. The (non-Recon) order of pins in modern servo connectors is {PWM signal, Vcc, GND}, respectively, with Vcc in the range of +4.8V to +12V. The reasoning for this order for servo connectors is that the 1x3 female jack may easily be plugged into the male header pins backwards; with this ordering (only), this is safe: it will result in the corresponding servo not functioning correctly until the plug is reversed, but it will not damage either the host or the servo. This ordering is well motivated and should not be changed.

4.5.3 Yukon: unpowered connectors

By removing the shared Vcc connection, to interconnect devices that are otherwise already powered, Recon connectors of the five types defined in (4.1) reduce to what we dub Yukon[24] connectors (again, leveraging the

[24] In contrast to the Recon ("Renaissance Connector") standard for short-distance powered connections to sensors, the name of the Yukon standard, which itself evokes extreme physical distancing, is derived as a homophone of Ucon ("Unpowered Connector").

fact that durable [PTH] JST-ZH shrouded headers with N pins can accept JST-ZH wire housings with M pins when $M \leq N$, thus creating a signal-extensible connector standard) as follows:

Yukon Basic	[GND,	S0,	S1,	S2,	S3,	S4,	S5,	S6, ...]
Yukon I2C	[GND,	SDA,	SCL,	INT/G2,	RES/G3,	G4,	G5,	G6, ...]
Yukon SPI	[GND,	MOSI,	MISO,	SCK,	SSa,	INT/SSb,	RES/SSc,	SSd, ...]
Yukon UART-T	[GND,	Tx,	Rx/G1,	SCK/Vbat/G2,	CTS/G3,	RTS/G4,	G5,	G6, ...]
Yukon UART-R	[GND,	Rx,	Tx/G1,	SCK/Vbat/G2,	RTS/G3,	CTS/G4,	G5,	G6, ...]

(4.3)

As with the Recon Basic standard discussed in §4.5.2.1, the Yukon Basic standard may be implemented in **Yukon GPIO**, **Yukon PWM**, **Yukon Enccoder**, **Yukon Clock**, **Yukon Analog**, and **Yukon Digital** variants.

Provided that significant care is exercised when plugging in the connector (in this case, NOT engaging the first pin), Recon connectors can actually be used as Yukon connectors. **Warning**: if this is done incorrectly, power on one side will be connected directly to GND on the other, likely damaging or destroying one or both devices; to reduce the likelihood of such a consequential mistake, while also making the connection of the wire housing even a bit more secure, a small dummy plug should be used to block the first pin of any Recon connector, thus safely reducing it into a corresponding Yukon connector.

As in §4.5.2.8, **Stackable Yukon** connectors are also defined, which follow exactly the same pin order as the Yukon standards defined above, but use single-row 0.1" pitch female headers instead of JST-ZH connectors. The Analog Header defined in Table 5.2 is an example of a Stackable Yukon Analog connector.

4.5.4 CAN and RS485 differential interfaces for remote connections

Though a variety of different effective distances are reported around the web under various conditions (spacing and characteristic impedance of the traces and wires used, electromagnetic interference, possible impedance mismatches at IC/trace and trace/wire junctions, etc), without a repeater and when operating at low comm speeds, I2C links are practically limited to somewhere around 5 m, SPI links are limited to around 10 m, and UART links are limited to around 15 m; at higher comm speeds these effective distance limits are all substantially reduced. To connect over longer distances, differential interfaces communicating over one or more twisted pairs of wires are needed. The two dominant standards today for such differential interfaces are[25] CAN and RS485[26]. A few useful comparisons of these two standards are available here, here, and here. There are a number of subtle issues, including interconnect topology, termination, biasing, grounding, etc., involved in making such systems work well; a succinct review is available here. Industrial RS485 data cables are typically 24 AWG with a characteristic impedence of 100 Ω to 120 Ω; automotive CAN data cables are typically 18 to 20 AWG with characteristic impedence of 110 Ω to 130 Ω. CAT5e or CAT6 cables are often-used inexpensive COTS substitutes for low-cost RS485 networks. Shielding is helpful for maintaining signal integrity, if it is available.

4.5.4.1 To ground, or not to ground?

The question of whether or not a GND connection should be shared between different devices when using a differential interface is particularly delicate. Notwithstanding advice to the contrary in the RS485 (aka TIA485-A) standard itself, which recommends simply using resistors between the ground wire on the interconnecting RS485 cable and the local GND on individual devices, as well as a lot of other misleading advice elsewhere on the web, careful modern guidance is somewhat more nuanced. In short, non-isolated GND should not be shared in situations for which the ground potential difference (GPD) of all devices to be connected will remain well within ±7 V, and thus non-isolated CAN transceivers and RS485 transceivers may be used, whereas isolated GND should be shared in situations for which the GPD might exceed this range, and thus slightly more expensive/complex isolated CAN transceivers and RS485 transceivers should be used instead.

4.5.4.2 Field-serviceable, secure, and durable wiring solutions

As discussed in the definition of the Recon standard §4.5.2, for the wiring of single-board computers to nearby sensors and other devices in mobile robots and within individual space-constrained electromechanical machines, standardized connectors are called for that are:

(a) small (1.5mm pitch appears to be the sweet spot),
(b) secure (not disconnecting due to system vibrations),
(c) durable (able to withstand hundreds of connector insertion/removal cycles – generally this means PTH),
(d) extensible (able to incorporate additional signals if available/necessary), and
(e) inexpensive (leveraging COTS connectors wherever possible, especially for mass-market products).

In contrast, for long-distance twisted-pair wiring solutions (e.g., for automotive, industrial, and outdoor applications), wires and connectors are needed that are:

[25] That is, other than Ethernet, which itself may be a good choice for many long-distance local networks.

[26] RS485 is related to the older but still commonly used RS422 standard. Through a driver enable (DE) feature, RS485 systems can operate with multiple drivers (transmitters) on a single pair of wires, thus facilitating half-duplex (two-way communication) over a single twisted pair (RS422 is simplex only over each pair of wires). RS485 transmitters can handle the load of 32 to 256 receivers on a single twisted pair; some RS422 transmitters can only handle the load of 10 receivers. RS485 can also handle much larger ground potential differences between devices. For these reasons, RS485 is generally preferred over RS422 for new designs.

(A) field serviceable (allowing wires to be replaced with only simple tools, or no tools whatsoever),
(B) even more secure (not disconnecting due to accidental direct tugs on the wire), and
(C) even more durable (surviving heat, direct sunlight, vibration, water, dust, grease, cleaning solvents, etc).

For long-distance connections, the wires themselves are often the weakest links, and must often be replaced when damaged, or cut to a new lengths when the system is reconfigured. In such a setting, connector size and cost are often only minor secondary issues, and field serviceability is paramount. A variety of standardized connectors and T-junctions are available, with 2 to 9 poles, that are well suited in such settings, notably including:

- Pico (M8), Micro (M12), and Mini (7/8 in) connectors, many of which are IP67[27] rated and IDC[28] type,
- RJ45 connectors (8-pin, straight-through T568B, also available as IDC), as used widely for wired ethernet,
- D-sub 9-pin (DE9, aka DB9) connectors, a 0.108" pitch D-shaped shrouded standard that is broadly adopted,
- Micro-D 9-pin connectors, a 0.05" pitch miniaturized version of the DE9 (also available in powered variants),
- simple terminal blocks, which are easy to service by hand but not environmentally hardened, etc.

In each setting, it is essential to follow the corresponding industry or manufacturer's spec as much as possible for where to attach the primary (and if present, secondary) pair of signal wires. For example, if using RJ45 connectors [typically, with inexpensive commercial off-the-shelf (COTS) CAT5e or CAT6 cables], it is recommended that the standard Power over Ethernet pin order be followed:

- the primary twisted pair should be attached to pins 3 and 6,
- the secondary twisted pair (if any; e.g., in full-duplex RS485 mode) should be attached to pins 1 and 2,
- GND (if connected) should be carried on pins 7 and 8, and
- DC power (if connected, which is sometimes convenient for small remote sensors) should be on pins 4 and 5.

4.5.4.3 Recon/Yukon Differential Pairs

The various types of rugged (field serviceable, extra secure, extra durable) connectors discussed above are often mounted on a bulkhead (that is, on the boundary of an environmentally-hardened shell protecting the electronics), and are often too big to mount directly on the PCB itself. In this common setting, the rugged connector on the bulkhead needs to connect to a (small, extensible) connector on the PCB via a short jumper wire (twisted pair ribbon cables are often a good choice). For the connector on the PCB in this setting, a simple extension of the Recon/Yukon standard is recommended:

Recon/Yukon Differential Pairs: [Vcc, GND, A+, A-, B+, B-, C+, C-, D+, D-, ...] (4.4)

Only the first differential pair, denoted here {A+, A-}, are required by this spec; additional differential pairs may be added if available. If both Vcc and GND are included, it is referred to as a Recon Differential Pair connector, otherwise it is referred to as a Yukon Differential Pair connector. Further, on Yukon Differential Pair connectors, GND is also optional; recalling the discussion in §4.5.4.1, a GND connection should generally not be made between two or more connected devices unless isolated CAN or RS485 transceivers are used[29]. Useful definitions of the CAN and RS485 variants of the Recon/Yukon Differential Pairs spec follow[30]:

[27] IP67 means that the component is dust-proof and capable of withstanding temporary immersion up to 1 m depth.

[28] Field-servicable Insulation Displacement Connection (IDC) type connectors may be installed with only simple tools, or in certain cases with no tools whatsoever.

[29] To prevent making a mistake in this regard, Yukon connectors without ground pins should be used on PCBs with non-isolated transceivers, and Recon or Yukon connectors with ground pins may be used on PCBs with isolated transceivers; the latter should always be selected, and the corresponding isolated GND pins connected, if significant GPDs are expected.

[30] If using simple terminal blocks on the bulkhead, we recommend following the same order.

Recon/Yukon CAN:	[Vcc, GND, CANH, CANL]	
Recon/Yukon RS485-H:	[Vcc, GND, A, B]	for half duplex connections over the A/B channel,
Recon/Yukon RS485-Y:	[Vcc, GND, Y, Z, A, B]	full duplex, with the Y/Z (transmit) channel first,
Recon/Yukon RS485-A:	[Vcc, GND, A, B, Y, Z]	full duplex, with the A/B (receive) channel first.

(4.5)

In the full duplex case, akin to the Recon/Yukon UART-T and UART-R standards, note that:

- master devices transmit on the first pair and receive on the second using Recon/Yukon RS485-Y connectors,
- slave devices receive on the first pair and transmit on the second using Recon/Yukon RS485-A connectors,

thus facilitating full duplex communication between any master and any slave on the network. Warning: in either the full-duplex or half duplex case, care must be taken in software such that, on any given twisted pair, only one RS485 node has an active driver (transmitting) at an given time.

By electrically connecting the Y and A pins and the Z and B pins of a full-duplex RS485 transceiver at the JST-ZH connector [e.g., with (initially-open) solder jumpers on the PCB nearby], and attaching a data cable to the A and B (or, to the Y and Z) pins only, a (full-duplex) Yukon RS485-A or RS485-Y connector is reduced to (half-duplex) RS485-H functionality. This is a useful way to configure full-duplex RS485 transceivers+connectors for general use, if you don't know whether a full-duplex or half-duplex RS485 network will ultimately be deployed.

In the full duplex case, the receive and transmit functions are decoupled from each other at any given node. In the half-duplex case, it is common to turn the receiver off whenever transmitting, and vice-versa, simply by tying the (active high) DE (driver enable) pin to the (active low) $\overline{\text{RE}}$ (receiver enable) pin. This is not the only valid approach, however:

- by receiving all the time (tying $\overline{\text{RE}}$ low), a transmitting node also receives its own data as it is being sent (this is called a "loopback" or "echo" function, and can be used to verify the quality of each transmission), or
- by shutting off both the transmitter and the receiver (setting DE low and $\overline{\text{RE}}$ high), a node can save energy.

4.5.5 Summary

Severos and ESCs are often controlled with simple PWM signals. Encoders generate signal transitions that get counted on hosts. The three short-range comm protocols, I2C, SPI, and UART, and the two long-range comm protocols, CAN and RS485, are also well established. Each of these standards has their place in the development of modern robotic systems. As discussed briefly in this short chapter, several useful extensions of these standard protocols are also readily available, and some exciting new extensions, like USART and I3C, are emerging.

As a developer of modern robotic systems, you will primarily use existing dedicated hardware subsystems to implement these comm protocols, on both MCU clients (controller boards) and hosts (sensors and actuators); you will not have to implement them from scratch yourself. However, it is still important to understand them, and their relative strengths and implementation details, so you can:

(a) select appropriately which standards you will implement to interconnect different subsystems, and

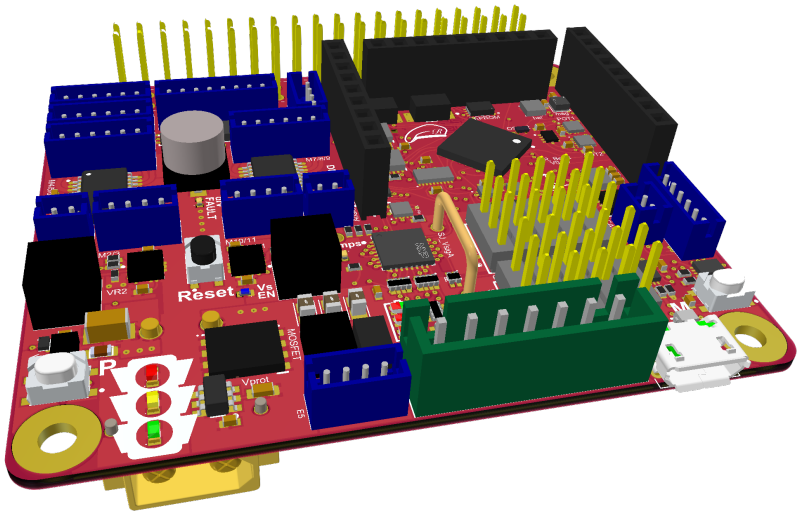
(b) wire together different subsystems implementing these standards with maximum effectiveness.

Curiously, a primary challenge in implementing these comm standards is to converge upon a self-consistent set of connector standards to attach hosts to clients. Available solutions today, as reviewed in §4.5.1, are rather all over the map. This chapter suggests a perhaps more deliberate approach to laying out connectors on client and host PCBs, using durable (PTH), standard (JST-ZH), 1.5 mm pitch, 1 A rated connectors wherever possible, with a deliberate pin order that, uniquely, allows the adoption of a *flexible* standard, with optional pins available on hosts that may or may not be needed on clients, and which may be connected with standard (socket-to-socket, straight, non-reversed) JST-ZH cable assemblies, which are readily available.

Chapter 5

Berets

Bespoke
Emerging
Robotics
Electronic
Technologies



The **Raspberry**, **Red**, **Black**, **White**, **Green** and **Blue** Berets, from RenaissanceRobotics.com, are a family of 6 bespoke motor control boards that are remarkably compact, powerful, efficient, and extensible. Berets are designed to be operated as daughterboards for essentially *all* of the leading SBCs, as reviewed in Table 1.15, or for standalone operation. Their modern hardware and modular software, described in detail in this chapter[1], are open source, providing a reference platform that can be reduced or extended for a host of practical applications in mobile robotics, industrial automation, precision agriculture, pharmaceutical development, food preparation, remote inspection, smart-grid HVAC, elder care, toys, etc. The motivation for developing and adopting the Beret family of boards is to simplify and accelerate the deployment of bespoke feedback controllers for complex mechatronic systems, empowering robotics students (in high school, college, and beyond), and expediting and streamlining the workflow from lab prototype to commercial product.

Berets are cross-platform[2], open-design demonstrators of emerging technologies for motor control, power regulation, and coordinated feedback in robotics applications. Most of the ICs selected for the Berets, by **TI** (motor drivers, voltage regulation, power protection, opamps, switches/multiplexers, digital pots, CAN and RS485 transceivers, level shifters, LED driver), **ST** (MCU, magnetometer, barometer), **NXP** (GPIO expander), and **TDK** (IMU), were announced shortly before the Berets were designed (as a Covid lockdown project, in 2020-2021). Being open design, the hardware and software used by the Berets facilitate the development of bespoke derivative boards custom fit to the user's application. To realize this vision, we present below various implementation details and salient features of the several subsystems of the Beret family.

[1] In fact, this chapter serves as the datasheet for the Beret family of boards. As a developer of feedback control solutions for mobile robots and cyber-physical systems, the reader must become adept at reading datasheets carefully. This chapter serves as a case study for such datasheets, and should be informative even if different components are ultimately needed in the user's application.

[2] That is, Berets facilitate (a) the easy porting of real-time multithreaded control code from one linux SBC family to another (see Table 1.15) as algorithmic demands (e.g., vision processing) on the SBC increase, or (b) the elimination of the SBC altogether (in favor of a multithreaded ARM-Cortex M implementation) to substantially reduce system cost for high-volume production (e.g., toys).

We begin with a chapter-level table of contents to help navigate this (rather long) chapter.

Contents

Beret	MB header	PCB dimensions		balance	analog	peripherals	bolt pattern
Raspberry	**RPi**	**full size**	65×56	✓	✓	3.3V, 5V	RPi
Red	**RPi**	**full size**	65×56	✓	×	3.3V, 5V	RPi
Black	**96B**	**full size**	85×54	✓	✓	3.3V, 5V	96B
White	**BB**	**full size**	$3.4'' \times 2.15''$	✓	✓	3.3V, 5V	BB
Green, **Blue**	(none)	**half size**	39×61.5	×	✓	3.3V only	(custom)

Beret	MB voltage	Drivers:	motors	encoders,	servos/ESCs	Remote Busses	Optional
Raspberry	**5V MB**	**24 HB**	8	7	10	**CAN/RS485**	Coin, Flash
Red	**5V MB**	**12 HB**	4	5	5	(local only)	Coin, Flash
Black	**12V MB**	**24 HB**	8	7	10	**CAN/RS485**	Coin, Flash
White	**5V MB**	**24 HB**	8	7	10	**CAN/RS485**	Coin, Flash
Green	(none)	**0 HB**	0	5	10	**CAN/RS485**	Coin, Flash
Blue	(none)	**12 HB**	4	5	5	(local only)	Coin, Flash

Beret	PCB edges	mounting hole centers	IMU center	Analog pin 9
Ras, **Red**	$(\pm 32.5, \pm 28)$	$(\pm 29, \pm 24.5)$	(25,0)	(0,?)
Black	$(\pm 42.5, \pm 27)$	$(\pm 38.5, -8.5), (\pm 38.5, 23)$	(?,?)	(0,?)
White	$(\pm 1.7'', \pm 1.075'')$	$(-1.475'', \pm 0.825''), (1.125'', \pm 0.95'')$	(?,?)	(0,?)
Green, **Blue**	$(\pm 19.5, \pm 30.75)$	$(\pm 16, 27.25), (0, -27.25)$	(12.5,0)	(-12.5,0)

Table 5.1: Essential features of the six initial Beret variants. All distances in mm except on the **White**, for which distances are given in inches. The center of each board is taken as the reference point when defining the PCB edges, the mounting hole centers, and the center of the IMU coordinate system. On all Berets, M2.5 hardware may be used; on the **White** + BB, #4-40 hardware fits a bit better in its $0.125''$ holes.

5.1 Overview

The essential features of the six initial Beret variants are outlined in Table 5.1. In particular:

- the **Raspberry** (aka **Ras**) Beret may be operated as a daughterboard for recent versions of the RaspberryPi,
- the entry-level (lower-cost) **Red** Beret is a partially-populated **Raspberry** Beret[3] with reduced specs,
- the **Black** Beret may be operated as a daughterboard for SBCs following the 96Boards CE specification,
- the **White** Beret may be operated as a daughterboard for the BeagleBone Black and AI, and
- the **Green** Beret is designed for standalone (or, wired remote) operation with 3.3V peripherals only.
- the **Blue** Beret is designed for standalone mobile applications with 3.3V peripherals only.

Note that, in fact, all six Beret variants may be operated standalone when properly programmed.

In this section, we briefly summarize the subsystems used in this family of boards; the balance of §5 explains these subsystems in greater detail. Note that the modern switching regulators, motor drivers, MCU, MOSFET, op amps, and other power components discussed below are best-in-class in terms of their efficiency. This, of course, means that the system can run a bit longer on a single battery charge than it could otherwise. Perhaps even more important, however, is that a reduced amount of waste heat is generated by these components, thus significantly improving the capability of Berets to handle high-current operating conditions.

[3] Because red is a less-fancy way of saying raspberry; Prince wrote a song about the latter, not the former...

POWER

The **Raspberry**, **Red**, and **White** Berets, dubbed **5V MB** Berets, target 5V motherboards. They are powered over an XT30 connector by a **Vin = 6.2V – 28V** input at up to 15A continuous / 20A peak, thus accommodating a **2S – 6S LiPo** (3.1 – 4.2V per cell) or LiHV (3.2 – 4.35V per cell), 3S – 7S LiFe (2.5 – 3.65V per cell), 7S – 18S NiMH (1 – 1.5V per cell), or a **7V – 28V wall adapter**, which is down-regulated as follows:

- the Vin->Vs1 switching regulator provides **Vs1 = 4.8V** to **min(12V, 0.8*Vin)** at up to **6A** for servos & ESCs,
- the Vin->Vmb switching regulator provides **Vmb = 5.1V** at up to **6A** for an RPi or BB compatible MB,
- the Vmb->3.3V switching regulator provides **3.3V** at up to **3A** for logic circuits and connected sensors, and
- the Vs2 power opamp provides **Vs2 = 1.2V** to **2.1V** at up to ± **400 mA**.

The **12V MB** **Black** Beret targets 12V motherboards. It is powered over an XT30 connector by a **Vin = 12V – 28V** input at up to 15A continuous / 20A peak, accommodating a **4S – 6S LiPo** or LiHV, 5S – 7S LiFe, 12S – 18S NiMH, or a **12V – 28V wall adapter**, which is down-regulated slightly differently:

- the Vin->Vs1 switching regulator provides **Vs1 = 4.8V** to **min(12V, 0.8*Vin)** at up to **6A** for servos & ESCs,
- the Vin->Vmb switching regulator provides **Vmb = min(12V, 0.8*Vin)** at up to **6A** for a 96B compatible MB,
- the Vmb->3.3V switching regulator provides **3.3V** at up to **3A** for logic circuits and connected sensors, and
- the Vs2 power opamp provides **Vs2 = 1.2V** to **2.1V** at up to ± **400 mA**;

in addition, per the 96Boards (96B) CE specification, the 96B motherboard, if one is attached, down-regulates the Vmb line and passes back (via the low-speed header) **5V** at up to **1A**. The 12V MB Beret thus does not itself have a 5V regulator (and, thus, the 5V subsystem on this Beret is not functional unless a 96B compatible motherboard is attached).

The **Green** and **Blue** Berets are built for compact standalone aplications with 3.3V peripherals only, bypassing the generation of both Vmb and 5V (saving both board area and cost). They are powered over an XT30 connector by a **Vin = 5V – 28V** input at up to 15A continuous / 20A peak, accommodating a **2S – 6S LiPo** or LiHV, 2S – 7S LiFe, 5S – 18S NiMH, or a **5V – 28V wall adapter**, which is down-regulated as follows:

- the Vin->Vs1 switching regulator provides **Vs1 = 4.8V** to **min(12V, 0.8*Vin)** at up to **6A** for servos & ESCs,
- the Vin->3.3V switching regulator provides **3.3V** at up to **3A** for logic circuits and connected sensors, and
- the Vs2 power opamp provides **Vs2 = 1.2V** to **2.1V** at up to ± **400 mA**.

In general, on all Berets:

- **Vin** powers the motor drivers directly,
- **Vs1** powers the signal headers (for servos & ESCs),
- **Vmb** powers the attached motherboard,
- the **3.3V** and **5V** lines power the logic circuits on the Beret, the JSTs, and the digital and analog headers, and
- all digital outputs operate at **3.3V TTL**, and all digital inputs are **5V tolerant**.

Again, note that the Vmb and 5V lines are absent on the **Green** and **Blue** Berets.

MICROCONTROLLER (MCU)

For real-time control of brushed (BDC) motors, stepper motors, servo motors, electronic speed controllers (ESCs), and brushless (BLDC) motors, coordinating motor commands with a wide variety of sensor inputs, Berets incorporate[4] a 100-pin 170 MHz (213 DMIPS) STM32G474VE with an ARM Cortex-M4 core, 512 KB flash[4], 128 KB SRAM, and integrated DSP, FPU, and CORDIC (transcendental) & FMAC (filter math) accelerators.

Berets run FreeRTOS and leverage the efficient, portable, open-source Robot Control library (written in C) for driving all hardware. A ROS interface to these subroutines is under development.

[4]Note that the entry-level **Red** Beret uses a STM32G474VB instead of a STM32G474VE, with 128 KB flash instead of 512 KB, but is otherwise identical in terms of the specs described here.

Significantly, Berets break out many of the STM32G474's dedicated subsystems, each of which operate without loading the main ARM core on the STM itself. An additional 24 GPIOs are provided by a dedicated GPIO expander. Note that computationally-heavy tasks (e.g., for vision-based situational awareness) should be deferred to the attached linux-based motherboard.

MOTORS, ENCODERS, SERVOS & ESCs

The **Raspberry**, **Black**, and **White** Berets, dubbed **24 HB** Berets, have **24 half bridges**, with drivers for simultaneous independent bidirectional control of **8 brushed DC motors** operating at Vin at up to 12A total; the connectors to these half bridges may be ganged together in various ways to control: 2 motors at 6A, 4 motors at 3A, 4 motors at 2A + 4 motors at 1A, etc., or attached in a unique sequential mode for independent bidirectional control of up to 24 1A motors at reduced duty cycles. The 24 HB Berets also provide dedicated hardware support for **7 quadrature encoders** and **10 servos or ESCs** of a wide variety of sizes and types.

The **12 HB** **Red** and **Blue** Berets have **12 half bridges**, with drivers for control of **4 brushed DC motors** operating at Vin at up to 6A total; the connectors to these half bridges may be ganged together to control: 1 motor at 6A, 2 motors at 3A, 2 motors at 2A + 2 motors at 1A, etc., or attached in sequential mode for independent bidirectional control of up to 12 1A motors at reduced duty cycles. The **Red** and **Blue** Berets also provide dedicated hardware support and pinouts for **5 quadrature encoders** and **5 servos or ESCs**.

The **0 HB** **Green** Beret actually has no half bridges itself (which, admittedly, at first seems peculiar for a motor control board!). This Beret leverages daughterboards, dubbed **Beret Shields**, for implementing the specific motor drivers that may be required in any given application. The **Green** Beret provides dedicated hardware support and pinouts for **5 quadrature encoders** and **10 servos or ESCs**.

BATTERY MONITORING & CHARGING

The **Raspberry**, **Red**, **Black**, and **White** Berets, dubbed **full size** Berets, monitor individual battery cell voltages when running. The **half size** **Green** and **Blue** Berets, on the other hand, forgo individual battery cell voltage monitoring; when running on a battery, they only monitors the overall battery charge.

Note that, on all Berets, battery charging must be done via a (high-quality) off-board battery charger.

SENSORS & CONNECTORS

All six Berets include a sensitive 6-axis **IMU** (3 accels + 3 gyros), 3-axis **magnetometer**, and **barometer**, in addition to discrete connectors for driving a host of **I2C**, **SPI**, and **UART** sensors and other devices (such as GPS/GNSS units and DSM radio receivers), a **USB Micro-B** input for programming, and numerous channels configurable as **GPIOs**. In addition, the **Raspberry**, **Black**, **White**, and **Green** Berets, dubbed the **CAN/RS485** Berets, include high-speed **CAN-FD** and full duplex **RS485** transceivers and connectors.

Servos and ESCs are supported on the Berets by industry-standard (triple row, 0.1" pitch, 3A per pin, PTH) **Signal Headers**, arranged in one or two easy-to-use 3x5 cluster(s) of pins. All six Berets also have an Arduino-style (1x9, 0.1" pitch, 3A per pin, PTH) **SPI Header** and **I2C Header**.

ANALOG SUBSYSTEM

The **Raspberry**, **Black**, **White**, **Green** and **Blue** Berets also have a 1x9, 0.1" pitch **Analog Header**, limited to 0V to 3.3V operation in a unipolar or bipolar setting, with reference GND at 0V or Vs2 ≈ 1.65V, and:

- two 16-bit **ADC**s with tunable gain (x1 to x4096) & tunable second-order low-pass filtering (f_c = 34 to 3400 Hz),
- two 12-bit **DAC**s with ±400 mA outputs, and
- a ±400 mA **opamp** pinout (V+, V-, Vo).

The entry-level **Red** Beret forgoes the entire analog subsystem, including the Analog Header.

CONNECTOR STANDARDS: RECON AND YUKON

Sturdy JST-ZH connectors (1.5 mm pitch, 1A per pin, PTH) are used for motors, encoders, U(S)ART, CAN, and RS485. The pin order on all JST-ZH connectors and single-row headers follow the standards defined in §??:

- E1-2 (I2Cd), E3-4 (UARTt), E5 (UARTr), E6-7, USART (UARTb/SPIb) follow the Recon Basic, Recon I2C, Recon UART-T, Recon UART-R, and Recon SPI pin order standards defined in §4.5.2,
- CAN follows the Yukon CAN standard defined in §4.5.4.3, while RS485 follows the Yukon RS485-Y (host) standard on the **Raspberry**, **Black**, and **White** Berets, and the RS485-A (client) standard on the **Green** Beret (and, when acting as UARTa, the Recon UART-T standard on all four of these Berets). On the **Blue** Beret, the corresponding connector is UART-T only.
- the SPI and IC2 Headers follow the Stackable Extended Recon standards defined in §4.5.2.8, and
- the Analog Header follows the Yukon Basic standard defined in §4.5.4.1.

EXPANSION BOARDS: BERET SHIELDS

The (1x9) SPI Header, I2C Header, and (if included) Analog Header, in addition to the (3x5) Signal Header A, are all aligned on a 0.1" pitch grid, facilitating the easy and secure mounting of stackable COTS and user-designed **Beret Shields** with additional analog and digital circuitry.

MOTHERBOARD (MB) HEADERS

Berets communicate with motherboards using SPI. To make this connection easier,
- the **Raspberry** and **Red** Berets, dubbed the **RPi** Berets, have a 2x20, 0.1" pitch stackable RPi header,
- the **Black** Beret, dubbed the **96B** Beret, has a 2x20, 2 mm pitch stackable 96B header, and
- the **White** Beret, dubbed the **BB** Beret, has a 2x23, 0.1" pitch stackable BB header.

The **Green** and **Blue** Beret do not have any board-specific MB headers, though they may be connected locally to virtually any MB (e.g., over SPI, CAN or RS485, etc), or operated standalone.

On the **full size** Berets, the MB Header may be broken out with additional compact PCBs (a.k.a. SHIMS), and standardized EEPROMs are included to identify the Berets appropriately to connected MBs.

OTHER FEATURES

The STM's dedicated hardware timers (for, e.g., encoder counting and PWM generation) are highly reconfigurable, with additional (unidirectional) encoder counters easily configurable on the Signal Headers, or additional PWM outputs (for more servos and ESCs) easily configurable on the encoder connectors, if necessary.

The battery state of charge is indicated with a three bicolor LED power gauge. Three buttons (reset/shutdown, pause, mode), three user-programmable stoplight LEDs, and various status LEDs are also included.

An (optional) rechargeable **Vcoin = 2.6V** to **3.05V** coin cell may be installed on Berets to keep the real-time clock (RTC) current, thus facilitating scheduled system wakeups.

Berets are also easily upgraded with an (optional) low-cost 4 MB to 512 MB flash IC.

Note: **Ras** and **Red** are 36.4 cm^2, **Black** is 45.9 cm^2, **White** is 47.2 cm^2, **Green** and **Blue** are 24 cm^2.

FEATURE SET SUMMARY, PINOUTS, LAYOUT, AND FUNCTIONAL DIAGRAMS

The distribution of the above-described features over the six different Beret versions is summarized in Table 5.1. Layout and functional representations of each is given in Figures 5.1-5.6.

The pinouts of the several connectors on the Berets are listed in Table 5.2. The rest of §5 is devoted to explaining these subsystems further. In particular, discussion of how the specific channels on the STM32 are hooked up to these several connectors is summarized in §5.5; see in particular Figure 5.9 and Table 5.4.

Connector[A]	Pins	Signal
MB Header[D]	all	(see §5.9)
USB Micro-B	1	USB_5V
	2,3,4	DM, DP, GND
XT30 (Power In)	1	Vin (up to 28V/20A)
	2	GND
Balance[E,F]	1	B1 (cell 1 low)
	2	B2 (cell 1 high / 2 low)
	3	B3 (cell 2 high / 3 low)
	4	B4 (cell 3 high / 4 low)
	5	B5 (cell 4 high / 5 low)
	6	B6 (cell 5 high / 6 low)
	7	B7 (cell 6 high)
M1	1,2	M1a, M1b
M2-3	1-4	M2a, M2b, M3a, M3b
(*M4-5-6*, *M7-8-9*, *M10-11*, and *M12* are similar)		
E1-2[G] (I2Cd)	1,2	[**3.3V**/5V/Vs1/off][B], GND
	3	E1b/SDA/G0
	4	E2a/SCL/G1
	5	E2b/SMBA/G2
	6	E1a/RES/G3
E3-4 (UARTt)	1,2	[**3.3V**/5V][B], GND
	3,4	E3a/Tx/G0, E3b/G1
	5,6	E4a/G2, E4b/G3
E5 (UARTr)	1,2	[**3.3V**/5V][B], GND
	3,4	E7a/Rx/G0, E7b/G1
E6-7	1,2	[**3.3V**/5V/Vs1/off][B], GND
	3,4	E5a/G0, E5b/G1
	5,6	E6a/G2, E6b/G3

Connector[A]	Pins	Signal
USART (UARTb/SPIb)	1,2	[**3.3V**/5V][B], GND
	3,4	Tx/MOSI/G0, Rx/MISO/G1
	5,6	SCK/G2/Vcoin, CTS/SSa/G3
	7	RTS/INT/SSb/G4
	8	RES/SSc/G5
CAN[C]	1,2	CANH, CANL
RS485[C] (*UARTa*)	1	Y or A or [**3.3V**/5V][B]
	2	Z or B or GND
	3	A or Y or Tx/G0
	4	B or Z or Rx/G1
Signal Header A[H] (S1-S5/I2Cb/I2Cc) (top row)	1	S1/I2Cb_SDA/I2S_MCK
	2	S2/I2Cb_SCL/I2Cc_SMBA
	3	S3/I2Cb_INT
	4	S4/I2Cc_SDA
	5	S5/I2Cc_SCL
(middle row)	1-5	[**Vs1**/Vin][B]
(bottom row)	1-5	GND
(*Signal Header B* is similar, with S6 - S10 in top row, and restricted to Vs1 in the middle row)		
SPI Header (SPIa)	1,2,3	5V, 3.3V, GND
	4,5,6	MOSI/SD/G0, MISO/G1
	4,5,6	SCK/G2, SSa/WS/G3
	7,8,9	SSb/G4, SSa/IR_OUT/G5
I2C Header (I2Ca)	1-5	Vcoin, Vs2, Vs1, 3.3V, GND
	6,7	SDA/G0, SCL/BOOT0/G1
	8,9	G2, G3
Analog Header[A,I]	1,2	DAC1buf, DAC2buf
	3,4,5	V+, V-, Vo
	6,7,8	Vref, ADC1, ADC2
	9	ADC2filt

Table 5.2: Pinouts (primary role: output, input, i/o, power/ground) of the connectors on all six Berets, with some of the connectors in *italics* dropped on the **12 HB** and **0 HB** Berets (for details, see Table 5.1).

USAGE NOTES

A. Connector pins, except on the MB Header, are numbered W (left) to E (right), or N (top) to S (bottom) [see Figs 5.1-5.6]. All digital signals on the encoder and USART JSTs, and on the Signal, SPI, and I2C Headers, may be configured as GPIOs in software. All connectors follow the Recon and Yukon standards of §??. Digital outputs all operate 0 to 3.3V TTL. Digital inputs are all 5V tolerant, however: **warning:** all pins on the Analog Header are limited to 0 to 3.3V operation.

B. **Warning:** the power supplied to this pin (default is **bold**) may be changed changed by the user, using a multiplexer, a backside solder jumper, or a shunt connector (see §5.2.9).

C. **Warning:** the differential CAN and RS485 transceivers operate 0 to 3.3V; RS485 may be changed to 0 to 5V via a backside solder jumper. Using a 4PDT switch, if the Beret GPIO RS485_SEL=1 (see §5.5), the RS485 connector functions as a full-duplex RS485-Y (host) connector on **Raspberry**, **Black**, and **White** Berets, and as a RS485-A (client) connector on **Green** Berets; if RS485_SEL=0, it functions as a UART-T connector. On the **Blue** Beret, the corresponding connector is UART-T only.

D. The wiring of the Beret's SPI3 channel, its user-defined {MB_G0, MB_G1, MB_G2} GPIOs, and (optionally) its three sensor interrupt channels to the motherboard (MB) header varies somewhat amongst the different Beret versions, as described in §5.9. All JSTs and single-row pin headers on the Berets are connected to the STM and its associated ICs on the Beret, not to the attached MB. Compact breakout-boards (aka SHIMs) are available separately to conveniently break out the functionality on the corresponding MB Headers.

E. The Balance connector is JST-XH, and all other JSTs on the Beret are JST-ZH. As opposed to, e.g., the fragile SMD JST-SHs on the Beaglebone Blue, all JSTs and headers on the Beret are PTH, for durability (still, be gentle!).

F. The custom JST-XH connector on the Beret is modified in such a way as to connect securely to the (3-pin to 7-pin) JST-XH balance connectors on 2S-6S batteries. Included with this custom connector is a set of small/snug "dummy plugs" that may be used to cover the 1 to 4 unused pins on this connector when using 5S to 2S batteries, which helps to provide a more secure fit, and also prevents the balance connector from being plugged into the wrong pins when swapping such batteries to recharge; always make sure that one end of the JST-XH balance connector lines up with white triangle printed on the PCB. **Warning**: these dummy plugs can be pried out with a small screwdriver when necessary, but do so only when the board is not connected to a power source.

G. As specifically permitted in the flexibility of the Recon Basic standard in §4.5.2.1, note that the order of the {E1a, E1b, E2a, E2b} signals on the E1-2 connector are permuted in such a way that, if used for I2Cd, this connector is in standard Recon I2C pin order using the hardware I2C4 channel on the STM.

H. The signal headers, designed to drive up to[5] 10 servos (motors packaged with simple control logic designed to turn a shaft to a desired position and hold it there) or ESCs (motors packaged with control logic designed to accelerate/decelerate a shaft to a desired angular velocity and keep it there), accept all modern 0.1" pitch 3x1 servo connectors[6,7], including both Fubata J and JR/Universal/Hitec S/Airtronic Z. Low-power servos and ESCs typically power both their control logic and their motor using the GND and Vs1 pins on the signal header. High-power servos and ESCs, on the other hand, typically power their control logic using the GND and Vs1 pins on the signal header, but power their motor directly from the power supply (i.e., not via the Beret!). The Beret provides Vs1 = 4.8V to 12V (adjustable in software), at up to 3A on each individual servo connector (6A total), to support a broad range of small servos, ESCs, or other peripherals. Y Adapters (available separately) may be inserted between high-capacity LiPos and the Beret to break out power separately for higher-current servos and ESCs.

I. The (buffered) DAC outputs operate between GND and 3.3V at up to 400mA. The ADC inputs may be configured for software-tunable amplification (x1 to x4096) and 2nd-order low-pass filtering (with f_c =34 to 3400 Hz) of unipolar or bipolar analog signals, or for differential comparison of two analog signals (see §5.7).

[5] The 12 HB Berets have one bank of 0.1" pitch 3x5 pin headers, for driving up to 5 servos or ESCs. The 24 HB Berets have two such banks of 0.1" pitch 3x5 pin headers, for driving up to 10 servos or ESCs; the signal headers are broken out into separate 3x5 banks like this because many servo connectors are, unfortunately, about 0.1 mm wider than the standard 2.54 mm pin spacing near their tips, and 0.2 mm wider away from their tips, which gradually adds up; putting more than N=5 such connectors next to each other on standard 0.1" 3xN pin headers ultimately puts undue stress on the header pins.

[6] Note: modern servo connectors, all of which have power (the red wire) attached to the central pin, may easily be accidentally plugged into the 3x5 pin headers backwards. This is completely safe; it will result in the corresponding servo not functioning correctly until the plug is reversed, but it will not damage either the Beret or the servo.

[7] **Warning**: do **not** use old servos with Airtronic T connectors with the Beret; damage to the Beret and/or servo will result. This obsolete configuration can be recognized easily by the fact that, on it, power is connected to pin 1 or 3, not to the central pin.

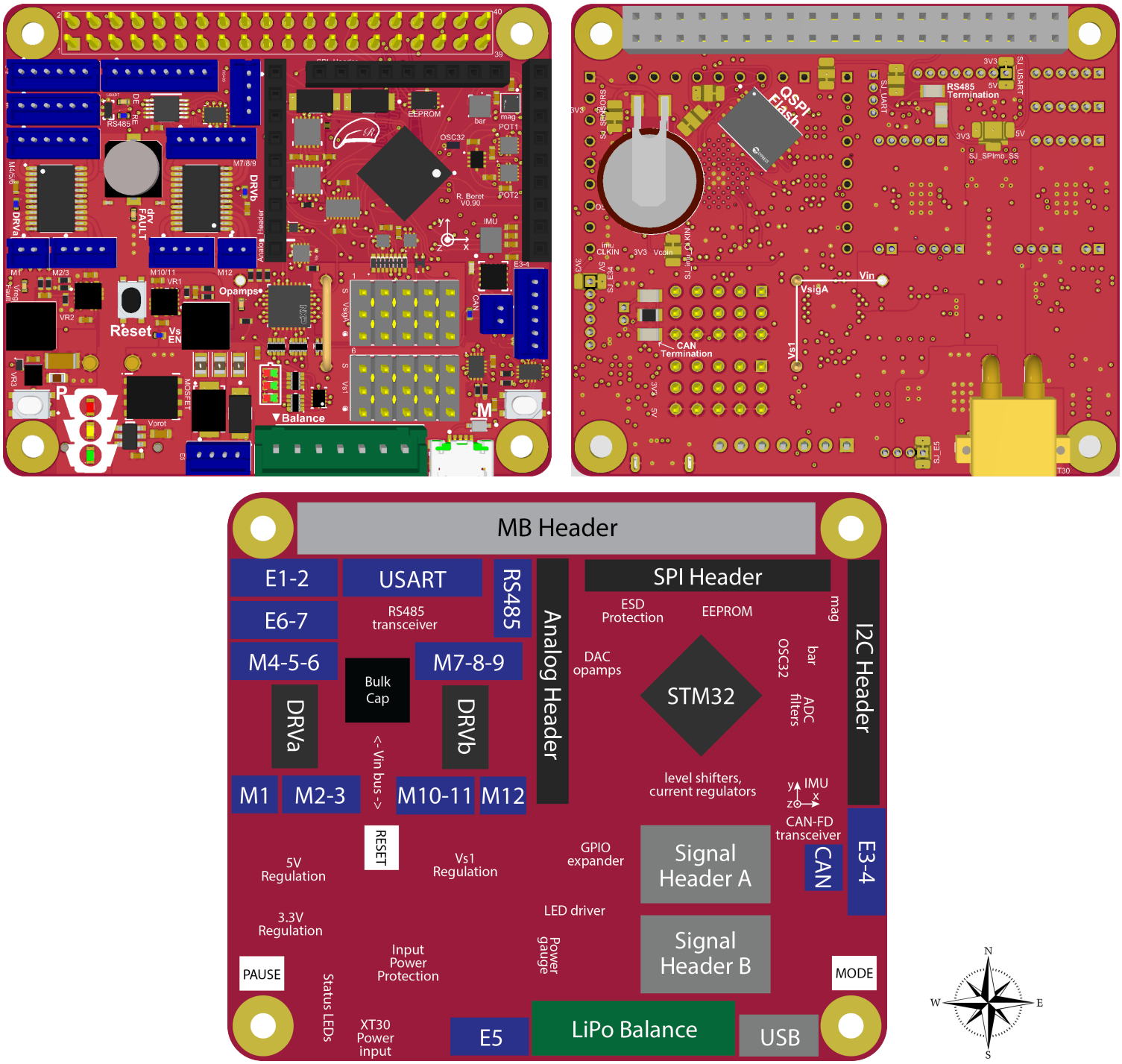
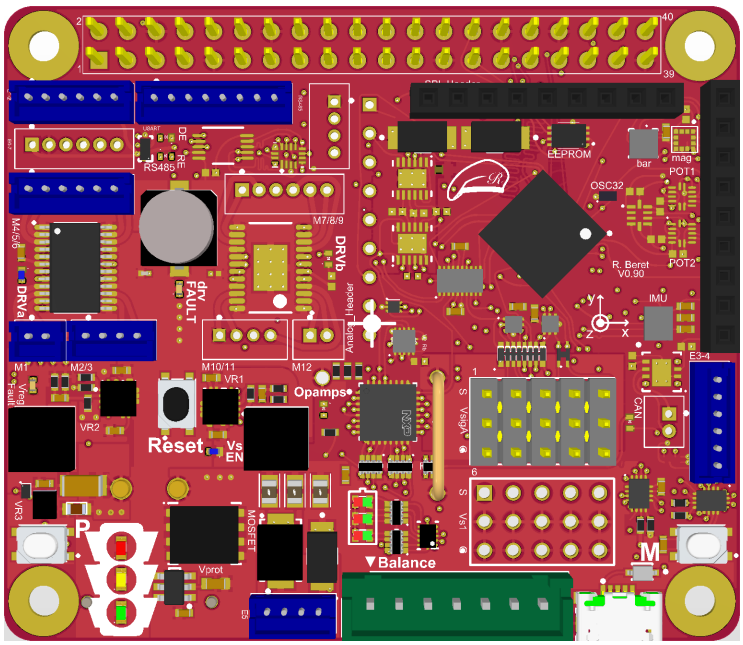
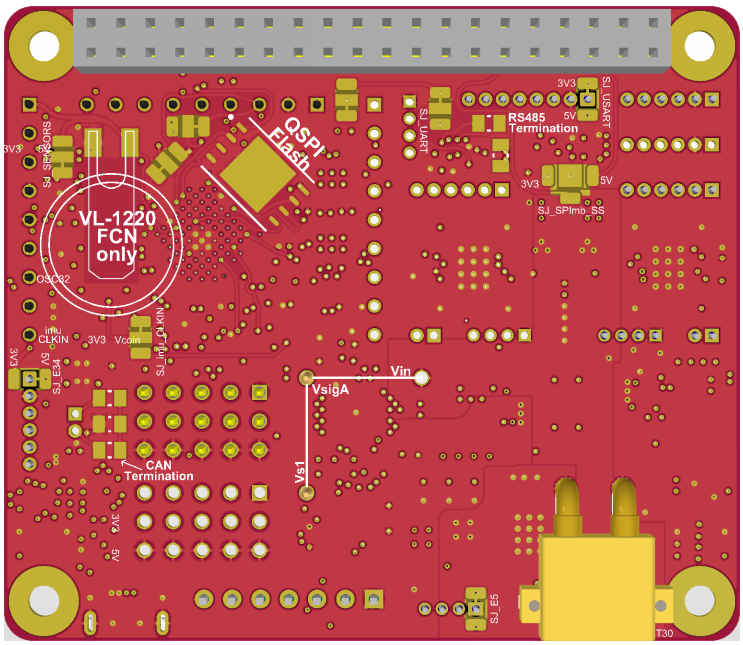


Figure 5.1: (top) Layout of the (left) front and (right) back sides, and (bottom) a functional cartoon, of the **Raspberry** Beret (**RPi** Header, **full size**, **5V MB**, **24 HB**, **CAN/RS485** busses). In the cartoon: blue denotes JST-ZH connectors of various sizes; gray is used for the MB Header, the 3x5 Signal Headers, and the USB Micro-B port; green is used for the JST-XH Balance connector. Beret Shields (§5.8) may be placed atop the Analog Header, the SPI Header, the I2C Header, and (optionally) the first row of the 3x5 Signal Header A (SigA), with the Beret's main logic ICs lying on the PCB beneath. The XT30 main power input and all solder jumpers are situated on the back side of the board; the user may (optionally, also on the back side) install a 6mm x 8mm Flash IC, a rechargeable VL-1220/FCN coin cell, and/or passives for RS485 or CAN termination.





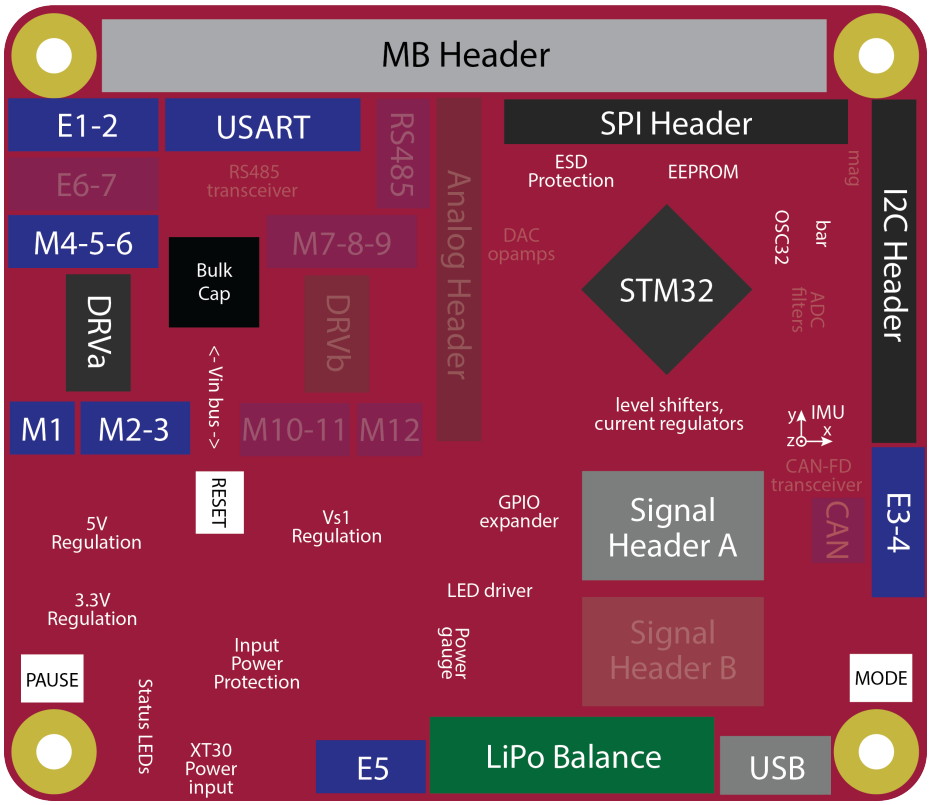


Figure 5.2: (top) Layout of the (left) front and (right) back sides, and (bottom) a functional cartoon, of the **Red** Beret (**RPi** Header, **full size**, **5V MB**, **12 HB**); for legend, see Fig. 5.1. Note that the entry-level **Red** Beret is simply a partially-populated **Raspberry** Beret, with the lower-cost STM32G4VB implemented (with 128KB flash instead of 512KB).

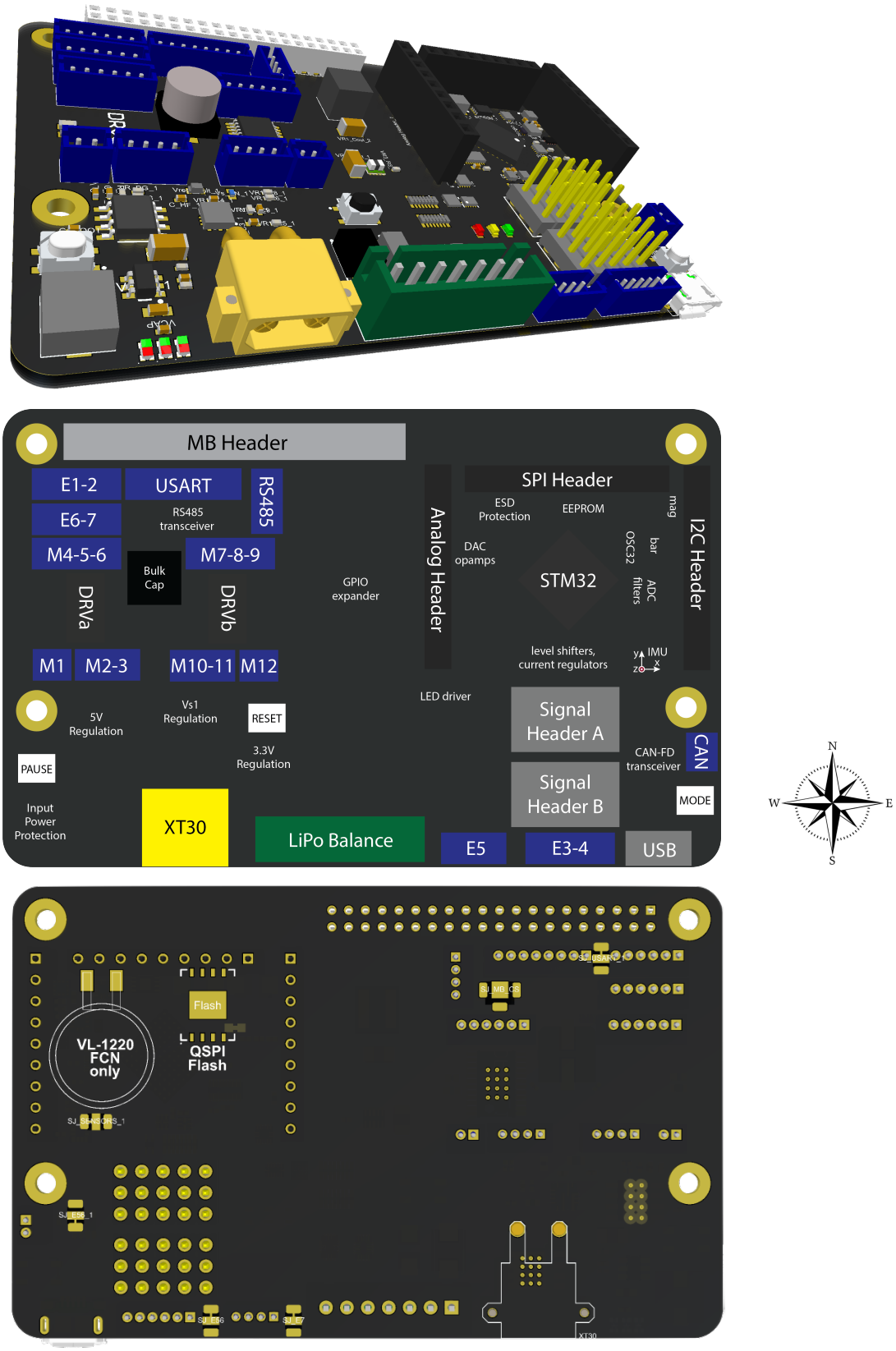


Figure 5.3: Layout of the (top) front and (bottom) back sides, and (center) a functional cartoon, of the **Black** Beret (**96B** Header, **full size**, **12V MB**, **24 HB**, **CAN/RS485** busses); for legend, see Fig. 5.1. **NOTE**: this board is still under development.

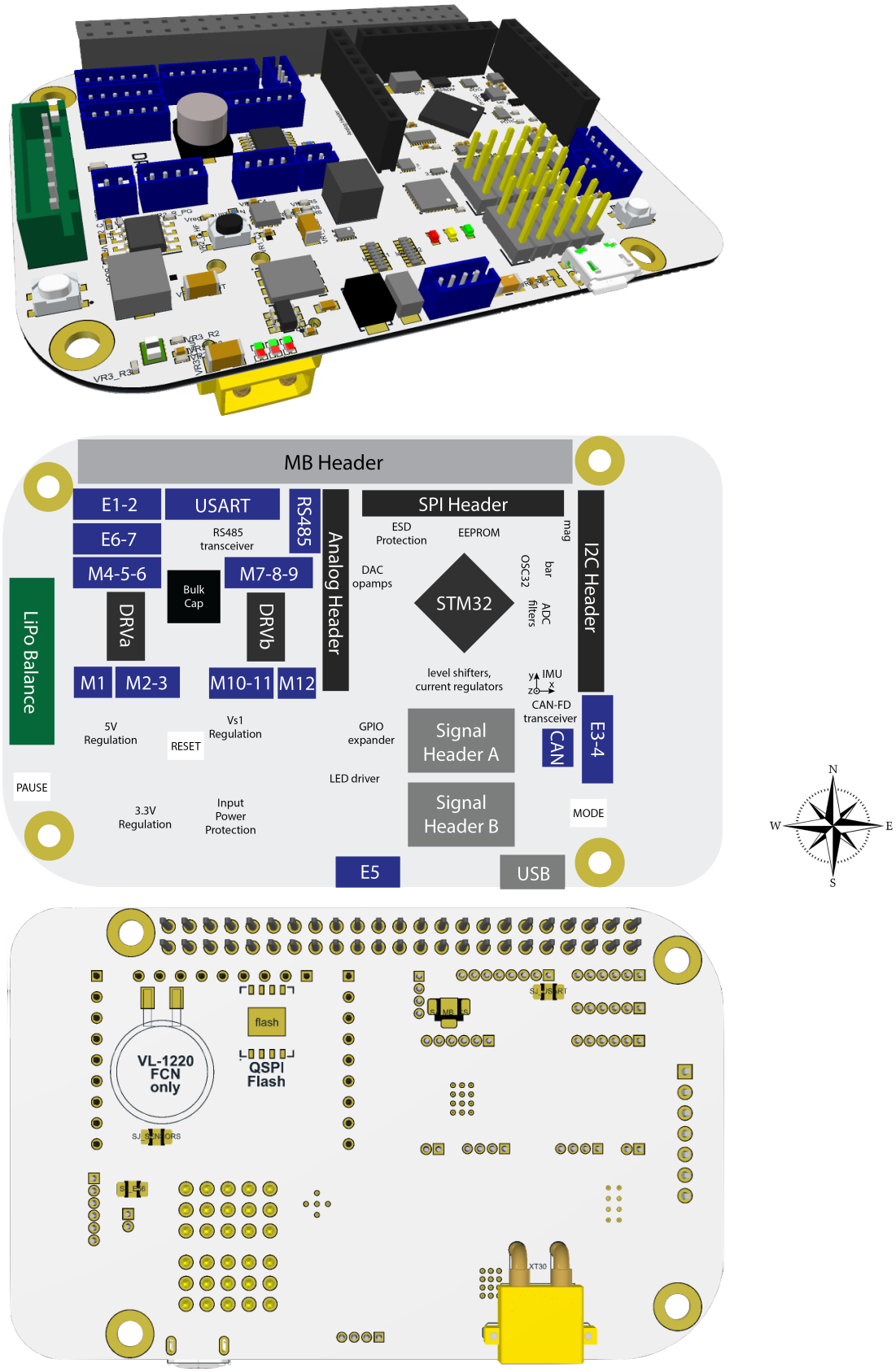


Figure 5.4: Layout of the (top) front and (bottom) back sides, and (center) a functional cartoon, of the **White** Beret (**BB** Header, **full size**, **5V MB**, **24 HB**, **CAN/RS485** busses); for legend, see Fig. 5.1. **NOTE**: this board is still under development.

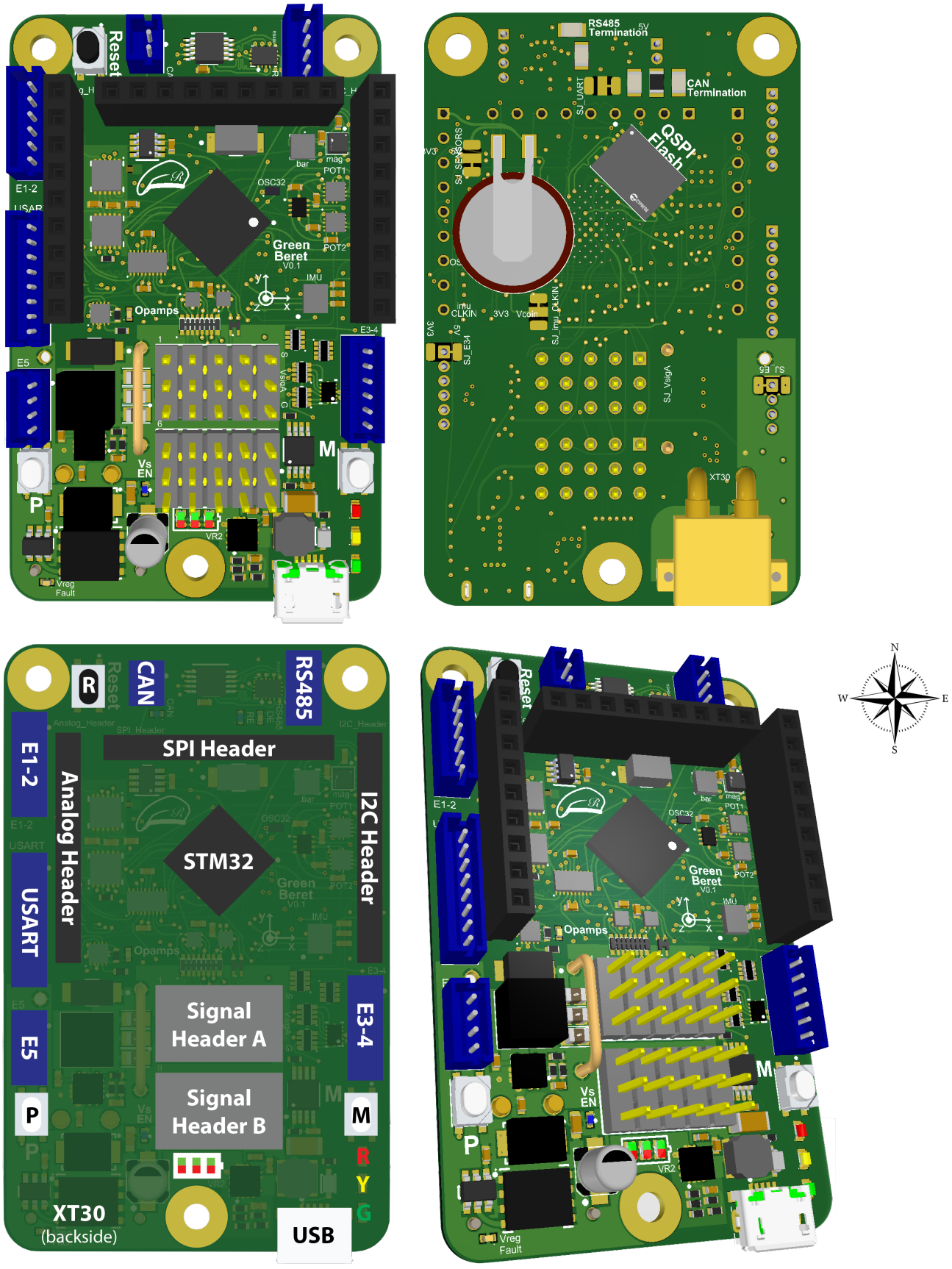


Figure 5.5: Layout of the (a) front and (b) back sides, (c) a functional cartoon, and (d) an oblique view, of the Green Beret (standalone, half size, 0 HB, CAN/RS485 busses); for legend, see Fig. 5.1.

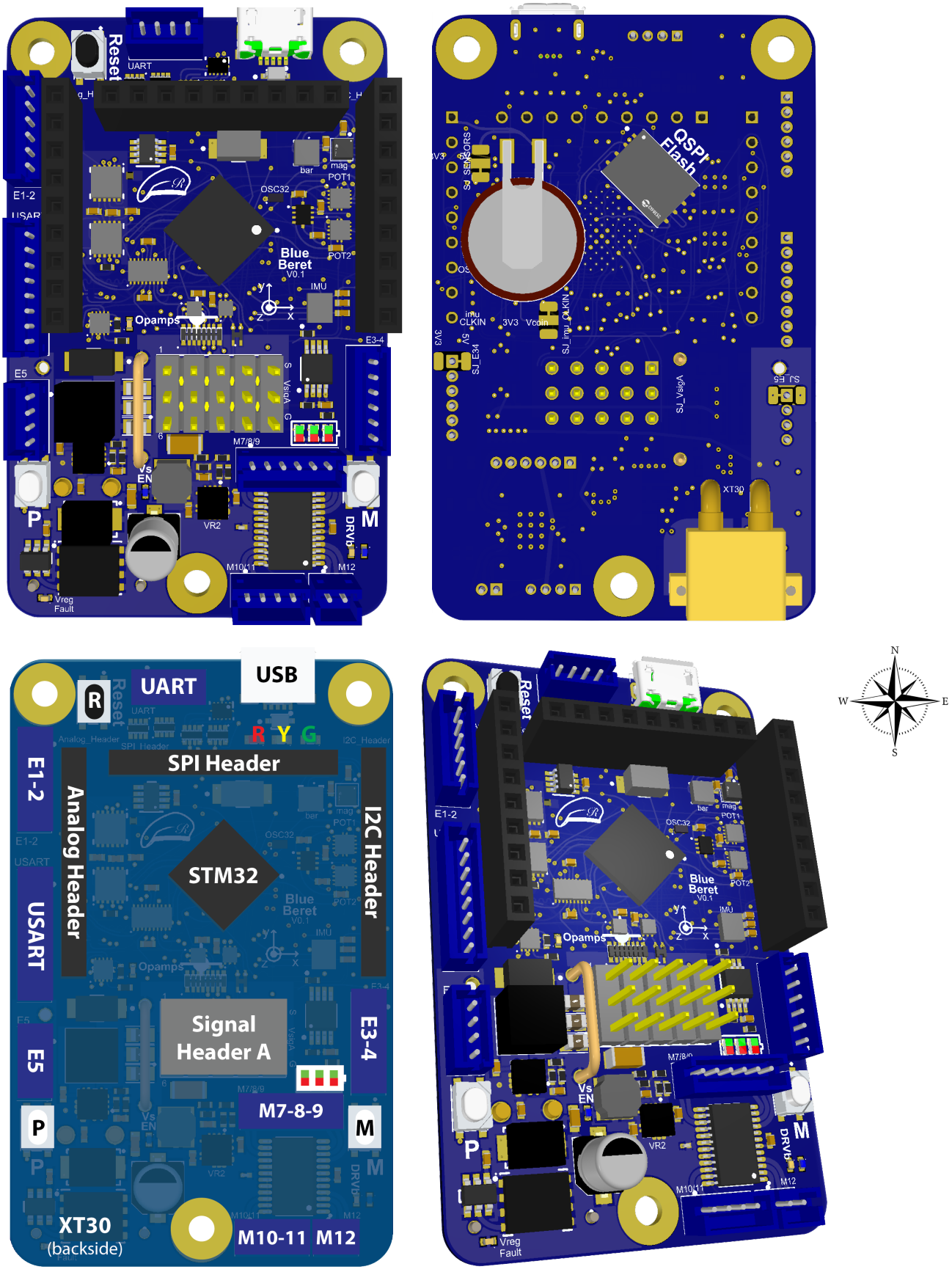


Figure 5.6: Layout of the (a) front and (b) back sides, (c) a functional cartoon, and (d) an oblique view, of the **Blue** Beret (standalone, **half size**, **12 HB**); for legend, see Fig. 5.1.

5.2 Power subsystem

5.2.1 Main power source (Vin)

As outlined in §5.1, the power input (Vin = **6.2V – 28V** on the **5V MB** Berets, and Vin = **12V – 28V** on the **12V MB** Beret, at up to 15A continuous / 20A peak) is brought onto the Berets via a backside XT30 connector. The XT30 jack provides the sole source of power for the motor drivers and servo headers on the Berets; power provided over the XT30 jack also drives all other circuits on the Beret and the attached MB, via the voltage regulators discussed in the sections that follow. Adapters are available separately to convert the connector on the main power source, if necessary, from XT60, Traxxas, EC3, and Deans (aka "T") connectors, 5.5 mm x 2.1 mm barrel jacks[8], and 4.75 mm x 1.65 mm barrel jacks[9] to the XT30 standard[10].

WARNINGS: when using high-power batteries like LiPos, it is essential that the user be aware of several delicate issues regarding their selection, care, use, charging, storage, and disposal. One of the more thorough online guides available on this important subject is available here; key takeaways include:
i. invest in a high-quality charger appropriate to the batteries you will be using,
ii. though LiPo cells can operate from 3.0 V to 4.2 V, a range of 3.1 V to 4.1 V extends battery life significantly,
iii. always balance while charging, as some cells will discharge faster than others (especially in older batteries),
iv. never use parallel charging boards,
v. store batteries at about half charge (~3.8V per cell for LiPos),
vi. don't tug on bare wires (AB Clips can help),
vii. watch for swelling (dispose of swollen battery immediately), and
viii. always dispose of batteries properly.
Keeping an inexpensive tester handy for checking the charges of all cells of a given battery is often useful.

DC motors and steppers connected to a Beret operate directly at Vin, after the TVS and reverse-voltage protections (see §5.2.3); this power is modulated by the H-bridges (§5.3) to control the motor speed and direction. Y adapters may be inserted between the power source and the Beret to drive high-current servos, ESCs, and BLDC motors directly off the power source, at Vin. The choice of the voltage Vin of the power source (e.g., a 2S to 6S LiPo) should thus be made according to the voltage required the motors and steppers (and, the high-current servos, ESCs, and BLDC motors) to be used. Note that the voltage of a LiPo battery reduces significantly as the battery is discharged; tuned feedback gains used to drive the motors (see §5.3) may be scaled inversely with the battery voltage to offset this effect over time. In contrast, the behavior of most servos and ESCs, which incorporate their own feedback, is relatively insensitive to the voltage supplied, as long as it remains within the recommended operating range; selecting Vs1 (for low-current servos/ESCs) or Vin (for high-current servos/ESCs) near the upper end of this operating range generally provides increased maximum torque.

5.2.2 Supplemental power sources

As discussed in §5.2.1, if main power is provided to the Beret over the XT30 connector (which is, in fact, the only way that the Beret will actually power the motor drivers and signal headers), then this power also drives the STM and all other circuits on the Beret, as well as the attached MB, as detailed in the sections that follow. There are a few other places that supplemental power might be brought onto a Beret (or, to a Beret+MB system), however, that the user also needs to be well familiar with.

[8] Wall adapters with 5.5 mm barrel jacks, with positive 2.1 mm pins and negative sleeves, is an emerging standard in robotics applications that the user should stick with. **Warning:** this standard for 5.5 mm barrel jacks is not universal; some 5.5 mm barrel jacks have either negative pins with postive sleeves, and/or 2.5 mm pins. If you are using a wall adapter with such a barrel jack, which is not recommended, a custom adapter to the XT30 standard will be required; make absolutely certain you get the polarity right!

[9] Wall adapters with 4.75 mm barrel jacks, with positive 1.65 mm pins operating at 12V, is the power supply standard for 96Boards.

[10] A substantial power source, capable of delivering 4A or more, is recommended for driving motors, servos, etc. with a Beret.

In normal operation, power is provided to a Beret via ONE of the following three sources:

1. the main XT30 power input on the Beret (connected to a high-power battery or a wall wart), OR
2. via the MB Header, up from a MB that is powered directly, OR
3. via the USB Micro-B connector on the Beret itself (connected, e.g., to a laptop or desktop computer).

Modes 2 and 3, which are discussed further in the following few paragraphs, may be useful at times for programming the STM on the Beret, for reading the Beret's onboard sensors, and for testing various other devices connected to the Beret's JSTs. **WARNING:** Mode 1 and mode 2 must NOT be used at the same time, to eliminate the possibility of these two power sources interfering with each other. Note that mode 3 is protected with a diode, and may thus be wired in conjuction with mode 1 or mode 2 (or, used stand alone) without concern.

Power from the MB, up through the MB Header, to the Beret

As mentioned above, RPi, 96B, and BB (and compatible) motherboards can all be powered by a Beret over the corresponding MB Header. Power may INSTEAD be provided to such motherboards directly, e.g.:
- via USB C on an RPi-4B or BeagleBone AI,
- via a USB Micro B input attached to a substantial (2A to 3A) 5V wall wart on RPi-2 and RPi-3 models,
- via the DC input jack on the BeagleBone Black and 96B CE boards,
etc. If power is provided to such motherboards directly, in most cases this power is transmitted back up to the Beret via the same MB Header pins that are otherwise used to transmit power down to the MB from the Beret. When this happens, the Vmb circuit on the Beret is energized with sufficient power (again, in most cases) to drive the Vmb->3.3V switching regulator (see §5.2.7), thus booting the STM on the Beret, and powering up the Beret's onboard sensors (see §5.4) as well as any small devices connected to the Beret's JSTs.

WARNING: As mentioned above, one should NOT provide power directly to both the Beret (via the XT30) and an attached MB[11]. Doing this can fry the Beret, the MB, and/or one or both of the power sources used.

5V power from the USB Micro-B port on the Beret

An unpowered, isolated (not connected to any MB) Beret may also, conveniently, be powered via the 5V line on the USB Micro B connector on the Beret, which may also be used for programming the (3.3V) STM on the Beret, for reading the Beret's onboard sensors, and for testing various other small devices connected to the Beret's JSTs. This programming mode is quite convenient, as it only requires a single standard USB cable (Type A Male to Micro B Male), in addition to the Beret and your laptop/desktop computer.

Unfortunately, the current capability of USB ports on most laptop/desktop computers is not sufficient to drive both the Beret and a connected MB, and attempting to do so is thus not advised.

Note that the USB specification prohibits a USB client from back-driving a USB host with 5V power on the USB voltage bus. Doing such can, in fact, cause major damage to a (potentially, expensive) USB host (i.e., your laptop/desktop computer). To protect against this possibility, a schottky diode (rated to 10V/3A on the 5V MB Berets, and rated to 40V/3A on the **Black**, **Green**, and **Blue** Berets), with a 0.35 to 0.55V voltage drop, is implemented between the 5V power bus of the USB Micro B connector on the Beret and the Vmb line on the Beret (or, the Vin line on the **Green** and **Blue** Berets). Thus, when powered by the USB Micro-B port alone, the Vmb (or, Vin) line will be held to about 4.5V, which is sufficient to generate 3.3V and fire up the STM.

When the Beret is otherwise powered (either via the XT30, or by the MB via the MB Header), the diode mentioned above will be under reverse bias, and no current will flow (either in to, or out of) the 5V power bus on the USB Micro B connector on the Beret; this configuration is thus considered to be **safe**.

[11] In certain special cases, this situation is known to be ok. Specifically, on the **5V MB** Berets, the Vin->Vmb switching regulator (see §5.2.5) is operated in a mode that can handle ~5V being driven at its output, whether or not power is provided (from the XT30) at its input [thus, no protection diode (aka ZPD) is required to isolate the Vin->Vmb regulator on the Beret]; note also that the last few paragraphs of the RPi HAT design guide specifically state conditions in which ~5V may safely be provided over the MB Header to an already-powered RPi. Regardless, there truly appears to be no practical reason to push your luck by trying this. So, don't.

RTC backup battery (VL-1220/FCN rechargeable 3V coin cell)
All Berets come configured with backside solder pads for a VL-1220/FCN (only!) rechargeable 3V coin cell. **Warning**: special care is required careful when soldering the VL-1220/FCN battery bracket onto the Beret. Panasonic's Lithium Handbook, which describes the VL-1220/FCN on pages 56-59, gives the following advice (on page 80): *Do not allow the soldering iron to make direct contact with the body of the battery. Proceed with the soldering quickly (within 5 seconds) while maintaining the iron tip temperature at about 350°C, and do not allow the temperature of the battery body to exceed 85°C.*

The VL-1220 keeps the real-time clock (RTC) of the STM powered up during replacements of the main battery (for recharging), and facilitates the scheduled wake-up of the Beret and the arrached MB from a low-power sleep (aka VBAT mode), during which main power is turned off to both the Beret and the MB (see §5.5.1). The VL-1220 battery is automatically recharged, when necessary, by the 3.3V line on the Beret when the STM is in run mode.

5.2.3 Reverse-voltage, over-current/short-circuit, and ESD protections

All Berets implement a TVS diode across the XT30 input, and TI TPD6E004 TVS diode arrays on USB and S1-S10, for protection from Electrostatic Static Discharge (ESD, i.e.. voltage spikes). On the 5V MB Berets, the {Vs1, 5V, 3.3V} lines are protected from voltage spikes by {13V, 5.6V, 3.6V} 1.5W zener diodes; on the 12V MB Beret, the {Vs1, 12V, 5V, 3.3V} lines are protected using {13V, 13V, 5.6V, 3.6V} 1.5W zener diodes. 100 Ω resistor arrays are used on S1 - S10 for over-current (short-circuit) protection. The modern voltage regulators (§5.2.4-5.2.7), motor drivers (§5.3), and CAN & RS485 transceivers (§5.6.4) on the Beret provide further over-current and ESD protections. A TI CSD18510Q5B MOSFET [with $R_{DS(on)}$ = 0.96 mΩ and $t_{d(off)}$ = 44 ns] is implemented at the XT30 to provide reverse voltage protection, and to turn off the board when necessary; the gate voltage of this MOSFET is optimally adjusted by a TI LM74700-Q1 ideal diode controller, wired as suggested in figure 21 of its datasheet. The battery gauge LEDs (see §5.6.8) are illuminated whenever this main MOSFET is enabled [and, thus, the Vmb (5V or 12V) and 3.3V switching regulators (see §5.2.5-5.2.7) are powered up].

WARNINGS: Notwithstanding the modern power protections implemented, as outlined above, it is still quite possible to "release the magic smoke" [12] from a Beret; the user is thus strongly urged to carefully:
- ensure proper polarity at the XT30 (red/positive wire to the right when viewed from above; see Figures 5.1-5.6),
- keep the power input at the XT30 at or below 28V, preventing any voltage spikes beyond this value, and
- avoid any short circuits between any two pins (see Table 5.2), or draw current on any subsystem beyond the maximum current values highlighted in §5.2.4 - §5.2.8, and §5.3, of this datasheet.

In §5.2.4 through §5.2.8, the voltage regulation circuits on the Berets are described in detail.

5.2.4 Vin->Vs1 switching regulator

The **Vin->Vs1 switching regulator** implemented on all five of the Berets, for generating a software-adjustable Vs1 = 4.8V to min(12V, 0.8*Vin) at up to 6A (for SigA, SigB), is the TI TPS56637, coupled with a 5.6 μH inductor and three 22 μF output capacitors. The TPS56637 converter is wired as suggested in figure 17 its datasheet, with EN and PG wired to Vs1_EN and Vreg_FAULT on the GPIO expander (see Table 5.6), taking[13] R7=6.49 kΩ and replacing R6 by a 28.7 kΩ resistor in series with half of POT3, a TI TPL0102-100 dual digital pot, in Rheostat

[12] It is sometimes said that "magic smoke" makes an IC work, as whenever this smoke escapes, the IC ceases to function; some vendors even sell replacement magic smoke. **Warning**: more seriously, the Beret can deliver very high currents indeed; severe injury or death may result from its misuse, so extreme caution and adherance to the warnings in this datasheet is absolutely required.

[13] These resistor values were selected by noting eq. 5 in the TPS56637 datasheet, applying a 3% margin to the desired limits, solving the following simultaneous systems of equations in Matlab, and rounding to the nearest common resistor values:

```
syms x y; S=solve( 3.3/1.03==0.6*(1+y/x), 12*1.03==0.6*(1+(y+100)/x) ); x=eval(S.x), y=eval(S.y)
```

mode, that is adjustable electronically[14] from 0 to 100 kΩ in 256 increments, providing a maximum peak-to-peak ripple current accoring to eq. 7 of the datasheet of up to 2.4 mA (at Vs1=12V). A blue status LED in this quadrant illuminates whenever the Vs1 voltage regulator is enabled, and an amber status LED illuminates whenever any of the voltage regulators signal a fault condition (see §5.6.8).

5.2.5 Vin->Vmb switching regulator

The **Vin->Vmb switching regulator** implemented, for generating both Vmb = 5.1V at up to 6A on the **5V MB** Berets and Vmb = min(12V, 0.8*Vin) at up to 6A on the **12V MB** Berets, is another TI TPS56637 again coupled with a 5.6 μH inductor, and a 68 μF output capacitor on the **5V MB** Berets, or a 47 μF output capacitor on the **12V MB** Berets. In both cases, the TPS56637 converter is again wired as suggested in figure 17 of its datasheet, with EN tied to Vin and PG to Vreg_FAULT, using values from its table 4 for 5V or 12V output as appropriate.

The Vin->Vs1 and Vin->Vmb switching regulators, as well as the motor drivers (see §5.3), are placed close to each other, allowing them to share the same 100 μF bulk capacitor near their respective inputs.

As mentioned previously, on the **12V MB** Beret, the 96B motherboard, if one is attached, down-regulates from the Vmb=12V line (and, passes back via the low-speed header) 5V at up to 1A; the 12V MB Beret makes use of this 5V line directly, and thus does not itself have a 5V regulator.

5.2.6 Vin->3.3V switching regulator

The **Vin->3.3V switching regulator** implemented on the **Green** and **Blue** Berets, for generating 3.3V at up to 3A, is another TI TPS56637 with a 2.2 μH inductor, and a 33 μF output capacitor. The TPS56637 converter is again wired as suggested in figure 17 of its datasheet, with EN tied to Vin and PG to Vreg_FAULT. The Vin->Vs1 and Vin->3.3V switching regulators are placed close to each other, allowing them to share the same 18 μF bulk capacitor near their respective inputs.

5.2.7 Vmb->3.3V switching regulator

The **Vmb->3.3V switching regulator** implemented on the **5V MB** Berets, for reducing Vmb = 5.1V to 3.3V at up to 3A, is the fixed voltage TI TPS6208833, coupled with a 220 nH inductor and a 22 μF output capacitor. This converter is wired as suggested in figure 6 of its datasheet, with EN tied to Vmb and PG to Vreg_FAULT.

On the other hand, the **Vmb->3.3V switching regulator** implemented on the **12V MB** Beret, for reducing Vmb = 12V to 3.3V at up to 3A, is the TI TPS62913, coupled with a 4.7 μH inductor and a 47 μF output capacitor. This converter is wired as suggested in figure 8.1 of its datasheet (with Rfbt=15.4 kΩ, Rfb=4.87 kΩ, Cff open, and Lf removed), with EN tied to 5V and PG to Vreg_fault.

The Vin->Vmb and Vmb->3.3V switching regulators are placed close to each other (on both the 5V MB and 12V MB Berets), allowing the output capacitor of the former to serve as the input capacitor of the latter.

[14] Specific care is taken in the software driving this digital potentiometer in order to cycle the power to the switching regulator off completely before adjusting the value of the resistance of the digital potentiometer (leveraging a look-up table, which is calibrated on the fly, and a false-position search based on this look-up table) in order to, whenever the value of the digital potentiometer is changed, both (a) reset the (internal) compensation coefficient of the switching regulator IC, and (b) prevent Over Voltage Protection (OVP) from kicking in and triggering a soft restart of the switching regulator IC.

5.2.8 Vs2 power opamp

The **Vs2 power opamp** implemented, for generating a second adjustable voltage Vs2 = 1.2V to 2.1V at up to ± 400 mA, is provided by a TI ALM2402-Q1 high-power opamp, wired as described in Figure 5.10a-b (and surrounding discussion). As opposed to the other regulated voltages mentioned in the above four subsections, Vs2 can source or sink up to 400 mA.

Note that, in addition to the Vs1, 12V, 5V, 3.3V, and Vs2 lines discussed above, DAC1 and DAC2 (see §5.7.1) can also each source (or, sink) 400 mA, and can be set as constant (or, time varying) voltage sources, if needed, in the 0V to 3.3V range.

Note that Vs2, DAC1, and DAC2 are not available on the **Red** Beret and, again, that Vmb and 5V are not available on the **Green** and **Blue** Berets.

5.2.9 Switching default power on various connectors and sensors

The power provided to {E1-2, E6-7} may be switched between {3.3V, 5V, Vs1, off} by a TI TS5A3359 multiplexer, via the Venc_3.3V and Venc_5V Beret GPIOs (see §5.5).

The nominal 3.3V power provided to {E3-4, E5, USART, USART} may be switched to 5V via backside solder jumpers, as illustrated in Figures 5.1-5.6.

The nominal 3.3V power supplied to the IMU, magnetometer, and barometer may be switched to the coin cell (i.e., Vcoin) via a backside solder jumper, as discussed further in §5.4.1.

The nominal Vs1 power supplied to Signal Header A may be changed to Vin by reorienting the 12A shunt connector. TODO: provide some photos of how to make these changes.

5.2.10 Charging and voltage monitoring of Vcoin

When in run mode, the STM monitors the voltage, Vcoin, of the VL-1220/FCN coin cell (if installed) using the STM's internal ADC1_IN17 channel, and (if necessary) recharges this cell, which generally operates in the 2.6V to 3.05V range, using the 3.3V power bus. If it is ever found that 2.6V < Vcoin < 2.8V, software on the STM selects VBRS=1 in the PWR_CR4 register on the STM (see the STM datasheet), thereby charging the coin cell (until Vcoin = 3.0V, after which charging is turned off) through a 1.5 kΩ resistor internal to the STM, while limiting the charging current applied to

$$(3.3\,\mathrm{V} - 2.6\,\mathrm{V})/1.5\,\mathrm{k\Omega} \approx 0.47\,\mathrm{mA}$$

or less, as per the VL-1220 specification on page 57 of Panasonic's Lithium Handbook, which requires this charge current to be 0.5 mA or less. If it is ever found that 2.0V < Vcoin < 2.6V, software on the STM selects VBRS=0 in the PWR_CR4 register on the STM, thereby charging the coin cell (until Vcoin = 2.6V, after which VBRS is set to 1) through a 5 kΩ resistor internal to the STM, while limiting the charging current applied to

$$(3.3\,\mathrm{V} - 2.0\,\mathrm{V})/5\,\mathrm{k\Omega} \approx 0.26\,\mathrm{mA}.$$

or less. If ever the charge of the VL-1220 coin cell is found to be less than 2.0V, the battery is flagged as either dead or not installed, and no charging is attempted.

	DRVa						DRVb (**24 HB** Berets only)					
Outputs on DRVs	6,4	9,11	3,10	5,1	7,12	8,2	3,10	11,9	4,6	8,2	12,7	1,5
JST connector on Beret	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12

Table 5.3: Connections between DRVs and the motor JSTs M1 through M12.

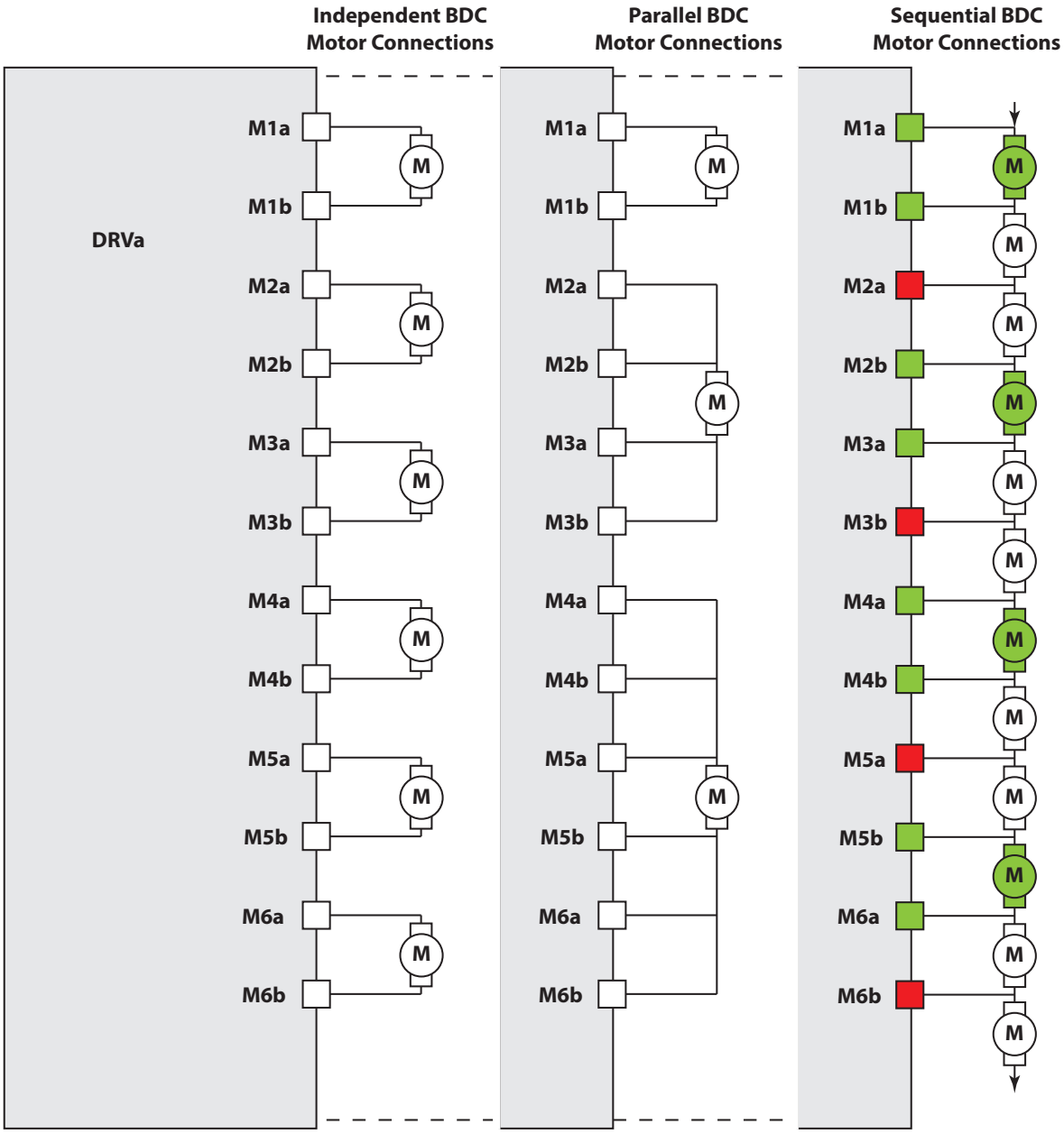


Figure 5.7: Illustration of the independent, parallel, and sequential modes of operation of the 12 outputs of DRVa. For convenience when operating in independent mode, these outputs (reordered and renamed as described in Table 5.3) are arranged on the Beret as six pairs of outputs on each DRV8912-Q1 (M1-M6 on DRVa and, if present, M7-M12 on DRVb).

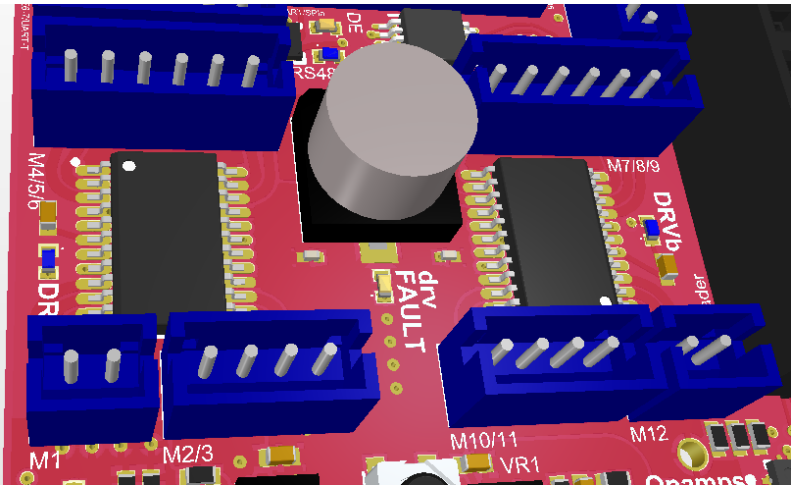


Figure 5.8: Closeup of the connections to the two DRV8912-Q1 motor drivers on a **Raspberry** Beret.

5.3 Control of brushed DC motors & steppers with the DRV8912-Q1

A central feature of the Berets is the TI DRV8912-Q1 brushed motor driver(s) (hereafter, DRVa and/or DRVb) about which they are built[15] (see Figure 5.8). This powerful subsystem draws a maximum of 6A per DRV, or 1A max per channel. Each of these remarkably versatile motor drivers incorporates **12 half bridges**, which may be configured in any of a myriad of ways, and **4 synchronized internal PWM generators** (operating at about **80 Hz**, **100 Hz**, **200 Hz**, or **2000 Hz**, selectable in software) to drive them. The outputs of the DRVs are broken out on the Beret one pair at a time, labelled M1 through M6 on DRVa (and, if present, M7 through M12 on DRVb). To simplify layout, the Beret reorders the outputs of the DRVs, as shown in Table 5.3, with a single combined drv_SLEEP channel and a single combined drv_FAULT channel tied to the GPIO expander (see Table 5.6). A blue status LED near the DRVs illuminate whenever they are enabled, and an amber status LED illuminates when the DRV signals a fault (see §5.6.8). Note that the outputs of the DRVs may also be used to drive high-power PWM-driven circuits like LEDs (again, max 6A total per DRV, or max 1A per output).

Note that the Beret's SPIdrv channel (that is, the STM's SPI4 channel), operating with a single SS line in daisy-chain mode, gives the STM a dedicated high-speed connection to DRVa and DRVb.

The user might at first just plug into the M1 – M12 discrete JSTs directly, and use the half-bridge outputs one pair at a time, in **independent mode**, for bidirectional control on each DRV of up to 4 "independent" motors running at up to 1 A each, and up to 2 "slave" motors operating in brake, coast, full forward, or full reverse, or duplicating the PWM frequency and duty cycle of one of the "independent" motors. If 6 motors are wired to one DRV at the same time, which 4 are chosen to be the "independent" motors, and which 2 as the "slave" motors, may be redefined on the DRV whenever necessary.

Notably, by ganging the DRV outputs together in parallel mode, and/or stringing the DRV outputs along in sequential mode (see Figure 5.7), a wide array of different motor configurations also becomes possible, as discussed further in the following two subsections. Note, of course, that the independent, parallel, and/or sequential modes can actually be combined on each of the DRV(s), for a wide variety of possible configurations for developing complex yet practical systems.

[15] The DRV8912-Q1 drivers described here provide a compact general-purpose solution for the control of a wide range of small (up to 28V and 6A) brushed-DC (BDC) motors (in forward, reverse, coasting, and braking operational modes, and alternating between two such modes with PWM) and stepper motors. The other main type of motors used in mechatronic systems, brushless-DC (BLDC) motors, require more carefully tuned electronic coordination, with BLDC motor drivers fairly closely matched to the BLDC motors to be used. BLDC motor drivers are therefore located on Beret Shields (see §5.8) when using the Beret ecosystem.

5.3.1 Parallel mode

Parallel mode gangs the outputs of 2 to 6 pairs of half bridges on each DRV together, using external wire harnesses, to create full H-bridges for powering higher-current motors, thus driving (on each DRV):
- 1 motor at 1A, 1 motor at 2A, and 1 motor at 3A (as shown in Figure 5.7),
- 1 motor at 6A,
- 2 motors at 3A, etc.

Warnings: on the **24 HB** Berets, DRVa & DRVb operate independently, and thus asynchronously. Thus:
a. any individual motor may be wired to DRVa or DRVb, but never to both;
b. parallel connections must join together the same number of DRV outputs on the left and right sides of any motor using a custom wire harness, using a sufficient wire gauge to handle the resulting current; and
c. parallel connections must synchronize, via appropriate programming, the wires that are ganged together (e.g., in the configuration shown in Figure 5.7, {M2a, M2b} and {M3a, M3b} must be synchronized to form the left and right sides of the 2A motor, and {M4a, M4b, M5a} and {M5b, M6a, M6b} must be synchronized to form the left and right sides of the 3A motor, see, e.g., §8.3.1.1.3 and §8.3.1.1.4 of the DRV8912-Q1 datasheet).

5.3.2 Sequential mode

Sequential mode, on the other hand, strings the output of one motor together with the input of the next, in a serial fashion, for bidirection control of a remarkable total of **up to 12 motors at up to 1 A on each DRV**, albeit at reduced duty cycles (i.e., not all at once). Note that interleaving half-bridges must to be turned off at any instant for sequential mode to work correctly, and the voltage used must be slightly above the stall torque of a single motor, so that two motors can not be driven by the supply voltage when applied in series. Thus, for example, if 12 motors are hooked up to a single DRV as shown in Figure 5.7, then:

A. the 1st, 4th, 7th, & 10th motors are driven for a while (with M2a, M3b, M5a, & M6b off), then
B. the 2nd, 5th, 8th, & 11th motors are driven for a while (with M1a, M2b, M4a, & M5b off), then
C. the 3nd, 6th, 9th, & 12th motors are driven for a while (with M1b, M3a, M4b, & M6a off);

these three steps then repeat from the beginning. PWM with independently controllable duty cycles may be used on each channel (e.g., at 2000 Hz) to run these motors (4 at a time on each DRV) at partial power and in either direction, while the cycling between these three steps happens much more slowly (e.g., at 10 to 100 Hz). If, periodically, 4 to 8 of the motors hooked to each DRV are not used (e.g., in sequential assembly-line operations), then these motors may be grouped together as appropriate, and the corresponding step(s) in the above-described cycle can be skipped, improving the smoothness in the driving of the remaining motors (by reducing the time that they spend in a step that turns them off).

5.4 IMU, magnetometer, and barometer

All six Berets includes three built-in environmental sensors:

- a TDK ICM-42688-P 6-axis IMU (3-axis accel + 3-axis gyro + thermometer), connected via SPI,
- an ST LIS3MDLTR 3-axis magnetometer, connected via I2C, and
- an ST LPS22HB barometer + thermometer, connected via I2C.

The dedicated SPIimu connection between the STM and the IMU facilitates, if needed, remarkably fast update rates (up to 32 kHz) for IMU data, suitable for problems in which high-frequency mechanical vibrations need to be characterized. Data from the magnetometer and barometer is usually low-pass filtered (on the sensors themselves) to reduce noise. The magnetometer is generally operated at 80 Hz or less, and the barometer at 75 Hz or less; for such signals, communication over the shared I2Ca bus is thus sufficient.

Note that the extensive bus connectivity (UART, I2C, SPI, USB, CAN) provided by the Beret allows, of course, many additional sensors to be attached easily, as necessary for the user's particular application.

If the center of the top surface of each Beret is taken as the origin, then the centers of the mounting holes and the IMU coordinate system are situated as defined in Table 5.1. Note also that, on each Beret, the magnetometer is located in a corner of the logic quadrant that is as distant as possible from the board's power electronics, thus minimizing the contamination of its readings by stray EM fields.

As depicted by the axes printed next to the STM, the native (x, y, z) coordinates of both the IMU and the magnetometer on the Beret are (in the reference orientation depicted in Figs 5.1-5.6, where "up" is towards the viewer when looking at the top side of the board) aligned with the (East, North, Up) [a.k.a. ENU] directions, respectively. Positive rotations are given by right-hand rotations about each of these axes. For example, as in the ISO 8855 automotive standard, if the body-fixed (x, y, z) axes are taken as vectors out the (front, left, top) of a vehicle, respectively, then positive [a.k.a. 3-2-1 Tait-Bryan] rotations about the (z, y, x) coordinates may be referred to as (yaw, pitch, roll), and denoted (α, β, γ), respectively, where each of these rotations being positive is given by, respectively, (nose to the left, nose down, right side down).

Other coordinate conventions are easily derived from the native (x, y, z) coordinates used by the Beret. For example, taking $(\hat{x}, \hat{y}, \hat{z})$ as (North, East, Down) [a.k.a. NED] on the board (again, in the reference orientation depicterd in Figs 5.1-5.6), with positive (right-hand) rotations about each denoted $(\hat{\alpha}, \hat{\beta}, \hat{\gamma})$, it follows that $(\hat{x}, \hat{y}, \hat{z}) = (y, x, -z)$ and $(\hat{\alpha}, \hat{\beta}, \hat{\gamma}) = (\beta, \alpha, -\gamma)$; linear and angular velocities and accelerations in these two different coordinate conventions are related similarly. For example, as implemented broadly in the aerospace industry, as well as by the SAE J670 and J1594 automotive standards, if the body-fixed $(\hat{x}, \hat{y}, \hat{z})$ axes are taken as vectors out the (front, right, bottom) of a vehicle, respectively, then positive rotations $(\hat{\alpha}, \hat{\beta}, \hat{\gamma})$ about the $(\hat{z}, \hat{y}, \hat{x})$ coordinates may again be referred to as (yaw, pitch, roll), where each of these rotations being positive is now given by, respectively, (nose to the right, nose up, right side down).

To estimate the time evolution the 6DOF configuration [position plus orientation] of the system to which the Beret is attached, in addition to the 6DOF rate of change of this configuration [together referred to as the 12DOF state of the system], one must integrate the raw linear acceleration and angular velocity data from the IMU, and (optionally, on a slower time scale) fuse this information with the absolute position and orientation measurements that may be obtained from the magnetometer, barometer, and/or attached GPS/GNSS unit, as well as the linear and angular velocity measurements that may be obtained from laser rangefinders, optical flow processing, etc. This complex task, discussion of which is beyond the scope of the present document, must be solved on the STM (not on the IMU itself).

However, the IMU does features a flexible set of built-in programmable digital filters, and includes a proprietary on-chip motion processing engine designed for gesture recognition and activity classification, and can also function effectively as a pedometer (see also §5.4.1).

Note that not all IMUs, magnetometers, and barometers are created equal. Far from it, in fact (buyer beware!). The specs on the sensors selected for the Beret are, as of 2021, best in class:

- the accelerometer noise is about 70 $\mu g/\sqrt{Hz}$, and the gyro noise about 2.8 mdps/$\sqrt{Hz}$,
- the sensitivity of the magnetometer is about ± 0.4 mT, and
- the relative accuracy of the barometer is about ± 1 Pa (i.e., ± 8.8 cm change in altitude at sea level[16]).

See the corresponding datasheets for further such characterizations, and compare such specs carefully when selecting IMUs, magnetometers, and barometers for your own board designs. Note in particular that many popular "9-axis" IMUs, which (conveniently) incorporate a 3-axis magnetometer as well as a motion processing engine to automatically estimate the system state (albeit, without incorporating inputs from auxiliary sensors such as barometers, GPS/GNSS units, laser rangefinders, etc), compromise on sensor sensitivity in order to fit more functionality onto a single IC; we have thus avoided using such a 9-axis IMU in the Berets.

Note also that the STM and IMU are slaved to the same external 32.7680 kHz MEMS oscillator (see §5.5.1) on the Berets, which keeps their clocks accurately in sync.

5.4.1 Data-ready and sensor-driven interrupts

The IMU, magnetometer, and barometer used on all six Berets are each capable of taking measurements at a pre-specified rate[17], and alerting the STM (via the imu_INT2_DRDY, mag_DRDY, and bar_INT_DRDY channels, as shown in Tables 5.5-5.6) as soon as there is fresh data ready (DRDY) to be read from the corresponding sensor, so that this sensed data may subsequently be used, with minimum latency, in feedback algorithms.

The IMU, magnetometer, and barometer are also capable of running in the background (without constant monitoring by the STM), and issuing interrupts when a variety of environmental conditions are detected. The IMU can be programmed to issue an interrupt to the STM, over the imu_INT1_DRDY and/or imu_INT2 channels, when any of the following events are detected:

- a step or tap,
- a tilt beyond 35° for more than a certain (programmable) period of time,
- a raise-to-wake or lower-to-sleep gesture, or
- when net accelerations exceed a certain (programmable) magnitude.

Similarly, the magnetometer can be programmed to issue an interrupt, over the mag_INT channel, when

- the magnetic field exceeds a (programmable) magnitude in the x, y, or z direction (selectable),

and the barometer can be programmed to issue an interrupt, over the bar_INT_DRDY channel, when

- the atmospheric pressure attains a (programmable) maximum or minimum threshold.

The {imu_INT1_DRDY, mag_INT, bar_INT_DRDY} channels may also be made connected (via backside solder jumpers on the Beret that are initially open) to the MB (see §5.5 for details).

As also noted in §5.2.9, the nominal power supplied to the IMU, magnetometer, and barometer is 3.3V, but this may be switched via a backside solder jumper to the coin cell (i.e., Vcoin). Combining this feature with the programmable interrupts discussed above, any of the above noted events can programmed to generate an interrupt that wakes both the Beret and the motherboard from a low power sleep mode, which operates solely on the coin cell.

[16] Note that atmospheric pressure decreases at a rate of about 11.3 Pa per meter increase in altitude at sea level.

[17] The IMU operates at datarates of 12.5 Hz to 32 kHz, the barometer operates at datarates of 1 Hz to 75 Hz, and the magnetometer nominally operates at datarates from 0.625 Hz to 80 Hz, though in fast mode the magnetometer can be operated from 155 Hz to 1 kHz. The IMU and barometer both incorporate convenient digital low-pass filters that may be enabled.

Figure 5.9: Connectivity of the primary STM busses on Berets, with rectangles denoting ICs, brown rectangles denoting environmental sensors, octagons denoting connectors, capsules denoting channels, ellipses denoting buttons & LEDs, and $()^a$ denoting optional backside components. See Table 5.4 for the correspondence between Beret channel names and the STM32 hardware channel names.

Beret location	Beret channel names	STM32 hardware channel names	§
USART JST	USARTb; UARTb; SPIb	USART3	5.6.5
RS485 JST	RS485; UARTa	UART5 (+ RS485 transceiver)	5.6.4
CAN JST	CAN	FDCAN1 (+ CAN transceiver)	5.6.4
E1-2 JST	E1-2; I2Cd	TIM3_CH1/2, TIM8_CH1/2; I2C4	5.6.1
E3-4 JST	E3-4; UARTt	TIM4_CH1/2, TIM2_CH1/2; USART1_TX	5.6.1
E5 JST	E5; UARTr	TIM1_CH1/2; LPUART1_RX	5.6.1
E6-7 JST	E6-7	TIM5_CH1/2, TIM20_CH1/2	5.6.1
Signal Header A	S1-S5; I2Cb, I2Cc	HRTIM_CHA1/A2/E2/E1, TIM16_CH1; I2C2, I2C3	5.6.2
Signal Header B	S6-S10	TIM15_CH1/2, TIM17_CH1, HRTIMCHD1/C2	5.6.2
SPI Header	SPIa; I2S, IR	SPI2, I2S2, IR_OUT	5.6.7
I2C Header	I2Ca	I2C1	5.6.6
Analog Header	DAC1, DAC2	DAC1_OUT1, DAC1_OUT2	5.7.3
MB Header	SPImb	SPI3	5.9
comm to DRVs	SPIdrv	SPI4	5.3
comm to IMU	SPIimu	USART2	5.4
comm to flash	QSPIflash	QSPI1_BK2	5.5.2

Table 5.4: High-level correspondence between various Beeret channel names and the corrgesponding STM32 hardware channel names. See Table 5.2 for the assignments to each individual pin on the Beret connectors, and Table 5.5 for the assignments to each individual pin on the STM32. Note that, by default, UARTt is transmit-only or half duplex, and UARTr is receive only. See also Figure 1.7.

pin		STM name(s)	Beret name(s)
B5	PB5	SPI3_MOSI (GPIO)	SPImb_MOSI (I2C_G3)
C7	PC11	SPI3_MISO (GPIO)	SPImb_MISO (Vs1_EN)
A5	PB3	SPI3_SCK (GPIO)	SPImb_SCK (Vreg_FAULT)
H6	PE9	ADC3_IN2	Vmon1
K7	PE13	ADC3_IN3	Vmon2
C5	PB4	TIM3_CH1	E1a
F10	PC7	TIM3_CH2 I2C4_SDA	E1b I2Cd_SDA
F9	PC6	TIM8_CH1 I2C4_SCL	E2a I2Cd_SCL
B10	PA14	TIM8_CH2 I2C4_SMBA	E2b I2Cd_SMBA
A4	PB6	TIM4_CH1 USART1_TX	E3a UARTt_TX
G9	PD13	TIM4_CH2	E3b
B9	PA15	TIM2_CH1	E4a
A8	PD4	TIM2_CH2	E4b
F2	PC0	TIM1_CH1 LPUART1_RX	E5a UARTr_RX
J6	PE11	TIM1_CH2	E5b
J4*	PB2†	TIM5_CH1 (GPIO)	E6a_s (Vmon_A0)
G2*	PA1†	TIM5_CH2 (GPIO)	E6b_s (Vmon_EN)
C3	PE2	TIM20_CH1 (GPIO)	E7a (CAN_SHDN)
B2	PE3	TIM20_CH2 (GPIO)	E7b (CAN_STB)
C9	PA12	USB_DP	USB_DP
C10	PA11	USB_DM	USB_DM
B8	PD0	FDCAN1_RX	CAN_RX
A9	PD1	FDCAN1_TX	CAN_TX
B7	PD2	UART5_RX	UARTa_RX
A10	PC12	UART5_TX	UARTa_TX
H7	PE15	USART3_RX	USARTb_RX
C8	PC10	USART3_TX	USARTb_TX
G7	PD10	USART3_CK	USARTb_CK
D8	PA13	USART3_CTS	USARTb_CTS
J10*	PD12†	USART3_RTS_DE	USARTb_RTS_s
E10	PA8	HRTIM1_CHA1 I2C2_SDA I2S2_MCK	S1 I2Cb_SDA I2S_MCK
D10	PA9	HRTIM1_CHA2 I2C2_SCL I2C3_SMBA	S2 I2Cb_SCL I2Cc_SMBA
C4	PE0	TIM16_CH1	S3
E9	PC9	HRTIM1_CHE2 I2C3_SDA	S4 I2Cc_SDA
E8	PC8	HRTIM1_CHE1 I2C3_SCL	S5 I2Cc_SCL
E3	PF9	TIM15_CH1	S6
E4	PF10	TIM15_CH2	S7
B3	PE1	TIM17_CH1	S8
H9*	PB14†	HRTIM1_CHD1	S9_s
J9*	PB13†	HRTIM1_CHC2	S10_s

Table 5.5: Pinouts of the STM32G474 (VE or VB) on all six Berets. ()* denotes an STM pin that is not natively tolerant to 5V inputs; those marked ()* are also available at a JST or header on the Beret. The 8 digital i/o channels marked ()† are passed through an 8-channel TI TXB0108 level shifter to assure 5V tolerance. The 12 channels marked (GPIO) have different functions on the **Green** and **Blue** Berets, as indicated. (Note: table continues on next page.)

pin		STM name(s)	Beret name(s)
K1*	PA7	OPAMP1_VINP	ADC1
H3*	PA3	OPAMP1_VINM	Vref
H1	PA2	OPAMP1_VOUT	lpf1a
F8*	PD14	OPAMP2_VINP	lpf1c
J3*	PC5	OPAMP2_VINM	Vref
H4*	PB0	OPAMP3_VINP	ADC2
J8*	PB10	OPAMP3_VINM	Vref
K3*	PB1	OPAMP3_VOUT	lpf2a
H10*	PD11	OPAMP4_VINP	lpf2c
K10*	PD8	OPAMP4_VINM	Vref
H2*	PA4	DAC1_OUT1	DAC1
J1*	PA5	DAC1_OUT2	DAC2
B4	PB7	I2C1_SDA	I2Ca_SDA
A3	PB8	I2C1_SCL BOOT0	I2Ca_SCL BOOT0
A2	PB9	IR_OUT	IR_OUT
K9*	PB15†	SPI2_MOSI I2S2_SD	SPIa_MOSI_s I2S_SD_s
D9	PA10	SPI2_MISO	SPIa_MISO
E2	PF1	SPI2_SCK I2S2_CK	SPIa_SCK I2S_SCK
E1	PF0	SPI2_NSS I2S2_WS	SPIa_SS I2S_WS
A7	PD5	USART2_TX	SPIimu_MOSI
A6	PD6	USART2_RX	SPIimu_MISO
B6	PD7	USART2_CK	SPIimu_SCK
G4*	PA0	USART2_CTS	SPIimu_SS
J7	PE14	SPI4_MOSI (GPIO)	SPIdrv_MOSI (pause)
B1	PE5	SPI4_MISO (GPIO)	SPIdrv_MISO (mode)
G6	PE12	SPI4_SCK (GPIO)	SPIdrv_SCK (RS485_RE)
A1	PE4	SPI4_NSS (GPIO)	SPIdrv_SS (RS485_DE)
C2	PE6	RTC_TAMP3	reset
F3	PG10	NRST	reset
D4	PC13	RTC_OUT1	power
D3	VBAT	VBAT	Vcoin
C1	PC14	OSC32_IN	OSC32
F4*	PC1	QSPI1_BK2_IO0	QSPIflash_IO0
F1	PC2	QSPI1_BK2_IO1	QSPIflash_IO1
G1*	PC3	QSPI1_BK2_IO2	QSPIflash_IO2
K2	PC4	QSPI1_BK2_IO3	QSPIflash_IO3
K6	PE10	QSPI1_CLK	QSPIflash_SCK
C6	PD3	QSPI1_BK2_NCS	QSPIflash_SS
J2*	PB11	GPIO	gpio_INT
H8*	PA6	GPIO	mag_INT
K8*	PB12	GPIO (GPIO)	imu_INT2 (amp_OTF_SLEEP)
G5*	PE7†	GPIO	bar_INT_DRDY
G8*	PD9†	GPIO	imu_INT1_DRDY
D1	PC15	GPIO (GPIO)	SPImb_SS (RS485_SEL)
G3	PF2	GPIO	USART_G5
G10	PD15	GPIO	I2C_G2
H5	PE8	GPIO	SPI_G4

Table 5.5: Continued from previous page.

GPIO connections on the STM

pin	i/o	channel name	notes	§
G5†	i	bar_INT_DRDY	OD, AL, PU	5.4
G8†	i	imu_INT1_DRDY	OD, AL, PU	5.4
K8*	i	imu_INT2	OD, AL, PU	5.4
H8*	i	mag_INT	PP, AL	5.4
J2*	i	gpio_INT	OD, AL, PU	5.5
G3	i/o	USART_G5	configurable	5.6.5
G10	i/o	I2C_G2	configurable	5.6.6
H5	i/o	SPI_G4	configurable	5.6.7
D1	i	SPImb_SS	PP, AL	5.9

Connections on the LED Driver

pin	LED name	§
P0	gauge_G1	5.6.8
P1	gauge_G2	5.6.8
P2	gauge_G3	5.6.8
P3	gauge_R	5.6.8
P4	LED_R	5.6.8
P5	LED_Y	5.6.8
P6	LED_G	5.6.8

Connections on the GPIO Expander

pin	i/o	channel name	notes	§
P0_4	o	Vs1_EN	PP, AH, B/G	5.2.4
P0_5	i	Vreg_FAULT	OD, AL, PU, A/3	5.2.4
P1_3	o	drva_SLEEP	PP, AL, B/G	5.3
P1_4	o	drvb_SLEEP	PP, AL, B/G	5.3
P1_5	i	drv_FAULT	OD, AL, PU, A/3	5.3
P1_2	i	mag_DRDY	PP, AL	5.4
P1_6	o	Venc_3.3V	PP, AH	5.6.1
P1_7	o	Venc_5V	PP, AH	5.6.1
P2_0	o	CAN_SHDN	PP, AH	5.6.4
P2_1	o	CAN_STB	PP, AH, B/3	5.6.4
P2_2	o	RS485_SEL	PP, AH	5.6.4
P2_3	o	RS485_RE	PP, AL, B/3	5.6.4
P2_4	o	RS485_DE	PP, AH, A/G	5.6.4
P0_3	i/o	I2C_G3	OD, AL, PU	5.6.6
P2_5	i	pause	OD, AL, PU	5.6.8
P2_6	i	mode	OD, AL, PU	5.6.8
P2_7	i/o	amp_OTF_SLEEP	OD, AL, A/G	5.7
P0_6	o	Vmon_EN	PP, AH	5.7.4
P0_7	o	Vmon_A0	PP	5.7.4
P1_0	o	Vmon_A1	PP	5.7.4
P1_1	o	Vmon_A2	PP	5.7.4
P0_0	i/o	SPImb_G0	configurable	5.9
P0_1	i/o	SPImb_G1	configurable	5.9
P0_2	i/o	SPImb_G2	configurable	5.9

Table 5.6: GPIOs on the **Raspberry**, **Black**, and **White** Berets located on (top/left) the STM and (right) the GPIO expander; note that the **Green** and **Blue** Berets, with fewer GPIOs, do not use a GPIO expander at all (they have all GPIOs moved to the STM, as indicated in Table 5.5). (bottom/left) The LED Driver connections on all six Berets. Inputs with a Pull Up resistor are labelled PU; internal pull up resistors on the STM are 40 kΩ, those on the GPIO expander are 55 kΩ. Outputs: OD = Open Drain, PP = Push/Pull. Logic: AL = Active Low, AH = Active High. {B/G, B/3, A/G, A/3} indicate the channel is associated with a Blue or Amber status LED, connected to Ground or 3.3V; all 9 LEDs tied to the LED driver connect to 3.3V. **TODO**: update pin #s of GPIOs.

5.5 STM32G474 microcontroller features, pinouts, and GPIOs

All six Berets are controlled by a 100-pin STM32G474VE (or, VB) microcontroller[18], which includes a 170 MHz ARM Cortex-M4 core with 512 KB flash and 128 KB SRAM, and several useful features for efficient implementation of difference equations for feedback control of electromechanical systems, such as STM's Adaptive real-time (ART) memory accelerator and a Filter Math Accelerator (FMAC), which implements low-level circular buffers for offloading the computation of FIR and IIR filters from the main ARM core (see §??).

The STM32G474 includes an extensive set of dedicated hardware subsystems for driving busses and peripherals without putting a computational load on the ARM core itself. As shown in Figure 5.9, the Beret makes considerable use of these hardware subsystems on the STM. In fact, after connecting up these many subsystems and connectors on the **Raspberry**, **Black**, and **White** Berets, only 9 pins on the STM were left over to use as dedicated STM GPIOs; 24 additional GPIOs are thus obtained on these Berets using a NXP PCAL6524HEHP GPIO expander[19]. As stated in Note A of Table 5.2, any STM i/o channel available on a Beret JST that is not

[18] Key references for the STM32G474, which users of the Beret will need frequently, are its datasheet and reference manual. Also available from ST is a comprehensive set of training courses (both videos and PDFs) specifically designed for the STM32G4 series.

[19] A change of state of an input to the GPIO expander is indicated by the gpio_INT channel on the STM (see Tables 5.5-5.6).

otherwise being used for its primary purpose can be reconfigured as an auxiliary STM GPIO.

The pinouts and GPIO channels[20] of the STM32G474VE/B (TFBGA100 variants of the STM32G474), the LED Driver, and the GPIO Expander are specified in Tables 5.5-5.6. The {VSS, VSSA} pins {D2, D6, E5, E6, E7, F6, K4} are wired to GND, and the {VDD, VDDA, VREF+} pins {D5, D7, F5, F7, J5, K5} are wired to 3.3V.

5.5.1 Real-time clock (RTC), and scheduled/commanded wakeup from VBAT mode

All six Berets include a modern MEMS oscillator (specifically, an SiTime SIT1532AC-J5-DCC-32.768E, operating at 32.7680 kHz), rather than a quartz crystal resonator with load caps, to generate OSC32 to drive both the STM real-time clock (RTC) as well as the ICM-42688-P IMU (see §5.4).

When in run mode, pressing the reset button on the Beret drives the STM NRST channel on the PG10 pin low, resetting the STM. When in VBAT mode (low-power sleep, powering the STM solely by the VL-1220 coin cell, with the main power to the Beret turned off at the power MOSFET), pressing the reset button drives the RTC_TAMP3 channel on the PE6 pin[21] low. The STM RTC module is used to wake the board from VBAT mode (either if RTC_TAMP3 is driven low, or if scheduled, or if a wake-up signal is received over the LPUART channel), leveraging the STM RTC_OUT1 channel on the PC13 pin, which is wired to the enable pin on the Beret's ideal diode controller, which in turn is wired to the main power MOSFET (see §5.2.3); the 5V and 3.3V Vregs, as well as the STM, quickly power back up automatically when this power MOSFET is turned back on. [Check!]

5.5.2 Customization with Quad-SPI Flash

The **full size** Berets are also preconfigured with a pinout of the STM QuadSPI module QSPI1_BK2 for easy addition of a standard-size (6 mm x 8 mm, 8-WSON) fast, low-cost, high-capacity 4 MB to 512 MB flash IC for extending the (NAND or NOR) flash memory capacity of the system, for those applications that need it.

[20] A GPIO may be either active high or active low, and either push-pull or open drain. **Active high** means the channel is "active" or "on" in the Logical 1 state (3.3V on the Beret), and "inactive" or "off" in the Logical 0 state (GND); **active low** means the opposite. **Push-Pull** means the connected device either drives the channel low (through an NPN BJT or n-channel MOSFET to GND), or drives the channel high (through a PNP BJT or p-channel MOSFET to 3.3V). **Open Drain** (a.k.a. **Open Collector**), in contrast, means the connected device drives or "asserts" the channel to the low state (through an NPN BJT or n-channel MOSFET to GND), but the channel is left floating by the connected device instead of driving it to the high state, thus requiring a **pull-up resistor** (connecting the channel to 3.3V via a resistor, often internal to the MCU) to achieve a definitive boolean state. [A **pull-down resistor** (connecting to GND) is occasionally needed in certain analogous situations, in which a device (like a button) might assert a channel to the high state, but otherwise leaves it floating.] An advantage of the Open Drain setting is that several devices can be tied to a single GPIO channel on the MCU; this channel then functions as a **wired OR** device for active low logic (or, as a **wired AND** device for active high logic).

[21] The PE6 pin and the PG10 pin, operated as inputs, are wired together; when in run mode, PE6 is ignored, and when in VBAT mode, PG10 is ignored.

5.6 Connectivity and i/o

5.6.1 Quadrature encoder counters and connectors

As indicated in Figure 5.9 and Tables 5.4-5.5, the encoder channels E5, E2, and E7 on the Berets connect to pairs of outputs on the STM advanced control timers TIM1, TIM8, and TIM20, and the encoder channels E4, E1, E3, and E6 on the Beret connect to pairs of outputs on the STM general purpose timers TIM2, TIM3, TIM4, and TIM5. All 7 of these hardware timers are nominally operated in the quadrature encoder counting mode, with up/down counting, without loading the main ARM core. The encoder connectors on the Beret also include connections to power. As discussed in §5.2.9, the (3.3V or 5V) power provided to the E3-4 and E5 connectors may be changed via backside solder jumpers, and the (3.3V, 5V, Vs1, or off) power provided to the E1-2 and E6-7 connectors is selected via a multiplexer using the Venc_3.3V and Venc_5V Beret GPIOs.

Further, the E1-2, E3-4, E5, and E6-7 connectors are wired, primarily, for standard two-channel (AB) quadrature encoders. However, if needed, the STM32 ETR pins for 5 of these 7 encoder channels are readily available on other various other available pins the Berets, specifically:

- PE7 is connected to I2C_G2 on the I2C Header, and can act as ETR for TIM1 (E5).
- PB12 is connected to SPI_G4 on the SPI Header, and can act as ETR for TIM5 (E6).
- PA8 is connected to S1 on Signal Header A, and can act as ETR for TIM4 (E3).
- PE0 is connected to I2C_G2 on the I2C header, and can act as ETR for TIM20 (E7).
- PD2 is connected to UARTa_RX on the RS485/UART JST, and can act as ETR for TIM3 (E1).

This facilitates the use of up to five three-channel (ABZ) encoders without requiring a Beret Shield.

As discussed in section 3.24 of the STM32G474 datasheet, these timers are quite powerful, and thus the convenient E1-2, E3-4, E5, and E6-7 connectors may be used for a host of alternative functions, such as the generation of PWMs for driving up to 14 additional servos and ESCs (again, without loading the ARM core itself), which may be useful if the 10 discrete servo/ESC connectors discussed in §5.6.2 provide insufficient connectivity for a given application. Other functionality also available on the encoder connectors includes:
- I2C4 (including SMBA), available on E1-2 (see §5.6.6),
- a half-duplex UART channel (USART1_TX), available on E3-4 (see §5.6.5), and
- the LPUART1_RX channel (in receive only mode, as used by DSM receivers), available on E5 (see §5.6.5).

Warning: no specific Electrostatic Discharge (ESD) protection is provided on the encoder connectors.

5.6.2 PWM-based servo and ESC controllers and the Signal Headers

As indicated in Figure 5.9 and Table 5.5, signals {S1, S2, S4, S5, S9, S10} connect to the STM high-resolution timer HRTIM1, and signals {S6, S7, S3, S8} connect to the STM general purpose timers TIM15, TIM16, and TIM17. Signal Headers A and B expose signals S1 through S10, along with power and GND, in an industry-standard manner (see Note H of Table 5.2). Vs1 is provided on Signal Header B, and the user may select between Vs1 and Vin on Signal Header A (see §5.2.9). A blue status LED near the Vs1 voltage regulator illuminates whenever power (Vs1) to the signal headers is enabled (see §5.6.8).

Again, these timers are quite powerful, and thus signals S1 through S10 on Signal Headers A and B may be used for a variety of alternative functions, such as adding a few unidirectional encoder counters (using TIM15, TIM16, and TIM17, with up-counting only – and, again, without loading the ARM core itself), which may be useful if the 7 quadrature encoder counters (with up/down counting) discussed in §5.6.1 are insufficient for a given application. The following alternative functionality is also available on the signal headers:
- I2C2 is available on {S1,S2} (see §5.6.6), and
- I2C3 is available on {S4,S5}, with SMBA available on S2 (see §5.6.6).

Note also that signals S1 through S5, and thus the high-resolution timer to which they connect, are readily available for creative use on Extended Beret Shields, as discussed in §5.8.

ESD protection is provided on the S1 - S10 signal lines by two TI TPD6E004s.

5.6.3 Encoding for IR communication

The output of the STM IR communication module may be routed (as IR_OUT) to pin 9 of the SPI Header. When the IR communication module is enabled, S3 and S8 on Signal Headers A and B are converted into GPIOs.

5.6.4 CAN FD and RS485 transcievers and connectors

On the **CAN/RS485** Berets, the STM FD-CAN controller of the STM32G4 is paired with a (3.3V) TI TCAN334G **CAN-FD transceiver**, to generate the differential pair of signals, {CANH, CANL}, required for CAN communication, supporting the CAN FD (CAN with flexible data-rate) protocol at up to 5 Mbps. These two signals are made available on the CAN JST connector (see Table 5.2). Note that the STB and SHDN pins of the TCAN334G are wired to the Beret CAN_STB and CAN_SHDN GPIO channels (see §5.5), to put the transceiver in low-power standby and shutdown states when not in use (see the device datasheet for details).

On the **CAN/RS485** Berets, the STM's UART5 channel may be used to drive a (3.3V or 5V selectable) TI THVD1452 **RS485 transceiver**, with built-in idle bus failsafe, to generate the two differential pairs of signals, {Y,Z,A,B}, required for full-duplex RS485 communication, supporting the RS485 protocol at up to 50 Mbps. This is accomplished by setting the Beret's RS485_SEL GPIO to 1, and using the GPIO channels RS485_RE and RS485_DE to enable its receiver and driver (i.e., transmitter) as necessary. Note that this transceiver presents a 1/8 Unit Load, allowing up to 256 receivers on a single bus.

On the backside of these Berets, the user may install (in the footprints provided) two 0804 resistors and (optionally) a 0804 capacitor for termination of the CAN bus, and a single resistor (on each differential pair) for termination of the (full-duplex) RS485 bus, as discussed in the TI AN-1057 report.

Note that the CAN and RS485 transceivers provide certain levels of ESD protection, as specified in the corresponding device datasheets. For both, if necessary, additional TVS diodes may be placed on the bulkhead of the environmental housing (see §4.5.4.3), where the rugged field-serviceable connectors (see §4.5.4.2) attach to the short jumper cables that brings the differential CAN and/or RS485 signal pairs onto the Beret. This physical separation helps to prevent voltage transients, ESD, and noise from propagating onto the Beret itself.

5.6.5 USART, UART, and LPUART modules and connectors

Four independent STM USART and UART modules, and the STM LPUART module, are used on the Beret:
- the STM USART3 module (RX, TX, CK, CTS/NSS, & RTS/DE) is fully broken out on the USART JST,
- the STM UART5 module (RX & TX) is broken out on the RS485 (UARTa) JST if RS485_SEL=0 (see also 5.6.4),
- the STM USART1 module (TX only[22], for use in half duplex mode) is available (as E3a) on the E3-4 JST,
- the STM LPUART1 module (RX only[23], for use in receive mode only) is available (as E5a) on the E5 JST, and
- the STM USART2 module (in SPI mode) gives the Beret a dedicated high-speed connection to the IMU.

[22] Via a backside solder jumper, the 4th pin on the E3-4 JST can be switched over to STM pin B3, which can be set in software to function as the USART1_RX input. If this is done, E3-4 can then function as a full duplex Recon UART-T connector with two GPIOs. Note that doing this makes the S8 channel on Signal Header B unusable for other purposes.

[23] Via a backside solder jumper, the 4th pin on the E5 JST can be switched over to STM pin H8, which can be set in software to function as the LPUART1_TX output. If this is done, E5 can then function as a full duplex Recon UART-R connector. Note that doing this makes the mag_INT channel (to the STM and the MB Header) unusable as a magnetometer interrupt. Note also that, though H8 is not 5V tolerant, it is set as an output when E5 is converted to a UART-R connector in this manner, so this connector can still be connected to 5V TTL UART devices.

The four JSTs mentioned above are useful for connecting (primarily point-to-point, with one device per channel[24]) a variety of additional sensors and other devices to the Beret, such as GPS/GNSS units. Extra GPIOs, if needed by a particular device, can be picked up from unused pins on other JSTs, or from the I2C and SPI headers (see Note A of Table 5.2). The first three pins (power, GND, signal) of E5 is suitable for direct connection to 2.4Ghz DSMX/DSM2 radio receivers, such as the Spektrum SPM4648 and OrangeRx R110x.

Supported modalities on the STM32's flexible USART channel (available on the Beret's USART JST) include:
- full duplex (RX-to-TX and TX-to-RX) and single-wire half duplex (TX-to-TX) UART modes,
- UART (asynchronous, without CK) and USART (synchronous, with CK) modes,
- multiprocessor communication [e.g., direct from the STM on one Beret to the STM(s) on other Beret(s)],
- SPI master (TX-to-MOSI, RX-to-MISO) and SPI slave (TX-to-MISO, RX-to-MOSI, slave select on NSS) modes,
- Smartcard ISO7816 communication,
- IrDA serial infrared communication, and
- RS232 and RS485 hardware flow control.

Warning: no specific ESD protection is provided on the USART or UART JSTs, or other pins where UART functionality is available, as specified above.

5.6.6 I2C modules and the I2C Header

Four independent STM I2C modules, at comm speeds up to 1 MHz, are broken out and used on the Beret:
- the STM I2C1 module (SDA & SCL) is broken out on the I2C Header,
- the STM I2C2 module (SDA & SCL) is available (as S1 & S2) on Signal Header A,
- the STM I2C3 module (SDA, SCL, & SMBA) is available (as S4, S5, & S2) on Signal Header A, and
- the STM I2C4 module (SDA, SCL, & SMBA) is available on the E1-2 JST.

Note that the I2C1 channel (on which the STM operates as master) also connects on the Beret to:
- three TI TPL0102-100 dual digital pots[25]:
  - POT1, at I2C address 101 0101b (0x55h), POT2, at I2C address 101 0110b (0x56h) [see §5.7.2], and
  - POT3, at I2C address 101 0111b (0x57h) [see §5.2.4 and §5.7.1],
- the GPIO expander, at I2C address[26] 010 0010b (0x44h for write, 0x45h for read) [see §5.5],
- the LED driver, at I2C address 100 0101b (0x8Ah for write, 0x8Bh for read) [see §5.6.8],
- the barometer, at I2C address[27] 101 1100b (0xB8h for write, 0xB9h for read) [see §5.4], and
- the magnetometer, at I2C address[28] 001 1100b (0x38h for write, 0x39h for read) [see §5.4].

Warning: when connecting other common I2C devices on the I2C Header, care must be taken to not conflict with the eleven underlined I2C hex addresses listed above, which are already used. Due to this exisiting traffic on the I2C1 channel, when attaching additional I2C devices, it is recommended that the user instead consider the use of the I2C2, I2C3, and I2C4 modules, if the various pins for them are available, as discussed above.

Warning: no specific ESD protection is provided on the I2C Header, or other pins where I2C functionality is available, as specified above.

[24] Operating the USART3 module (wired to the USART JST) in SPI mode allows it to connect to multiple devices, with one Beret GPIO operating as a dedicated CS for each connected device.

[25] POT1 is wired with {A2,A1,A0}={1,0,1}, POT2 is wired with {A2,A1,A0}={1,1,0}, POT3 is wired with {A2,A1,A0}={1,1,1}.

[26] The GPIO expander is wired with {A2,A1,A0}={0,0,1}.

[27] The barometer is wired with SA0=0.

[28] The magnetometer is wired with SA1=0.

5.6.7 SPI modules and the SPI Header

Three[29] independent STM SPI modules are broken out and used on the Berets:
- the STM SPI2/I2S2 module (MOSI, MISO, SCK, NSS in SPI mode, or SD, WS/LRCLK, SCK/BCLK in I2S mode[30]) is broken out on the SPI Header, providing unencumbered SPI (or I2S) functionality to the user,
- the STM SPI3 module gives the Beret a dedicated high-speed connection to the SPI channel on the MB (except, of course, on the **Green** and **Blue** Berets), using (on the STM) GPIOs connected to MB's SS channels (see Table 5.6) for software slave select, and
- the STM SPI4 module, operating with a single SS line in daisy-chain mode, gives the Beret a dedicated high-speed connection to DRVa and DRVb (see §5.3).

The USART JST (i.e., USART3, see §5.6.5), operating in SPI mode, provides further unencumbered SPI functionality to the user if needed.

NSS on the SPI header (i.e., SPI2) provides a hardware slave select line when the Beret is used as an SPI slave on this channel. When the Beret is used as the SPI master on this channel, on the other hand, multiple SPI slaves can be attached using STM GPIOs as separate slave select pins, recalling again that extra GPIOs, where needed, can be picked up from any unused pins on the SPI and I2C headers, or over on the JSTs.

Warning: no specific ESD protection is provided on the SPI or MB Headers.

5.6.8 LEDs, Buttons, and Displays

LEDs. A TI TCA6507 LED driver is used to control the main LED circuits on the Beret, including three user LEDs (LED_R, LED_Y, LED_G, forming a "stoplight" of sorts) and three bicolor LEDs (controlled via the channels gauge_G1, gauge_G2, gauge_G3, gauge_R[31]), forming a "power gauge" next to the XT30 input, which are programmed to be illuminated (if a battery is detected at the Balance connector) based on the charge of the individual cell operating with the lowest charge as follows:

● ● ●	Three solid green	85% to 100% charge
● ● ○	Two solid + one flashing green	70% to 85% charge
● ●	Two solid green	55% to 70% charge
● ○	One solid + one flashing green	40% to 55% charge
●	One solid green	25% to 40% charge
○	One flashing green	10% to 25% charge
● ● ●	Three solid red	1% to 10% charge, user should shutdown immediately
○ ○ ○	Three (quickly) flashing red	Vmin reached, automatic board shutdown imminent.

The voltage levels for these indicator transitions come pre-programmed for LiPo batteries, with Vmin = 3.2V and Vmax = 4.1V, but may be adjusted further by the user. If no battery connection is detected on the Balance connector[32], the center green LED is illuminated simply to indicate that the board is powered up and running normally (most likely from a wall adapter). Note that alternate red/green patterns slowly flashing in the power gauge, using a simple binary representation in the greens, may be used to indicate eight distinct user-programmable error codes.

In addition, there are blue and amber status LEDs on the Berets, each located in the respective quadrant of Figs 5.1-5.6, and attached directly to the corresponding GPIO channel (see Table 5.6):

[29] In addition (see §5.6.5), recall that the STM USART2 module, operating in SPI mode, gives the Beret a dedicated high-speed connection to the IMU.

[30] If the I2S MCK channel is needed, it is available on pin S1 of Signal Header A when using Extended Beret Shields (see §5.8).

[31] The three green gauge LEDs are controlled independently, whereas the three red gauge LEDs are controlled by a single channel.

[32] Warning: this system is designed to be used with a balance connector whenever a battery is being used, and will not actively monitor the state of charge of the battery unless the balance connector is plugged in.

- the blue Vs1 LED illuminates whenever the Vs1 Vreg circuit is enabled,
- the blue DRVa LED illuminates whenever the DRVa motor driver is enabled,
- the blue DRVb LED illuminates whenever the DRVb motor driver is enabled,
- the blue CAN LED illuminates when the CAN transceiver is in run mode,
- the blue RS485_RE LED illuminates when the RS485 receiver is enabled,
- the amber RS485_DE LED illuminates whenever the RS485 driver (transmitter) is enabled,
- the amber Vreg_fault LED illuminates whenever any of the Vreg ICs signals a fault,
- the amber drv_FAULT LED illuminates whenever either of the DRVs signals a fault, and
- the amber opamps LED illuminates when the Beret opamps are enabled and no fault is triggered[33].

The red, yellow, amber, and bicolor LEDs on the Berets, including those in the bicolor LEDs, operate at about 2V and 6 mA, and the blue and green LEDs on the Berets operate at about 2.9V and 5 mA. As the LED circuits are all powered with 3.3V, (3.3V-2V)/6 mA≈ 220 Ω resistor arrays are used to regulate the current to the red, amber, yellow, and bicolor LEDs, and (3.3V-2.9V)/5 mA≈ 82 Ω resistor arrays are used to regulate the current to the blue and green LEDs.

The several red and amber LEDs on the Beret are manufactured using AlInGaP (Aluminium, Indium, Gallium and Phosphorous) technology, and the several green and blue LEDs on the Beret are manufactured using InGaN (Indium, Gallium and Nitrogen) technology. These modern choices provide maximum brightness while drawing the minimum amount of current.

Buttons. Two small user-programmable buttons (with white actuators) are included as inputs, with pull-up resistors on the GPIO expander; pressing these buttons connects these resistors to GND. The buttons can be used for any function in software, but are preassigned the names "pause" and "mode", which is appropriate for typical use cases. A third button (with a black actuator), named "reset", is used to reset the STM when in run mode, or to wake the entire board from low-power sleep mode when the board is powered down (see §5.5.1).

Displays. A Beret Shield (see §5.8) implementing COTS I2C 0.96" OLED Display module is planned. An eInk Beret Shield would also be nice. Size?

5.6.9 USB Micro-B connector and Device Firmware Upgrades

The STM USB module, available on pins USB_DP and USB_DM, is wired directly to a standard USB Micro B connector on all six Berets. This USB connector may be used for ordinary programming, in addition to performing Device Firmware Upgrades (DFUs) using the STM DeFuse software, as discussed further here, in conjunction with the BOOT0 pin available on pin 7 of the I2C header (see Table 5.2).

The USB connector can power a Beret for the purpose of STM programming only (see §5.2.2); the USB connector does not provide enough current to power motors or a connected MB, which should not be attempted.

ESD protection is provided on the USB data lines by a TI TPD6E004.

[33]The amp_OTF_SLEEP i/o channel, when set as an output and driven low by the GPIO expander, puts the ALM2402-Q1 opamps (§5.7.1) and TLV9002S opamps (§5.7.2) into a low-power sleep mode. When set as an input on the GPIO expander, it is left floating by the opamps during normal operation, and is pulled up to logic high by an external 2.5 kΩ resistor to 5V while simultaneously illuminating a corresponding 2V amber opamps LED tied to ground, unless/until a fault is triggered, then it is driven low.

5.7 Analog subsystem

The **Raspberry**, **Black**, **White**, **Green**, and **Blue** Berets include a 0V – 3.3V analog subsystem with:
- two high-power (400 mA) DAC channels,
- (V+, V-, Vo) pinouts for a spare opamp that may be configured (on a Beret Shield) by the user, and
- two ADC channels with:
  - adjustable gain, x1 to x4096, and
  - adjustable second-order filtering, with tunable cutoff frequency $\omega_c = 2\pi f_c$ and damping $\zeta = 1/(2Q)$.

A amber status LED (see §5.6.8) next to the Analog Header illuminates whenever the analog subsystem, and the opamps that drive it, are enabled by the STM.

5.7.1 Generation of Vs2 = 1.2V to 2.1V, and two high-power DAC outputs

Figure 5.10a illustrates how the reference voltage V2 = 1.2V to 2.1V is generated, with $R_a = R_b = 133\,\text{k}\Omega$ and R_v given by half of POT3, a TI TPL0102-100 dual digital pot (100 kΩ) operating in voltage divider mode.

Two TI ALM2402-Q1 dual high-power opamps are included on the full size Berets, each with a pair of IGBTs arranged as class AB amplifiers with zero crossover distortion, negligible voltage offset, and robust current limiting. The [open-drain, active low] amp_OTF_sleep i/o channel, with an external 2.5 kΩ pull-up resistor to 5V (CHECK for Blue) as well as a 2V amber diode to GND, is connected to the GPIO expander via a 220 Ω resistor (see Table 5.6). When this channel is set on the GPIO expander as an input, it is used to monitor for an Over Temperature Flag on these opamps; when set as an output and driven low, this channel puts both ALM2402-Q1s into a low-power sleep mode.

Figure 5.10b shows how three of these high-power opamps are used to buffer $V \in \{\text{V2, DAC1, DAC2}\}$, thereby generating the buffered outputs $\bar{V} \in \{\text{Vs2, DAC1buf, DAC2buf}\}$ made available on the Analog Header, each of which is capable of sourcing or sinking 400mA. [Note that the 3 terminals of the fourth high-power opamp are provided directly on the Analog Header, as discussed further in §5.7.3.] When set near 1.65V, Vs2 is useful as a bipolar offset for the analog subsystem.

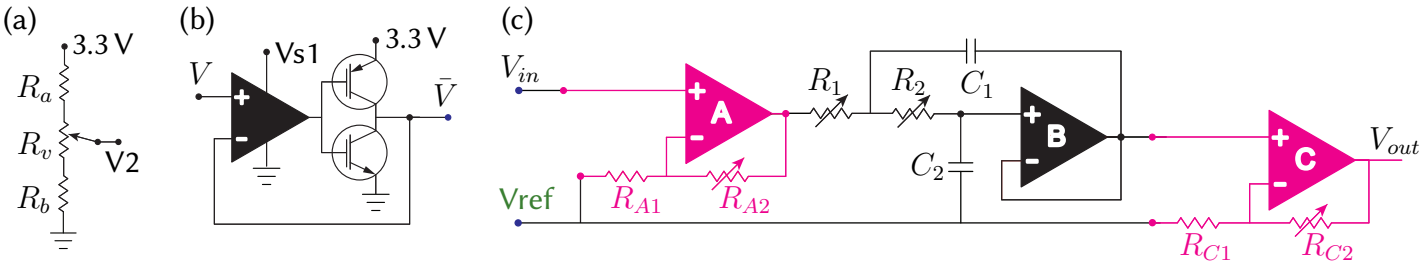


Figure 5.10: Circuits for (a) generation of V2 = 1.2 V to 2.1 V adjustable, (b) buffering of $V \in \{\text{V2, DAC1, DAC2}\}$ (each buffered output, $\bar{V} \in \{\text{Vs2, DAC1buf, DAC2buf}\}$, is capable of sourcing or sinking 400mA), and (c) amplification and filtering of the $V_{in} \in \{\text{ADC1, ADC2}\}$ inputs, each of which are compared to the reference voltage Vref $\in \{\text{GND, Vs2, ADC3}\}$, with outputs V_{out} routed to ADC2_IN3 and ADC4_IN3 channels internally on the STM. Note that magenta components/traces are within the STM, and black traces/components/signals are on the Beret. See Table 5.2 for how these various input, output, and power pins appear on the Analog Header.

5.7.2 Tunable filtering/gain of two unipolar, bipolar, or differential ADC inputs

Figure 5.10c illustrates the circuit used (leveraging 4 of the internal opamps on the STM, together with a TI TLV9002S dual low-cost opamp) to amplify and filter the $V_{in} \in \{\text{ADC1, ADC2}\}$ inputs on the Analog Header, before their corresponding amplified/filtered outputs V_{out} are routed to internal ADC units on the STM.

There are three natural choices for selecting the bias voltage Vref about which these ADC inputs are compared and amplified (this choice is made, by the user, by the wiring on the Beret Shield where the analog circuit is developed; for further discussion, see §5.8):

- for **unipolar** analog signals, the user should wire Vref to GND (again, on the Beret Shield);
- for **bipolar** analog signals, wire Vref to Vs2 (as a Bipolar Offset, tunable in the vicinity of 1.65V);
- for **differential comparison** of analog signals, wire Vref to a third (user-provided) analog signal ADC3.

Note that the ADC1 and ADC2 inputs on the Analog Header lead directly (without any intervening resistors or capacitors) to opamp input terminals in Figure 5.10c, so these filters perform predictably even for "weak" analog sources with low output impedance. Note further that:

- For $V_{in} =$ ADC1, **A** is OPAMP1, **B** is half of the TLV9002S, **C** is OPAMP2, and $V_{out} =$ ADC2_IN3;
- For $V_{in} =$ ADC2, **A** is OPAMP3, **B** is half of the TLV9002S, **C** is OPAMP4, and $V_{out} =$ ADC4_IN3.

In the (non-inverting) "PGA mode" shown for both opamps **A** and **C**, the internal resistors $\{R_{A1},\ R_{A2},\ R_{C1}\ R_{C2}\}$ can be selected (in software) to achieve amplification ratios of x2 to x64; note that $\{R_{A2},\ R_{C2}\}$ can also be bypassed, and the corresponding connections to Vref (through $\{R_{A1},\ R_{C1}\}$) cut (in software, by selecting "follower mode", as shown here for opamp B) in order to achieve amplification ratios of x1 on all three opamps. Thus, the overall low-frequency amplification of this circuit can be varied, in software, from x1 to x4096.

The pairs of resistors $\{R_1, R_2\}$ and capacitors $\{C_1, C_2\}$ in Figure 5.10c, looping around opamp B, form a **Sallen-Key second-order low-pass filter** (LPF), leading (see, e.g., here or here) to the transfer function

$$\frac{V_{out}(s)}{V_{in}(s) - \text{Vref}} = A_A\, A_C\, \frac{\omega_c^2}{s^2 + 2\,\zeta\,\omega_c\,s + \omega_c^2}$$

where $A_A = 1 + R_{A2}/R_{A1}$, $A_C = 1 + R_{C2}/R_{C1}$, and the cutoff frequency $\omega_c = 2\pi f_c$ and damping $\zeta = 1/(2Q)$ of the second-order low-pass filter are $\omega_c = 1/\sqrt{R_1\,C_1\,R_2\,C_2}$ rad/s and $\zeta = C_2\,(R_1 + R_2)\,\omega_c/2$.

A target value of damping to use in such a filter is $\zeta \approx 0.7$. Common capacitor values of $C_1 = 68$ nF and $C_2 = 33$ nF have been selected for the Beret, together with POT1 and POT2, two TI TPL0102-100 dual digital potentiometers that are adjustable electronically (over I2C) from 0 to 100 kΩ in 256 increments, for R_1 and R_2. Setting $R_1 = R_2$ and adjusting both (together) over a range from 100 kΩ down to 1 kΩ results in $\zeta \approx 0.7$, and

f_c ranging from 34 Hz to 3400 Hz, as appropriate for many small electromechanical systems; adjusting R_1 and R_2 separately also allows the damping ratio ζ to be tuned[34].

Warning: attempting the following is quite difficult, voids any sort of manufacturer's warranty expressed or implied, and should only be attempted by advanced users willing to possibly fry their board. Using a very fine-tipped soldering iron, advanced users might choose to attempt to replace C_1 and C_2 in these circuits with capacitors 1 or 2 orders of magnitude larger or smaller than those selected here in order to achieve different frequency ranges for the low-pass filter on the ADC channels. Alternatively, and more simply, removing C_2 altogether (i.e., taking $C_2 \rightarrow 0$ in the transfer function listed above and simplifying) reduces the circuit to a first-order filter with transfer function

$$\frac{V_{out}(s)}{V_{in}(s) - \text{Vref}} = A_A \, A_C \, \frac{\omega_1}{s + \omega_1}$$

with cutoff frequency $\omega_1 = 1/[C_1(R_1 + R_2)]$; setting $R_1 = R_2$ and adjusting both over the range from 100 kΩ to 1 kΩ results in $f_1 = \omega_1/(2\pi)$ ranging from 12 Hz to 1200 Hz. Subsequently replacing C_1 with a capacitor 1 or 2 orders of magnitude larger or smaller can again be done to achieve different frequency ranges. Finally, removing both C_1 and C_2 altogether (and setting $R_1 = R_2 = 1$ kΩ) removes the low-pass filter entirely, allowing the user to take responsibility for any necessary analog low-pass filtering (on a Beret Shield) of an ADC input before it is sampled by the STM's ADC unit. **Warning**: please re-read the warning at the beginning of this paragraph. Ok, you've been warned, good luck. (Actually, if you have a specific/substantial need for low-pass filtering over a different frequency range, it's probably better to contact us and have us make a variant of the board with different capacitors installed...)

5.7.3 Analog Header and user-developed analog filters

As shown in Table 5.2, the Analog Header provides the following nine analog (0V – 3.3V) signals:

- two buffered (up to +/- 400 mA) outputs DACbuf1, DACbuf2,
- the positive and negative inputs, and (0V – 3.3V, ±400 mA) output, of a power opamp, {V+,V-,Vo},
- the voltage Vref (input to the Beret) about which the ADCs are compared and amplified (see §5.7.2),
- two inputs ADC1, ADC2, and
- the low-pass-filtered ADC2filt [i.e., the analog output of OPAMP B (see Figure 5.10c) in the ADC2 filter].

Again, low-pass filtering with tunable gain (x1 to x4096), tunable cutoff frequency (f_c ranging from 34 Hz to 3400 Hz) and tunable damping (nominally, $\zeta \approx 0.7$) is applied to the ADC inputs before sampling by the STM.

Note also that GND, 3.3V, and Vs2 are readily available on the nearby SPI and I2C Headers.

The functionality describe above facilitates the connection of a number of analog sensors and actuators, the experimental determination of MIMO Bode Plots of continuous-time electro-mechanical systems, as well as the easy implementation of other user-developed analog filters on Beret Shields (see, e.g., TI's Op Amps For Everyone for several filter ideas).

Warning: no specific ESD protection is provided on the pins of the Analog Header.

[34] By sinusoidally exciting a DAC channel over a range of frequencies, and routing the output directly to an ADC channel, the Bode plot of the corresponding Sallen-Key second-order low-pass filter may be measured directly, and the values of R_1 and R_2 subsequently tuned in software to achieve the desired filter response.

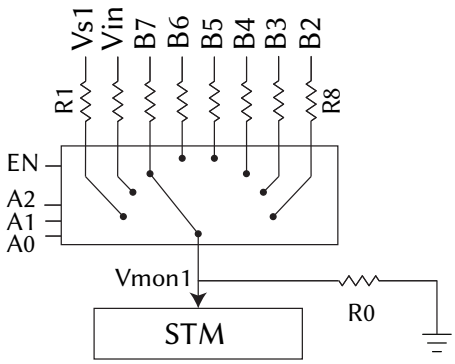


Figure 5.11: Vmon1 circuit with a TI MUX508 8:1 multiplexer, used to monitor {Vs1, Vin, B7, B6, B5, B4, B3, B2} on the **full size** Berets, with {R1, R2, R3, R4, R5, R6, R7, R8} = {27.4, 76.8, 73.2, 59.0, 45.3, 31.6, 17.4, 3.74} kΩ and R0 = 10 kΩ [standard 1% resistor values in the E96 series; see Table 9.5]. Note that B1=GND. When EN=0, Vmon1=GND, and the current through all voltage dividers is zero. Note that Vmon2 is connected directly to Vs2, with no voltage divider. A TI TMUX6219 SPDT switch is used to monitor {Vs1, Vin} only on the **half size** Beret, using effectively same circuit and dropping {A2, A1}, with the same values of {R1, R2} and R0. The nominal scale factors relating the measured values at Vmon1 to the quantities of interest is laid out in Table 5.7.

control inputs	000	001	010	011	100	101	110	111
input voltage	Vs1	Vin	B7	B6	B5	B4	B3	B2
nominal scale factor	3.74×	8.68×	8.32×	6.90×	5.53×	4.16×	2.74×	1.374×

Table 5.7: Quantities measured by the circuit in Figure 5.11 for different inputs {A2, A1, A0} on the **full size** Berets, and their nominal associated scale factors. Only the first two columns apply on the **half size** Berets.

5.7.4 Voltage monitoring of Vin, Vs1, Vs2, and the individual battery cells

Berets periodically monitor Vin, Vs1, and Vs2 when running. Noting warnings ii and iii of §5.2.1, **full size** Berets also periodically monitor (using the custom JST-XH balance connector described in Note F of Table 5.2) the differential voltage over the individual battery cells of 2S – 6S batteries via their stock JST-XH balance connectors, to ensure that no individual cell drops below the minimum allowed voltage, Vmin, while the Beret is operating. Vmin is adjustable in software; a value in the range of 3.1V to 3.2V is appropriate for LiPos.

Voltage monitoring of Vin, Vs1, and the battery cells is done using the multiplexed voltage divider circuit in Figure 5.11, with EN and {A2, A1, A0} tied to the GPIO expander (see Table 5.6) to enable the system and select the active input. The STM's ADC3_IN3 channel, denoted Vmon1 on the Berets, periodically monitors the output of the 8:1 multiplexer in this circuit on the **full size** Berets, or of the SPDT switch on the **half size** Beret, while the STM's ADC3_IN3 channel, denoted Vmon2, periodically monitors Vs2 directly. As suggested by the analysis of TI Report SLVA450A, a target current of about 0.3 mA was chosen for the voltage dividers. For a target maximum value for Vmon1 of about 3.2V, this gives R0=10 kΩ. Given the relationship between the voltages at the top, middle, and bottom of a voltage divider (see Example 9.1), the resistor values Ri, for i=1 to 8, were then selected (see code) so that Vmon1 was about 3.2V for the maximum values of each input Vi.

1% tolerance resistors (E96 series, 0.01 W rating is sufficient) are implemented in the Vmon1 circuit on the Berets. This gives some inaccuracy in the calculated voltages, as quantified by equation (13) of SLVA450A. The nominal scale factors (to determine each input voltage Vi from the measured value of Vmon1) listed in Table 5.7 should thus be **calibrated** using otherwise-measured voltages {Vs1, Vin, B7, B6, B5, B4, B3, B2} in order to eliminate these inaccuracies, as the relationships between the Vi and Vmon1 are accurately linear. Note that higher-precision resistors (with the same nominal values, in the E192 series) could be used in this circuit, but doing such is expensive and unnecessary if the scale factors are to be calibrated after manufacturing.

5.8 Beret Shields

On all six Berets, convenient and stackable **Small** (1.3" x 0.9") and **Extended** (1.3" x 1.1", 1.3" x 1.3", or larger) **Beret Shields** (that is, small daughterboards) may be attached, in a manner similar to Arduino Shields, atop

(i) the (1x9) SPI/I2S Header,
(ii) the (1x9) I2C Header,
(iii) the (1x9) Analog Header (if present), and
(iv) (on 1.3" x 1.1" or larger Beret Shields) the first row (signals S1 - S5) of Signal Header A, or
(on 1.3" x 1.3" or larger Beret Shields) all three rows (S1 - S5, Vs1/Vin, and GND) of Signal Header A,

thus enabling the user to build up quickly, and attach securely, any extra analog or digital circuitry that might be needed in a given application. The pins on these headers are aligned on a 0.1" grid, facilitating the use of:

Prototyping Beret Shields, with an array of predrilled holes on a 0.1" grid, which may be
- **plated**, for rapid development and testing of simple circuit designs, or
- **unplated**, providing a sturdy mechanical backing for COTS PCBs.

Prefabricated Beret Shields implementing commonly needed additional components, such as:
- 2x custom 128-pin solderless breadboards + 2x more 1x9 Headers (USART and GPIO) + LEDs/buttons,
- a 0.96" OLED display + 2x buttons,
- 2x more BDC motor drivers (24 half bridges), wired as described in §5.3,
- 6x sensorless BLDC motor drivers with integrated 28V/3A MOSFETs and JST-PA connectors,
- 2x continuous-time (CT, i.e., analog) notch filters (to eliminate the tonal "buzz" in 2 input signals),
- 2x CT lead/lag/PID feedback control circuits with digitally-adjustable poles, zeros, and gain,
- a u-blox zed-f9p dGPS/GNSS unit,
- a wifi/bluetooth module,
- an array of additional buttons, LEDs, and Recon UART and I2C connectors,
- arrays of connectors supporting other standards (PMOD, Grove, STEMMA, Qwiic, etc).

Custom Beret Shields compactly implementing your choice of components, layout, and connectivity.

Examples are shown in Figure 5.12. All three types of Beret Shields are low cost and easy to use. In particular:

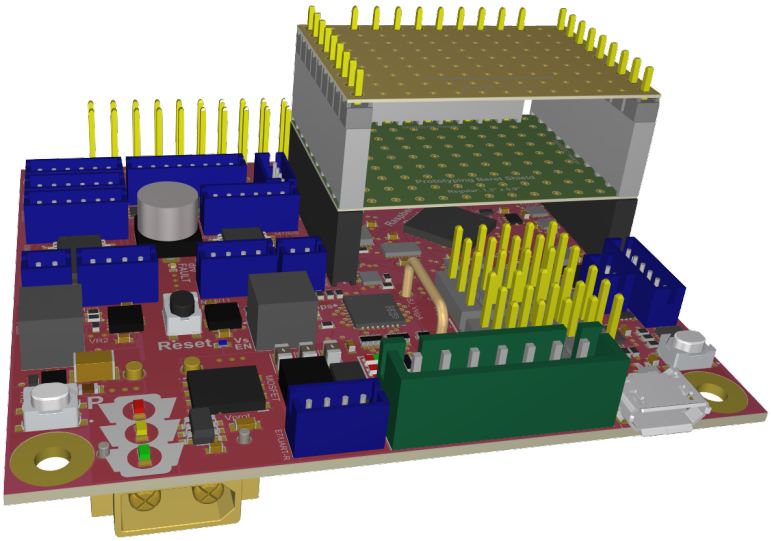
- Prefabricated Beret Shields provide a fast and flexible way to extend the capability of the Beret ecosystem with a variety of commonly-needed additional components via open hardware designs.
- Custom Beret Shields facilitate the dense and secure arrangements of electronic components of the user's choosing for long-term use, and may easily be designed using free software, leveraging directly the open hardware circuit designs of the Prefabricated Beret Shields, and may be fabricated at remarkably low cost.

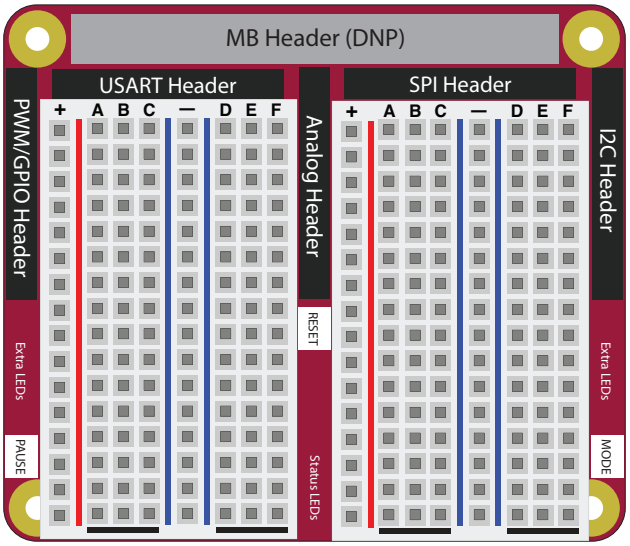
The (1.3" x 0.9") Small Beret Shields connect to the Beret using three 1x9 male headers. The (1.3" x 1.1" or larger) Extended Beret Shields may also include a 1x5 female header connecting to the first row of Signal Header A, whereas the (1.3" x 1.3" or larger) Extended Beret Shields may include a 3x5 female header connecting to all three rows of Signal Header A, including high-current Vs1 or Vin, and GND, on the second and third rows.

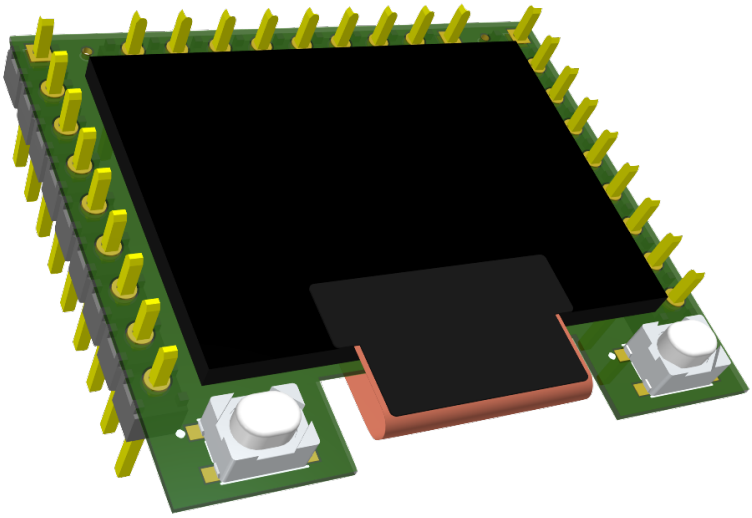
Use of a (1.3" x 0.9") Small Beret Shield leaves unobstructed all JSTs, buttons, and LEDs on the Berets, in addition to all 5 columns of Signal Header A and all 5 columns of Signal Header B, for easy attachment of 10 servo and/or ESC connectors. A (1.3" x 1.1" or 1.3" x 1.1") Extended Beret Shield connects directly to Signal Header A, but leaves unobstructed all 5 columns of Signal Header B.

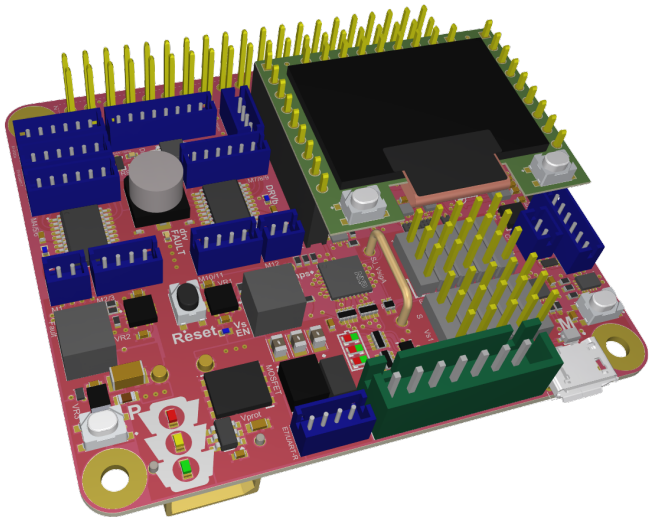
Small and Extended Beret Shields are directly portable across the entire line of Berets. Note that the entry-level **Red** Beret does not have an Analog substem, and the corresponding Analog Header is absent; Beret Shields that do not use the analog subsystem are still fully compatible with this board.

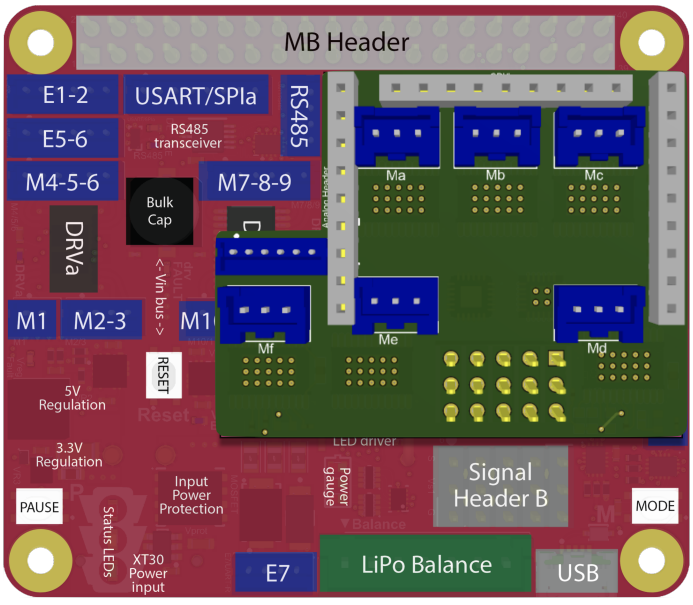
We are also designing a stand-alone high-current brushless motor driver board, in the footprint of a Raspberry Pi, with an STM32G474VE (controllable over a Recon SPI connector), 6x gate drivers, 18x 40V/50A dual MOSFETs, and high-current XT60 (LiPo) and 6x MR30 (BLDC motors) AMASS connectors.











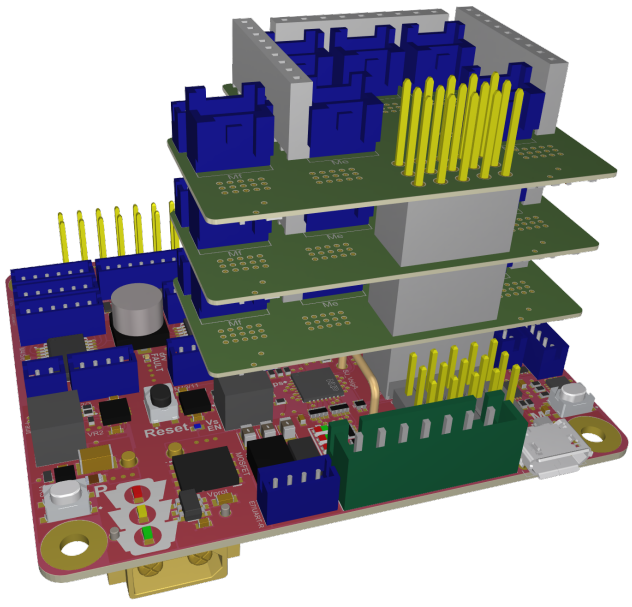


Figure 5.12: Layout of the (top) Prototyping (unplated and plated) and Breadboard, (middle) OLED, and (bottom) BLDC Beret Shields, including various views illustrating how they mount (and, stack) on a Raspberry Beret.

5.9 MB headers, MB header breakout SHIMs, and ID EEPROMs

5.9.1 RPi-compatible motherboards (MBs)

As shown in Table 5.8, the 2x20 header on the **RPi** Berets has 2 pins over which 5V/2A power may be provided to the RPi, 2 pins over which 3.3V power (regulated by the RPi) may be picked up by connected daughterboards, 8 GND pins, and 28 "BCM" pins, arranged in a peculiar order, connected to the RPi's Broadcom MCU. BCM0 and BCM1 are used for an I2C connection to an EEPROM to identify attached daughterboards (a.k.a. HATs). BCM2 through BCM27 may be used as GPIOs, denoted GPIO2 through GPIO27; each of these channels may instead be switched over in software to provide various alternative functions. The "primary" such alternative functions, available on all RPis with 40-pin headers (and most RPi clones, as outlined in Table 1.15), are listed in Table 5.8.

A more comprehensive list of the useful alternative functions of GPIO2 through GPIO27 is given by Table 5.9, which also highlights many of the new alternative functions available with the RPi4. In this table:
- CTS and RTS are (optional) UART hardware flow control (HFC) channels,
- PCM (pulse-code modulation) is an advanced digital audio standard (used, e.g., by hifiberry),
- SD0 is a proprietary Broadcom controller channel used to boot and communicate with the RPi eMMC,

Beret name	alt. function	PWR/BCM	pin	pin	PWR/BCM	alt. function	Beret name
	-	3.3V	1	**2**	**5V**	-	**Vmb**
	I2C1_SDA	GPIO2	3	**4**	**5V**	-	**Vmb**
	I2C1_SCL	GPIO3	5	**6**	**GND**	-	**GND**
mb_G0	GPCLK0	**GPIO4**	**7**	8	GPIO14	UART0_TX	
GND	-	**GND**	**9**	10	GPIO15	UART0_RX	
	SPI1_CE1	GPIO17	11	12	GPIO18	SPI1_CE0	
	SD0_DAT3	GPIO27	13	**14**	**GND**	-	**GND**
	SD0_CLK	GPIO22	15	16	GPIO23	SD0_CMD	
	-	3.3V	17	18	GPIO24	SD0_DAT0	
SPImb_MOSI	**SPI0_MOSI**	GPIO10	**19**	**20**	**GND**	-	**GND**
SPImb_MISO	**SPI0_MISO**	GPIO9	**21**	22	GPIO25	SD0_DAT1	
SPImb_SCK	**SPI0_SCLK**	GPIO11	**23**	**24**†	GPIO8	**SPI0_CE0**	**SPImb_SS**†
GND	-	**GND**	**25**	26*	GPIO7*	SPI0_CE1	mag_INT*/SPImb_SS†
ID_SDA	**I2C0_SDA**	0 (reserved)	**27**	**28**	1 (reserved)	**I2C0_SCL**	**ID_SCL**
mb_G1	GPCLK1	**GPIO5**	**29**	**30**	**GND**	-	**GND**
mb_G2	GPCLK2	**GPIO6**	**31**	32*	GPIO12*	PWM0	bar_INT_DRDY*
imu_INT1_DRDY*	PWM1	GPIO13*	33*	**34**	**GND**	-	**GND**
	SPI1_MISO	GPIO19	35	36	GPIO16	SPI1_CE2	
	SD0_DAT2	GPIO26	37	38	GPIO20	SPI1_MOSI	
GND	-	**GND**	**39**	40	GPIO21	SPI1_SCLK	

Table 5.8: PWR function or BCM (Broadcom pin number, each associated with a GPIO) corresponding to each pin on the 2x20 header on the **RPi** Berets, along with the "primary" alternative function and corresponding Beret name (if any) of each, indicating PWR, GPIO, UART, I2C, SPI, and SDIO channels, as well as PWM and GPCLK functions. Table 5.9 lists the additional functions available on each. **Boldface** indicates channels that are connected by default to the Beret. ()* denotes 3 optional interrupt GPIOs, connected via backside solder jumpers; ()† denotes 2 possible SPI0 SS connections, 1 of which must be selected via a backside solder jumper (default is SPI0_CE0).

- SD1 is a 50 MHz SD/SDIO standard channel for interfacing with secondary SD cards, eMMC, wifi, etc,
- ARM is a universal 6-pin JTAG debugging channel, with adaptive clocking via a Return TCK channel,
- ALT3 mode on BCM8 - BCM11 (on the RPi4) provides slave mode SPI or I2C, denoted here SPIs and I2Cs,
- PWM0/1 are hardware-generated (on the RPi) PWM channels, and GPCLK0/1/2 are gerneral purpose clocks.

Which function is selected on any given pin is configured by the corresponding {ALT0, ALT3, ALT4, ALT5} flag. [As explained at elinux.org, other features not shown (on ALT1 and ALT2) relate primarily to a secondary memory interface and a parallel display interface, neither of which can be used with the Raspberry Beret.]

BCM	pin	ALT0	ALT3	ALT4	ALT5	Beret name	SHIM	pin
0 (res)	**27**	**I2C0_SDA**	-	-	-	**ID_SDA**	**I2C0**	**3**
1 (res)	**28**	**I2C0_SCL**	-	-	-	**ID_SCL**	**I2C0**	**4**
GPIO2	3	I2C1_SDA	-	-	-		I2C1	3
GPIO3	5	I2C1_SCL	-	-	-		I2C1	4
GPIO4	**7**	GPCLK0	*SPI4_CE0*	*UART3_TX*	*I2C3_SDA*	**mb_G0**	**GPIO**	**3**
GPIO5	**29**	GPCLK1	*SPI4_MISO*	*UART3_RX*	*I2C3_SCL*	**mb_G1**	**GPIO**	**4**
GPIO6	**31**	GPCLK2	*SPI4_MOSI*	*UART3_CTS*	*I2C4_SDA*	**mb_G2**	**GPIO**	**5**
GPIO7*	26*	SPI0_CE1	*SPI4_SCLK*	*UART3_RTS*	*I2C4_SCL*	mag_INT*/SPImb_SS†	GPIO	6*
GPIO12*	32*	PWM0	*SPI5_CE0*	*UART5_TX*	*I2C5_SDA*	bar_INT_DRDY*	GPIO	7*
GPIO13*	33*	PWM1	*SPI5_MISO*	*UART5_RX*	*I2C5_SCL*	imu_INT1_DRDY*	GPIO	8*
GPIO10	**19**	**SPI0_MOSI**	*SPIs_MOSI*	*UART4_CTS*	*I2C5_SDA*	**SPImb_MOSI**	**SPI0**	**3**
GPIO9	**21**	**SPI0_MISO**	*SPIs_MISO*	*UART4_RX*	*I2C4_SCL*	**SPImb_MISO**	**SPI0**	**4**
GPIO11	**23**	**SPI0_SCLK**	*SPIs_SCLK*	*UART4_RTS*	*I2C5_SCL*	**SPImb_SCK**	**SPI0**	**5**
GPIO8	**24†**	**SPI0_CE0**	*SPIs_CE*	*UART4_TX*	*I2C4_SDA*	**SPImb_SS†**	**SPI0**	**6†**
GPIO14	8	UART0_TX	*SPI5_MOSI*	*UART5_CTS*	UART1_TX		UART0	3
GPIO15	10	UART0_RX	*SPI5_SCLK*	*UART5_RTS*	UART1_RX		UART0	4
GPIO20	38	PCM_DIN	*SPI6_MOSI*	SPI1_MOSI	GPCLK0		SPI1	3
GPIO19	35	PCM_FS	*SPI6_MISO*	SPI1_MISO	PWM1		SPI1	4
GPIO21	40	PCM_DOUT	*SPI6_SCLK*	SPI1_SCLK	GPCLK1		SPI1	5
GPIO18	12	PCM_CLK	*SPI6_CE0*	SPI1_CE0	PWM0		SPI1	6
GPIO16	36	-	UART0_CTS	SPI1_CE1	UART1_CTS		SPI1	7
GPIO17	11	-	UART0_RTS	SPI1_CE2	UART1_RTS		SPI1	8
GPIO24	18	SD0_DAT0	SD1_DAT0	ARM_TDO	-		SDIO	3
GPIO25	22	SD0_DAT1	SD1_DAT1	ARM_TCK	*SPI4_CE1*		SDIO	4
GPIO26	37	SD0_DAT2	SD1_DAT2	ARM_TDI	*SPI5_CE1*		SDIO	5
GPIO27	13	SD0_DAT3	SD1_DAT3	ARM_TMS	*SPI6_CE1*		SDIO	6
GPIO22	15	SD0_CLK	SD1_CLK	ARM_TRST	*I2C6_SDA*		SDIO	7
GPIO23	16	SD0_CMD	SD1_CMD	ARM_RTCK	*I2C6_SCL*		SDIO	8

Table 5.9: Alternative functions of each of the 28 digital i/o pins on the RPi header, and the corresponding Beret names (if any). *Italics* indicate functions that are available on the RPi4 only. As in Table 5.8, indicated are GPIO, UART, I2C, SPI, SDIO channels, as well as PWM and GPCLK functions with, as before, **boldface** indicating default connections, and ()* and ()† indicating optional connections, between the Beret and the RPi. BCM pins 0 through 8 have default pull up resistors, and BCM pins 9 through 27 have default pull down resistors. Data from the RPi v4 datasheet, elinux.org, and the RPi forums. Note that {*SPIs_MOSI*, *SPIs_SCLK*} on the RPi4 (pins 19 and 23 in ALT3 mode) can also function as {*I2Cs_SDA*, *I2Cs_SCL*}.

As indicated by Table 5.9, the RPi4 (when NOT connected to the Beret) can simultaneously operate, e.g.:

- 4 UART channels: UART0/3/4 (w/ HFC) and UART5 (w/o HFC), or
- 4 SPI channels (w/ 8 total SS lines): SPI0 (w/ CE0), SPI1 (w/ CE0/1/2), and SPI4/5 (each w/ CE0/1), or
- 6 I2C channels: I2C1/3/4/5/6, plus (in slave mode) I2Cs [in addition to I2C0, reserved for the EEPROM], or
- mix A: UART1 (w/ HFC), UART4/5 (w/o HFC), SPI4/6 (each w/ CE0/1), and I2C1/5/6, or
- mix B: UART0 (w/o HFC), SPIs (in slave mode), SPI4 (w/ CE0), SPI1 (w/ CE0/1/2), I2C1, PWM0/1, SD1.

When an RPi2, RPi3, RPi4, or RPi Zero is fully connected via the RPi Header to a Beret, including the MB SPI channel, the MB ID I2C channel, the interrupt connections {imu_INT1_DRDY, bar_INT_DRDY, mag_INT}, and the GPIOs {mb_G0, mb_G1, mb_G2}, the RPi still has 16 unused BCM channels available on the RPi Header. These channels can be used as GPIOs or, alternatively, can simultaneously operate, e.g.:

- I2C1, UART0 (w/o HFC), SPI1 (w/ 3 available SS lines), and either SD1 or JTAG.

If using an RPi4, and/or not using one or more of the (optional) sensor interrupt (INT) connections mentioned above, various alternative channels and functions also become available if needed, as shown in Table 5.9.

In summary, even though the RPi Berets connect to up to 10 of the BCM2 to BCM27 channels on the RPi Header, in addition to the ID pins on BCM0/1, substantial connectivity options remain for connecting the RPi to other boards or devices. Further, using different SS pins on the SPI0 channel for each board, and programming the {mb_G0, mb_G1, mb_G2} channels appropriately, two RPi Berets can be directly attached to a single RPi using COTS RPi HAT stacking solutions (see, e.g., here).

As discussed further in the paragraph below, convenient MB header breakout SHIMs, which may be used even when one or more Beret(s) are attached to the MB, are available separately, and may be used to break out the additional functionality on the MB header discussed above onto standard Recon connectors.

MB Header Breakout SHIM. A small **SHIM** is under development to conveniently break out all 28 BCM pins of the RPi Header, one functional group at a time, in Recon order, in addition to incorporating an SD card holder and a multichannel LED display driver. **TODO**: include further explanation and a pic of this SHIM, once available. **Warning:** Though the MB Header breakout JSTs on this SHIM can provide 3.3V or 5V power, as selected via backside power jumpers, they operate at 5V TTL, not 3.3V TTL, so any devices connected to the JSTs on this SHIM must be 5V tolerant.

ID EEPROM. On the RPi Berets, the UDFN8 version of the (32 Kb) CAT24C32 EEPROM is used for board identification, programmed as described in the RPi HAT ID EEPROM spec. The 7-bit address used for this EEPROM is 1010000b (0x50h) by default, as required by this spec; however, the last bit of this device's I2C address may be changed, via a backside solder jumper, to 3.3V or GND, thus enabling the use of either 0x50h or 0x51h as the ID EEPROM address on these Berets, and making the connection of two Berets to a single RPi straightforward (using an RPi header extension cable) while keeping both of the ID EEPROMs of the connected Berets individually readable. Note that the write protect (WP) pin on the EEPROM is by default connected to 3.3V (read only), but this may also be switched to GND (to enable write mode) via a backside solder jumper.

5.9.2 96B-compatible MBs

As shown in Table 5.10, the 2x20 header on 96B motherboards has 2 pins over which Vmb = 8V to 18V power may be provided to the 96B motherboard by connected daughterboards (aka **mezzanines**, e.g. the **Black** Beret), 1 pin over which 5V power (regulated by the 96B motherboard) may be picked up by connected mezzanines, 1 pin over which 1.8V power (regulated by the 96B motherboard) may be picked up by connected mezzanines, 4 GND pins, and 32 other pins, arranged in a rather well-structured order by their primry functions as defined by the MCU on the 96B motherboard. The pins on this header that are connected to the **Black** Beret are also indicated in Table 5.10. **Warning:** All digital pins on the 96B header operate at 1.8V TTL, and thus must usually be level shifted on attached mezzanines (e.g. to be used by 3.3V MCUs, as implemented on Berets).

MB Header Breakout SHIM. The pins on the 96B Header (Table 5.10) are logically ordered by their associated functions. A SHIM designed to level shift these functions to (5V tolerant) 3.3V TTL, and present these functions on JSTs in Recon order (with associated power/GND pins) will be developed if sufficient interest is expressed.

ID EEPROM. On the 96B Beret, the UDFN8 version of the (1 Mb) CAT24M01WI-GT3 EEPROM is used for board identification, programmed (for the moment) as described in §5.9.1 (as for the RPi Berets); the programming of this (larger-capacity) EEPROM is subject to change during the next rev of the 96B Mezzanine Design Guidelines.

Beret name	function	pin	pin	function	Beret name
GND	**GND**	**1**	**2**	**GND**	**GND**
	UART0_CTS	3	4	PWR_BTN_N	
	UART0_TxD	5	6	RST_BTN_N	
	UART0_RxD	7	**8**	**SPI0_SCLK**	**SPImb_SCK**
	UART0_RTS	9	**10**	**SPI0_DIN**	**SPImb_MISO**
	UART1_TxD	11	**12**	**SPI0_CS**	**SPImb_CS0**
	UART1_RxD	13	**14**	**SPI0_DOUT**	**SPImb_MOSI**
ID_SCL	**I2C0_SCL**	**15**	16	PCM_FS	
ID_SDA	**I2C0_SDA**	**17**	18	PCM_CLK	
	I2C1_SCL	19	20	PCM_DO	
	I2C1_SDA	21	22	PCM_DI	
mb_G0	**GPIO-A**	**23**	24	GPIO-B	
mb_G1	**GPIO-C**	**25**	26	GPIO-D	
mb_G2	**GPIO-E**	**27**	28	GPIO-F	
mag_INT*	GPIO-G	29*	30	GPIO-H	
bar_INT_DRDY*	GPIO-I	31*	32	GPIO-J	
imu_INT1_DRDY*	GPIO-K	33*	34	GPIO-L	
	1V8	35	**36**	**SYS_DCIN**	**Vmb**
5V	**5V**	**37**	**38**	**SYS_DCIN**	**Vmb**
GND	**GND**	**39**	**40**	**GND**	**GND**

Table 5.10: Primary functions corresponding to each pin on the 2x20 header of the 96B format, along with the corresponding net name (if any) on the **Black** Beret, indicating PWR, GPIO, UART, I2C, and SPI. **Boldface** indicates channels that are connected by default to the Beret. ()* denotes 3 optional interrupt GPIOs, attached to the header via backside solder jumpers.

5.9.3 BB-compatible MBs

As shown in Tables 5.11 and 5.12, the 2x23 header on BB motherboards has 2 pins over which a Vmb ≈ 5V power supply may be provided to the BB motherboard by connected daughterboards (aka capes, e.g. the **White** Beret), 2 pins over which 5V power (regulated by the 96B motherboard) may be picked up by connected capes, 2 pins over which 5V power (regulated by the BB motherboard) may be picked up by connected capes, 6 GND pins, and 34 other pins with various digital and analog functions. The pins on this header that are connected to the **White** Beret are also indicated in Tables 5.11 and 5.12.

MB Header Breakout SHIM. The pins on the BB Header (Tables 5.11 and 5.12) are generally clustered by their associated functions. A SHIM designed to present these pins on JSTs in Recon order (with associated power/ground pins) will be developed if sufficient interest is expressed.

ID EEPROM. On the BB Berets, the UDFN8 version of the (32 Kb) CAT24C32 EEPROM (as also ued on the RPi Berets) is used for board ID, programmed as described, e.g., in this BB ID EEPROM programming tutorial.

Beret name	alt. function	PWR/GPIO	pin	pin	PWR/GPIO	alt. function	Beret name
GND	-	**DGND**	1	**2**	**DGND**	-	**GND**
	-	3.3V	3	4	3.3V	-	
Vmb	-	**VDD_5V**	**5**	**6**	**VDD_5V**	-	**Vmb**
	-	SYS_5V	7	8	SYS_5V	-	
	PWR_BTN	-	9	10	-	SYS_RESET	
	UART4_RX	GPIO_30	11	12*	GPIO_60*	-	mag_INT*
	UART4_TX	GPIO_31	13	14*	GPIO_40*	PWM1A	bar_INT_DRDY*
	-	GPIO_48	15	16*	GPIO_51*	PWM1B	imu_INT1_DRDY*
SPImb_SS	**SPI0_CS0**	GPIO_4	**17**	**18**	GPIO_5	**SPI0_D1**	**SPImb_MISO**
ID_SCL	**I2C2_SCL**	-	**19**	**20**	-	**I2C2_SDA**	**ID_SCL**
SPImb_MOSI	**SPI0_D0**	GPIO_3	**21**	**22**	GPIO_2	**SPI0_SCLK**	**SPImb_SCK**
mb_G0	-	**GPIO_49**	**23**	24	GPIO_15	UART1_TX	
mb_G1	-	**GPIO_117**	**25**	26	GPIO_14	UART1_RX	
mb_G2	-	**GPIO_125**	**27**	28	GPIO_123	SPI1_CS0	
	SPI1_D0	GPIO_111	29	30	GPIO_112	SPI1_D1	
	SPI1_SCLK	GPIO_110	31	32	VDD_ADC	-	
	ADC_IN4	-	33	34	GND_ADC	-	
	ADC_IN6	-	35	36	-	ADC_IN5	
	ADC_IN2	-	37	38	-	ADC_IN3	
	ADC_IN0	-	39	40	-	ADC_IN1	
	-	GPIO_20	41	42	GPIO_7	SPI1_CS1	
GND	-	**DGND**	**43**	**44**	**DGND**	-	**GND**
GND	-	**DGND**	**45**	**46**	**DGND**	-	**GND**

Table 5.11: PWR function or GPIO number, and the "primary" alternative function, corresponding to each pin on the 2x23 header on the **BB Black**, along with the corresponding net name (if any) on the **White** Beret, indicating PWR, GPIO, UART, I2C, and SPI, as well as PWM and ADC functions (cf. Table 5.12 for the corresponding numbering of the GPIO, UART, SPI, I2C, and PWM channels on the BB AI). **Boldface** indicates channels that are connected by default to the Beret. ()* denotes 3 optional interrupt GPIOs, attached to the header via backside solder jumpers.

Beret name	alt. function	PWR/GPIO	pin	pin	PWR/GPIO	alt. function	Beret name
GND	-	**DGND**	1	**2**	**DGND**	-	**GND**
	-	3.3V	3	4	3.3V	-	
Vmb	-	**VDD_5V**	**5**	**6**	**VDD_5V**	-	**Vmb**
	-	SYS_5V	7	8	SYS_5V	-	
	PWR_BTN	-	9	10	-	SYS_RESET	
	UART5_RX	GPIO_241	11	12*	GPIO_128*	-	mag_INT*
	UART5_TX	GPIO_172	13	14*	GPIO_121*	PWM3A	bar_INT_DRDY*
	-	GPIO_76	15	16*	GPIO_122*	PWM3B	imu_INT1_DRDY*
SPImb_SS	**SPI2_CS0**	GPIO_209	**17**	**18**	GPIO_208	**SPI2_D0**	**SPImb_MISO**
ID_SCL	**I2C4_SCL**	-	**19**	**20**	-	**I2C4_SDA**	**ID_SCL**
SPImb_MOSI	**SPI2_D1**	GPIO_67	**21**	**22**	GPIO_179	**SPI2_SCLK**	**SPImb_SCK**
mb_G0	SPI2_CS1	**GPIO_203**	**23**	24	GPIO_175	UART10_TX	
mb_G1	-	**GPIO_177**	**25**	26	GPIO_174	UART10_RX	
mb_G2	-	**GPIO_111**	**27**	28	GPIO_113	SPI3_CS0	
	SPI3_D1	GPIO_139	29	30	GPIO_140	SPI3_D0	
	SPI3_SCLK	GPIO_138	31	32	VDD_ADC	-	
	ADC_IN4	-	33	34	GND_ADC	-	
	ADC_IN6	-	35	36	-	ADC_IN5	
	ADC_IN2	-	37	38	-	ADC_IN3	
	ADC_IN0	-	39	40	-	ADC_IN1	
	-	GPIO_180	41	42	GPIO_114	SPI3_CS1	
GND	-	**DGND**	**43**	**44**	**DGND**	-	**GND**
GND	-	**DGND**	**45**	**46**	**DGND**	-	**GND**

Table 5.12: PWR function or GPIO number, and the "primary" alternative function, corresponding to each pin on the 2x23 header on the **BB AI**, along with the corresponding net name (if any) on the **White** Beret, indicating PWR, GPIO, UART, I2C, and SPI, as well as PWM and ADC functions (cf. Table 5.11 for the corresponding numbering of the GPIO, UART, SPI, I2C, and PWM channels on the BB Black). **Boldface** indicates channels that are connected by default to the MB Header on the Beret. ()* denotes 3 optional interrupt GPIOs, connected to the MB Header via backside solder jumpers on the Beret. Note in particular that, between the BB Black pinout depicted in Table 5.11, and the BB AI pinout depicted here, the D0 & D1 nets are swapped on pins 18 & 21, and on pins 29 & 30; these swaps are easily accounted for in software.

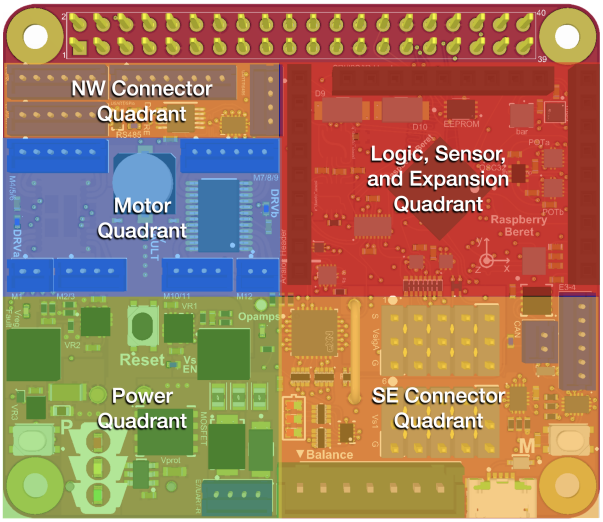


Figure 5.13: General layout of the Berets, as organized into quadrants.

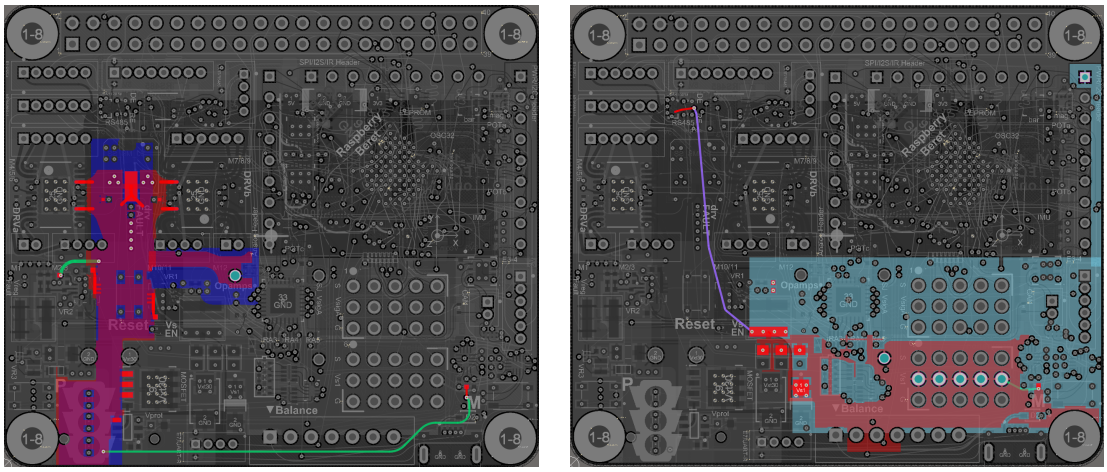


Figure 5.14: (left) Stitching of Vin traces on layers 1 and 8. (right) Stitching of Vs1 traces on layers 1 and 4.

5.10 Layout

5.10.1 Overall organization and power flow

As indicated previously, it is convenient to refer to directions on the Berets in terms of directions on a compass. As illustrated in Figure 5.13, the layout of the Berets is generally organized into four main quadrants:

- the Logic, Sensor, and Expansion Quadrant, in the NE,
- the Connector Quadrant, split between the NW and SE,
- the Power Quadrant, in the SW, and
- the Motor Quadrant, located just N of Power Quadrant.

The bottom layer of the Power and Motor Quadrants is mostly large Power and GND pours, with many "thermal vias" connected to the (hot) undersides of the high-power components, for enhanced **thermal radiation**. Also, **short** and **wide** traces are used for all high-current pathways. **Stitching** of overlying vias is performed across multiple current-carrying layers for the highest-current pathways, including those taking Vin from the Power Quadrant to the Motor Quadrant, as shown in Figure 5.14a, and those taking Vs1 from the Power Quadrant to the SE Connector Quadrant, as shown in Figure 5.14b.

Layer	Color	Logic & Connector Quadrants Function	Logic & Connector Quadrants Trace Directions	Power & Motor Quadrants Function	copper oz/ft^2
Top		Signal/ICs	N-S, E-W	Power/Signal/ICs	1.5
2		GND	(fills)	Power	2
3		Signal	E-W	Power/Signal	1
4		GND/5V/Vs1	(fills)	GND/5V/Vs1	2
5		GND/5V/Vs1	E-W	GND/5V/Vs1	1
6		Signal	N-S	Signal	1
7		3.3V	(fills)	3.3V	2
Bottom		Power/Signal	N-S	GND	1.5

Table 5.13: Eight-layer stackup used on the Berets, indicating the colors used in Figures 5.14, 5.16, and 5.17.

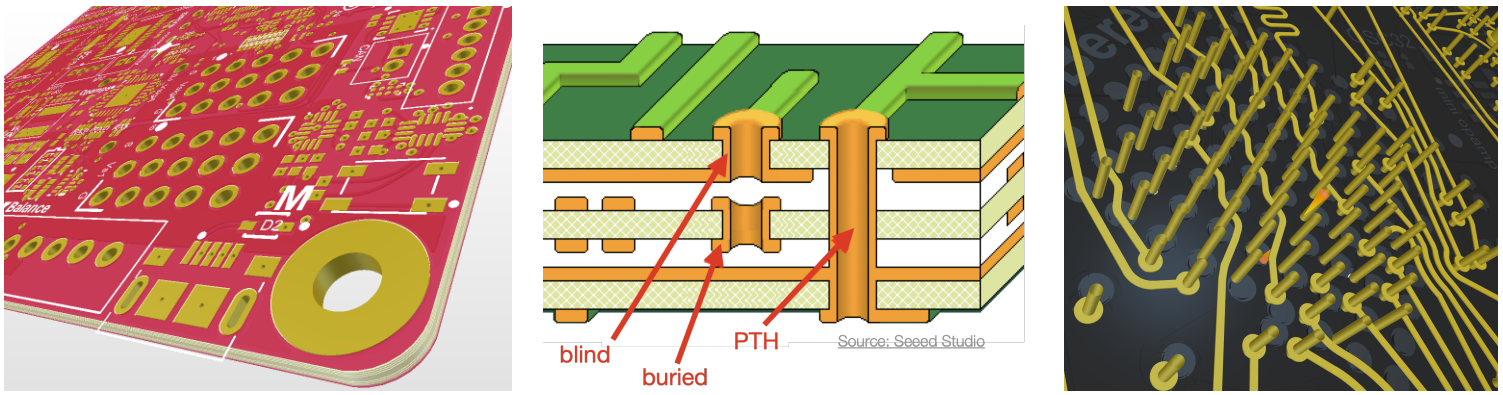


Figure 5.15: (left) Closeup of the SE corner of a Beret, illustrating the extensive use of via-in-pad techniques. (center) Comparison of blind, buried, and through-hole vias. (right) Copper traces on one of the layers under the BGA on a Beret, illustrating the removal of unnecessary annular rings, which gives substantially increased clearance for breaking out the 8mm pitch BGA using 6 mil traces, 7 mil spaces, and through-hole vias only.

5.10.2 Layer stackup, signal routing, and high-density integration (HDI)

With some careful design effort (see in particular the stackup plan in Table 5.13), Berets achieve a remarkabe high-density integration (**HDI**) of components and functionality in a very small footprint for an 8-layer board. This is achieved, in part, by making extensive use of **via-in-pad** technology, which allows the placement of solder pads for various components directly over vias, as illustrated in Figure 5.15a. The (per board, not per instance) cost of implementing this modern (but by now fairly common) technology for HDI, which necessitates plugging the vias and plating them over, is *much* less than the cost of traditional **blind vias** (exposed on only one side of the PCB) and **buried vias** (not exposed on either side of the PCB), as illustrated in Figure 5.15b. Indeed, *all* vias on the Berets are in fact the (much lower-cost) **through-hole vias**.

Careful tradeoffs were involved in the stackup design (Table 5.13). Layer thicknesses had to be made:

- sufficiently thin to use 6 mil traces, 7 mil spaces, 6 mil diameter vias, and 18 mil diameter annular rings to break out the 8 mm (31.5 mil) pitch BGA (ball grid array) on the STM, and
- sufficiently thick to handle high current (up to 12A in places) where necessary.

A useful technique implemented to break out the BGA at this resolution was the removal of the (unnecessary) annular rings on the (through-hole) vias under the BGA on layers in which these vias did not actually connect to traces, as illustrated in Figure 5.15c. This resulted in increased clearance to route traces out from under the BGA between the closely-spaced vias with, on any given layer, most of these annular rings removed.

Layout of all 8 layers of the Raspberry Beret is illustrated in full in Figures 5.16-5.17. As indicated in Table 5.13, note that different thicknesses of copper are used on these different layers, based on their differing primary functions, which is helpful to better handle the high-current traces/pours. Though this is a tad unusual, most PCB fab facilities can accomodate this when fabricating PCBs at high volume.

To address the routing of the many crossed traces in the design, the following approach was followed:

- primarily N-S traces were isolated on layers 6 and 8 (see, e.g., the purple traces of Layer 6 in Figure 5.17), while
- primarily E-W traces were isolated on layers 3 and 5 (see, e.g., the green traces of Layer 5 in Figure 5.17).

Carefully selecting where such traces are joined (using vias) facilitated the "untangling" of the hundreds of nets involved in the design.

5.10.3 EMI and signal-integrity considerations

Four primary techniques were used to maintain **signal integrity** on high-speed communication channels (SPI, USART, I2C) and (simultaneously) to reduce the electromagnetic interference (**EMI**) generated by the board:

1. GND and/or Power planes were situated immediately next to each high-speed signal trace[35].
2. Curved traces with no sharp corners were used everywhere.
3. Matched-length traces were used, on each layer, for the parallel traces associated with clocked high-speed communication channels (e.g., MOSI, MISO, SCK), as illustrated, e.g., by the extra wiggles of the purple traces in the NE corner of Layer 5 in Figure 5.17.
4. Power GND, which inevitably fluctuates some due to the strongly time-varying loads placed on it (associated with the PWM generation used by the high-power components), was carefully isolated from Signal GND.

The use of modern ECAD software (Altium) was essential in order to implement techniques 2 and 3 above.

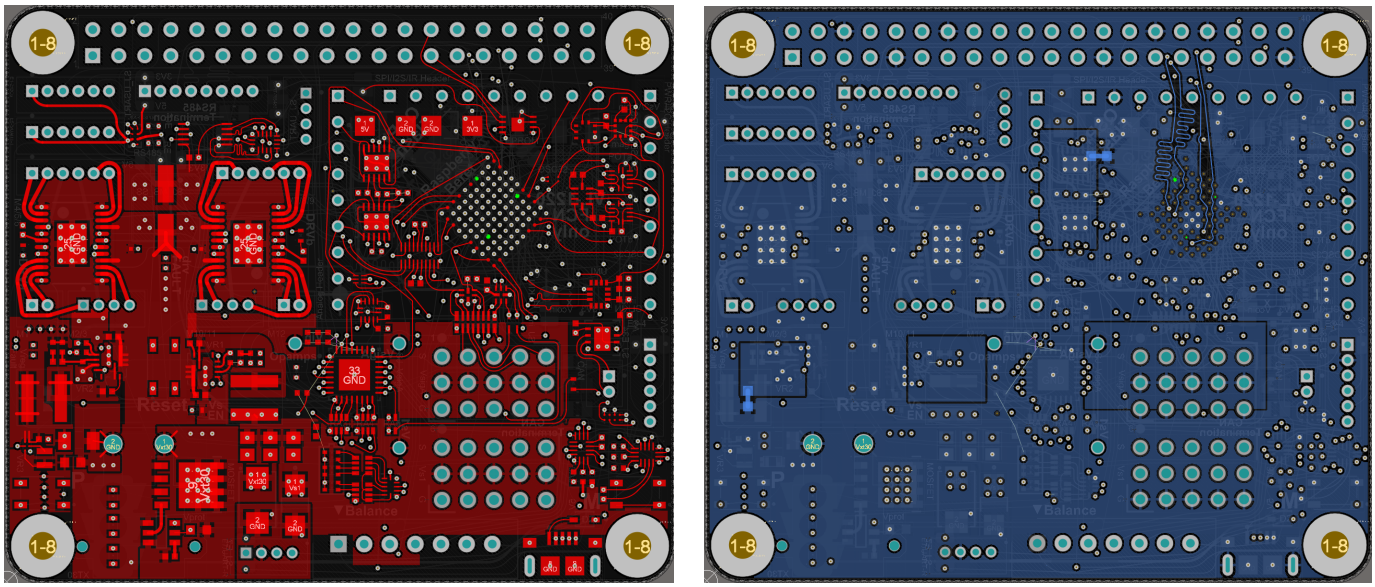


Figure 5.16: Layout of layers 1 and 2 of the Raspberry Beret (continued on next page).

[35] A high-speed signal is actually mostly carried in the space *between* the trace and the corresponding reference layer, not along the trace itself, so the close proximity of such reference layers is essential in order to not turn your PCB into a radio unintentionally!

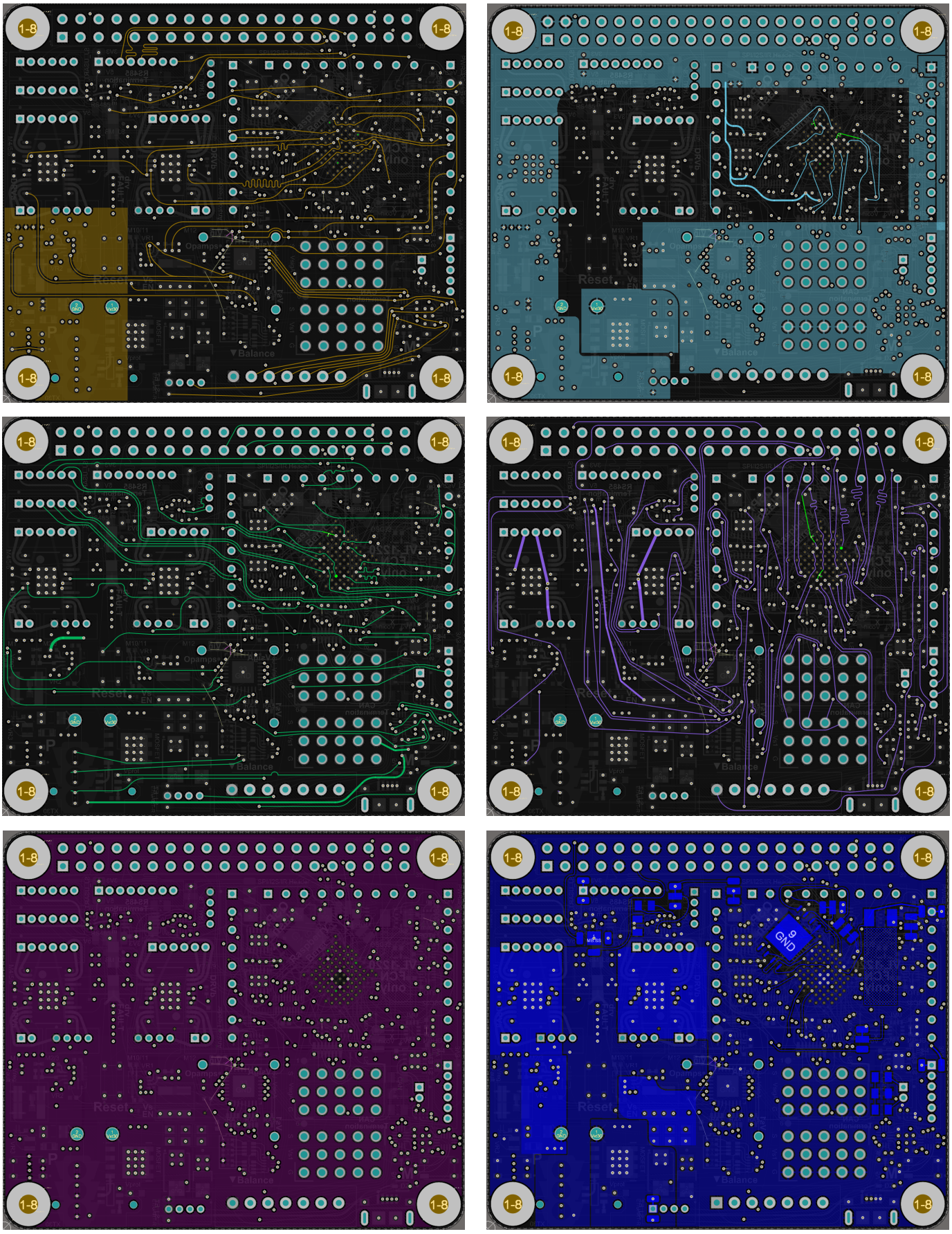


Figure 5.17: Layout of layers 3 through 8 of the Raspberry Beret (continued from previous page).

manufacturer	part	description	$	R	r	K	W	G	B	§
TI	DRV8912-Q1	motor drivers	2.42	2	1	2	2	0	1	5.3
TI	CSD18510Q5B	power MOSFET	0.75	1	1	1	1	1	1	5.2.3
TI	LM74700-Q1	ideal diode controller	0.62	1	1	1	1	1	1	5.2.3
TI	TPS56637	Vin->Vs1, Vin->Vmb Vregs	1.32	2	2	2	2	2	2	5.2.4, 5.2.5, 5.2.6
TI	TI TPS6208833	Vmb->3.3V switching Vreg	1.10	1	1	0	1	0	0	5.2.7
TI	TI TPS62913	Vmb->3.3V switching Vreg	1.10	0	0	1	0	0	0	5.2.7
TI	TS5A3359	Venc mux (3:1+off)	0.34	1	1	1	1	1	1	5.2.9
TI	TXB0108DQSR	8 channel 5V - 3.3V level shifter	0.39	1	0	1	1	0	0	5.5
TI	TCA6507	LED driver	0.40	1	1	1	1	1	1	5.5
TI	THVD1452	RS485 transceiver	1.003	1	0	1	1	1	0	5.6.4
TI	TCAN334GDCNR	CAN-FD transceiver	1.114	1	0	1	1	1	0	5.6.4
TI	SN74CBTLV3257	RS485-UART mux	0.200	1	0	1	1	1	0	5.6.4
TI	ALM2402-Q1	power opamps (DAC1, DAC2, Vs2, user)	1.13	2	0	2	2	2	2	5.7.1, 5.7.1, 5.7.3
TI	TLV9002	dual mini opamp (low-pass filters)	0.21	1	0	1	1	1	1	5.7.2
TI	TPL0102-100	digital pots (Vs1, Vs2, $\omega_{c1a/b}$, $\omega_{c2a/b}$)	0.60	3	1	3	3	3	3	5.2.4, 5.7.1, 5.7.2
TI	TMUX1208	Vmon mux (8:1+off)	0.32	2	2	2	2	0	0	5.7.4
TI	TS5A23166	Vmon SPDT switch	0.24	1	1	1	1	0	0	5.7.4
TI	TMUX1204DQAR	Vmon 4:1 switch	0.09	0	0	0	0	1	1	5.7.4
ST	STM32G474VEH6	microprocessor with 512KB Flash	4.488	1	0	1	1	1	1	5.5
ST	STM32G474VBH6	microprocessor with 128KB Flash	3.521	0	1	0	0	0	0	5.5
ST	LIS3MDLTR	3-axis magnetometer	0.714	1	1	1	1	1	1	5.4
ST	LPS22HB	barometer	1.28	1	1	1	1	1	1	5.4
TDK	ICM-42688-P	6-axis IMU	2.997	1	1	1	1	1	1	5.4
NXP	PCAL6524HEHP	GPIO expander	0.825	1	1	1	1	0	0	5.5
SiTIME	SIT1532AC	32.7680 kHz oscillator (STM RTC & IMU)	0.60	1	1	1	1	1	1	5.5.1, 5.4
ON	CAT24C32HU4I-GT3	32 Kb I2C EEPROM	0.276	1	1	0	1	0	0	5.9.1
ON	CAT24M01WI-GT3	1 Mb I2C EEPROM	0.523	0	0	1	0	0	0	5.9.2

Table 5.14: BOM Part A: primary components of the **Ras**, **Red**, **Black**, **White**, **Green**, **Blue** Berets (R, r, K, W, G, B). Identifying components by the underlined abbreviated names is convenient.

5.11 Bill Of Materials (BOM)

The Bill Of Materials of the Raspberry Beret is listed in Tables 5.14-5.15. A complete BOM, including all minor components (including, e.g., all of the small 0402 discrete resistors and capacitors) are listed in the Altium viewer for the project, available online at http://dynamics.ucsd.edu/berets. Some commonly-needed add-on components are listed in Table 5.16. Random notes and questions:

1. The custom 7-pin XH-compatible has a slot cut out of one side of a JST-XH, so 3-pin to 6-pin connectors can be plugged in as well. We need to find someone to manufacture that for us. See the Revolectrix SPA Single Port Safe Parallel Adapter and the ISDT PC-4860 1S-6S Lipo Battery Charger for examples.

2. Do we need common-mode chokes for Electro Magnetic Interference (EMI) filtering on USB, RS485, or CAN? This post says it degrades USB signal quality, but might be necessary to pass FCC Title 47 CFR Part 15. This post discusses it for CAN. For RS485 and CAN, maybe we can leave places for it off of the Beret, out on the enclosure bulkhead (next to the spot for the optional TVS diodes; see §5.6.4).

manufacturer	part	description	$	R	r	K	W	G	B	§
Littelfuse	SMBJ33CA	main TVS diode for Vin spikes	0.140	1	1	1	1	1	1	5.2.3
ON	1SMA5928BT3G	13V zener diode for 12V spikes	0.0924	1	1	2	1	1	1	5.2.3
ON	1SMA5919BT3G	5.6V zener diode for 5V spikes	0.0924	1	1	1	1	0	0	5.2.3
ON	1SMA5914BT3G	3.6V zener diode for 3.3V spikes	0.0924	1	1	1	1	1	1	5.2.3
ST	BAT60JFILM	10V Schottky for USB 5V protection	0.0363	1	1	1	1	1	1	5.2.3
TI	TPD6E004	ESD diode arrays for USB, S1-S10	0.16	2	2	2	2	2	2	5.2.3
Bourns	SRP5050FA-5R6M	5.6 μH (7.2A) inductors on Vs1, Vmb	0.533	2	2	2	2	2	2	5.2.4, 5.2.5
Vishay Dale	IHHP0806ABERR22M01	220 nH (5.3A) inductor on 3.3V	0.148	1	1	0	1	1	1	5.2.7
Coincraft	XGL4030-472	4.7 μH (3.2A) inductor on 3.3V	0.49	0	0	1	0	0	0	5.2.7
Chemi-Con	EMZR350ARA101MF61G	100 μF bulk cap on Vin	0.220	1	1	1	1	1	1	5.2.4
Samsung	CL21A226MAYNNNE	22 μF output caps on Vs1	0.0716	3	3	3	3	3	3	5.2.4
TDK	C3216X5R0J686M160AB	68 μF output cap on Vmb	0.2871	1	1	0	1	0	0	5.2.5
Taiyo-Yuden	EMK316BBJ476ML-T	47 μF output cap on Vmb	0.2210	0	0	1	0	0	0	5.2.5
Samsung	CL10A226MQ8NRNE	22 μF output cap on 3.3V	0.0527	1	1	0	1	1	1	5.2.7
Murata	GRM188R60J476ME15D	47 μF output cap on 3.3V	0.1493	0	0	1	0	0	0	5.2.7
Panasonic	EXB-28V820JX	82 Ω 4RA for {B,G} LEDs, S1-S10	0.0084	1	1	1	1	1	1	5.2.3
Panasonic	EXB-24V820JX	82 Ω 2RA for {B,G} LEDs, S1-S10	0.0105	1	1	1	1	1	1	5.2.3
Panasonic	EXB-28V221JX	220 Ω 4RA for {A,Y,R,R/G} LEDs	0.0084	1	1	1	1	1	1	5.2.3
Panasonic	EXB-24V221JX	220 Ω 2RA for {A,Y,R,R/G} LEDs	0.0105	1	1	1	1	1	1	5.2.3
Inolux	IN-S42BTR	red LED (stoplight)	0.0606	1	1	1	1	1	1	5.6.8
Inolux	IN-S42BT5Y	yellow LED (stoplight)	0.0707	1	1	1	1	1	1	5.6.8
Inolux	IN-S42BT5G	green LED (stoplight)	0.0818	1	1	1	1	1	1	5.6.8
Kingbright	APHB1608LZGKSURKC	bicolor LEDs (power gauge)	0.2282	3	3	3	3	3	3	5.6.8
Inolux	IN-S42BT5B	blue LEDs (enable status)	0.0873	6	5	6	6	2	2	5.6.8
Inolux	IN-S42BT5A	amber LEDs (fault status)	0.0707	3	3	3	3	2	2	5.6.8
C&K	PTS815-SJM-250-SMTR	white buttons (pause, mode)	0.111	2	2	2	2	2	2	5.6.8
C&K	PTS815-SJG-250-SMTR	black button (reset)	0.143	1	1	1	1	1	1	5.6.8
Amass	XT30PW-M	sideways XT30 (main power in)	0.500	1	1	1	1	1	1	5.2.1
custom	custom	custom 7-pin XH (Balance)	cost?	1	1	1	1	0	0	5.7.4
4Ucon	01056	3x5 0.1" male (SIGa, SIGb)	0.0422	2	1	2	2	2	1	5.6.2
4Ucon	11071	USB Micro-B female	0.0795	1	1	1	1	1	1	5.6.9
4Ucon	00532	1x9 0.1" female for Analog, SPI, I2C Headers	0.0625	3	2	3	3	3	3	5.8
JST	B8B-ZR-3.4(LF)(SN)	8-pin JST-ZH for USART	0.171	1	1	1	1	1	1	5.6.5
JST	B6B-ZR-3.4(LF)(SN)	6-pin JST-ZH for M4-5-6, M7-8-9, E1-2, E3-4, E6-7	0.131	5	3	5	5	2	3	5.3, 5.6.1
JST	B4B-ZR-3.4(LF)(SN)	4-pin JST-ZH for M2-3, M10-11, E5 RS485/UART	0.096	4	2	4	4	2	3	5.3, 5.6.1, 5.6.4
JST	B2B-ZR-3.4(LF)(SN)	2-pin JST-ZH for M1, M12, CAN	0.072	3	1	3	3	1	1	5.6.5
4Ucon	20565	2x20 0.1" stackable RPi header	0.4139	1	1	0	0	0	0	5.9.1
4Ucon	00324 pins too short?	2x20 2mm 96B header	0.4139	0	0	1	0	0	0	5.9.2
4Ucon	20582	2x23 0.1" stackable BB header	0.4139	0	0	0	1	0	0	5.9.3

Table 5.15: BOM Part B: secondary components for the **Ras**, **Red**, **Black**, **White**, **Green**, **Blue** Berets (R, r, K, W, G, B). This table still under construction.

manufacturer	part	description	$	#	§	usage notes
Panasonic	VL1220/FCN	rechargeable 3V coin cell	4.03	1	5.2	with SMT bracket (solder on backside)
Winbond	W25Q64JVZEIQ	8 MB 133 MHZ	1.053	1	5.5.2	QSPI Flash (solder on backside)
Winbond	W25N512GVEIG	64 MB 166 MHz	1.95			
GigaDevice	GD5F1GQ4UFYIGR	128 MB 120MHz	3.05			
4Ucon	00812	1x9 0.1" male	0.0207	3	5.6.2	mates with Analog, SPI, I2C headers
4Ucon	00526	1x5 0.1" female	0.0370	1	5.8	mates with first row of SigA header

Table 5.16: Commonly-needed add-on components for Berets, including the rechargeable coin cell, flash memory, and connectors for Beret Shields. Prices quoted are for single unit quantities as of Spring 2021, except for the 4Ucon connectors, which are quoted for quantities of 1000. The components used for CAN and RS485 termination (also an optional add-ons) are standard 0804 resistors and capacitor (see §5.6.4), solder footprints for which are provided on the back of the Berets. This table still under construction.

5.12 Schematics

A shared schematic arrangement is used to define the six Berets. The **Raspberry** Beret, which was designed first, uses eleven schematic sheets (included as the following eleven pages of this datasheet):

1. **Master_Raspberry** (the main datasheet that connects all others for the **Raspberry** Beret; see §5.1),
2. **Power** (defines the wiring of the various voltage regulators on the PCB; see §5.2),
3. **Motors** (defines the wiring of the DRV8912-Q1 motor drivers; see §5.3),
4. **Sensors** (defines the wiring of the IMU, magnetometer, and barometer; see §5.4),
5. **MCU** (defines the pinouts of the STM32, GPIO expander, Level Shifter, Flash, and OSC32; see §5.5),
6. **Connectors** (defines the wiring of most of the connectors on the PCB; see §5.6),
7. **Signal_Headers** (defines the wiring of the Signal Headers; see §5.6.2),
8. **UI** (i.e., User Interface, defines the wiring of the buttons and LEDs; see §5.6.8),
9. **Analog** (defines the wiring of the analog subsystem; see §5.7),
10. **Vmon** (defines the wiring of the voltage monitoring circuit; see §5.7.4), and
11. **Header_RPi** (defines the wiring to the RPi header; see §5.9.1).

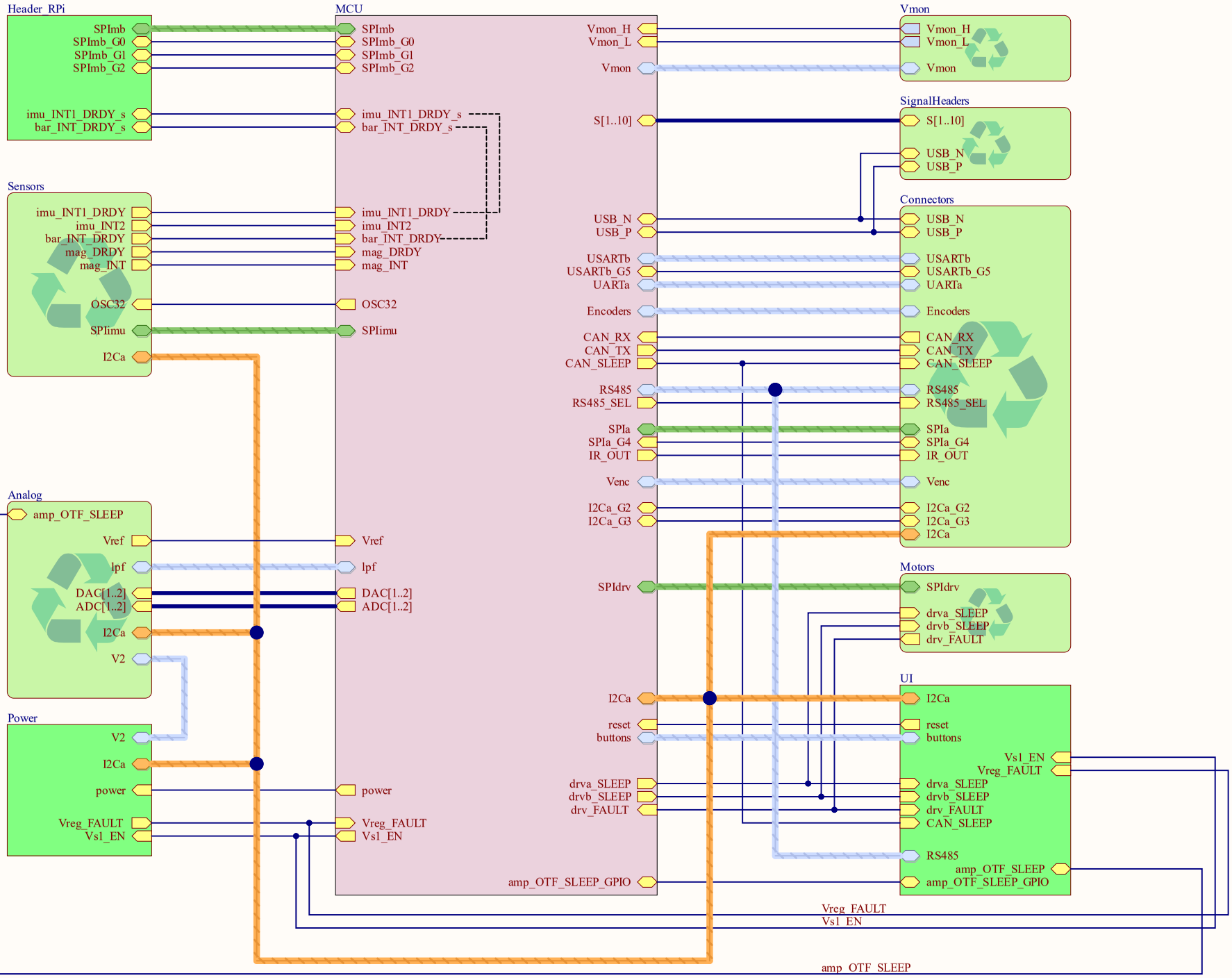
Note that the **Red** Beret is a **variant** of the **Raspberry** Beret, using the same PCB (and, thus, the same schematics), but with a lower-cost STM32G4, and several components flagged Do Not Populate (**DNP**). Eleven alternative schematic sheets are also defined, with different functionality implemented (generating Vmb = 12V instead of Vmb = 5V, etc), as listed here:

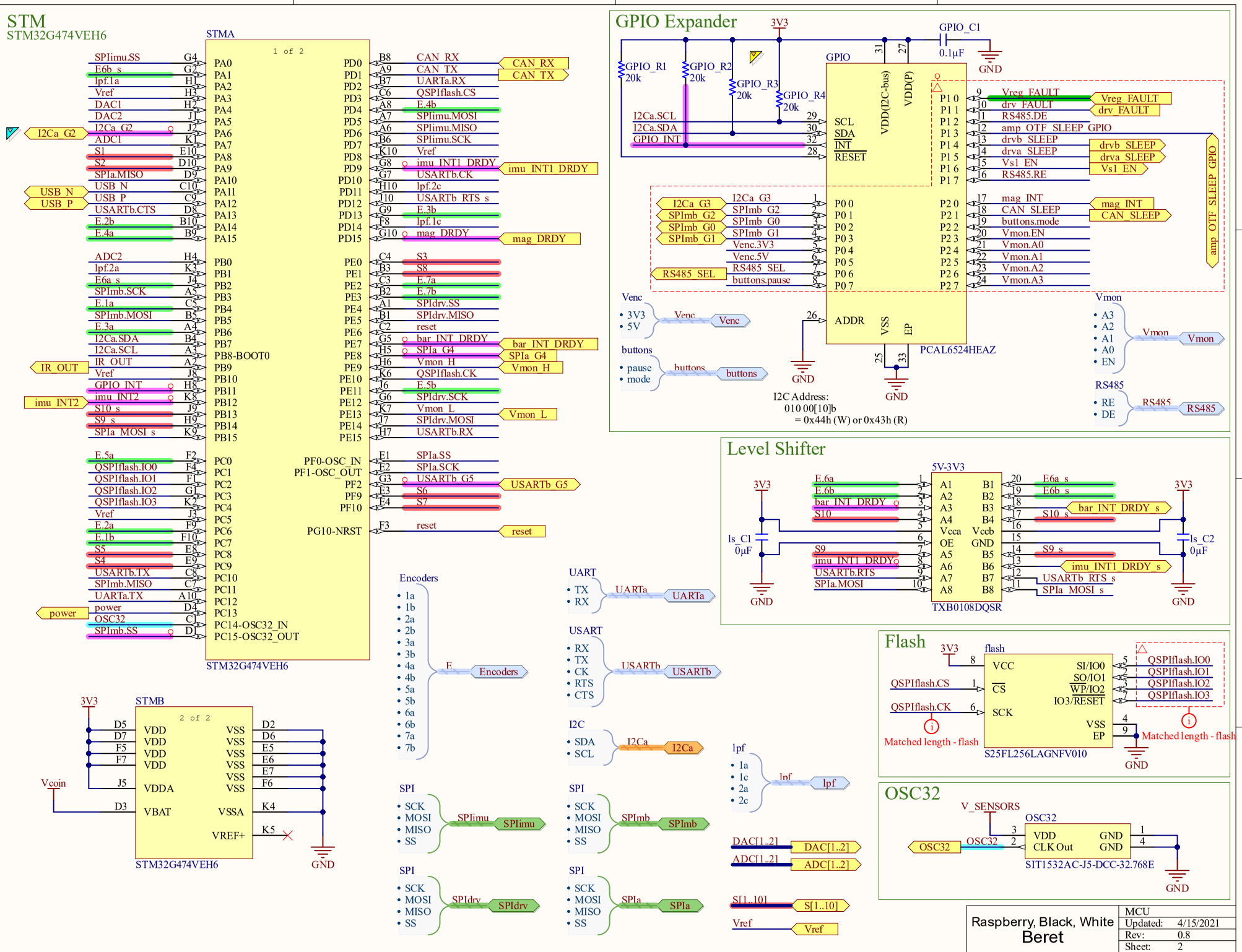
1a. **Master_Black**,
1b. **Master_White**,
1c. **Master_Green**,
1d. **Master_Blue**,
2a. **Power_12V_MB**,
2b. **Power_No_MB**,
5a. **MCU_No_GPIO_Expander**,
6a. **Connectors_Black**,
6b. **Connectors_Green**,
6c. **Connectors_Blue**,
10a. **Vmon_No_Balance**,
11a. **Header_96B**,
11b. **Header_BB**.

The other Berets are then defined using the following shared schematic sheet arrangement:

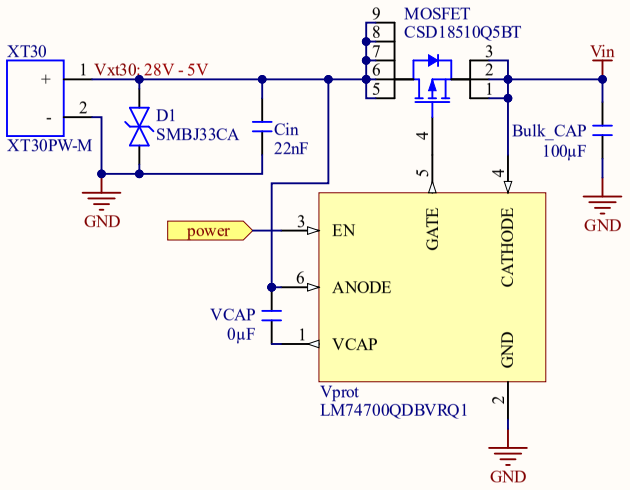
- Schematic sheets {1a, 2a, 3, 4, 5, 6a, 7, 8, 9, 10, 11a} define the **Black** Beret,
- Schematic sheets {1b, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11b} define the **White** Beret,
- Schematic sheets {1c, 2b, 4, 5a, 6b, 7, 8, 9, 10a} define the **Green** Beret.
- Schematic sheets {1d, 2b, 3, 4, 5a, 6c, 7, 8, 9, 10a} define the **Blue** Beret.

In this way, as the designs of the Berets are tweaked (changing resistor values in certain circuits, etc), the entire set of PCBs in the Beret family can more easily inherit all the updates made, and thus be kept in sync.

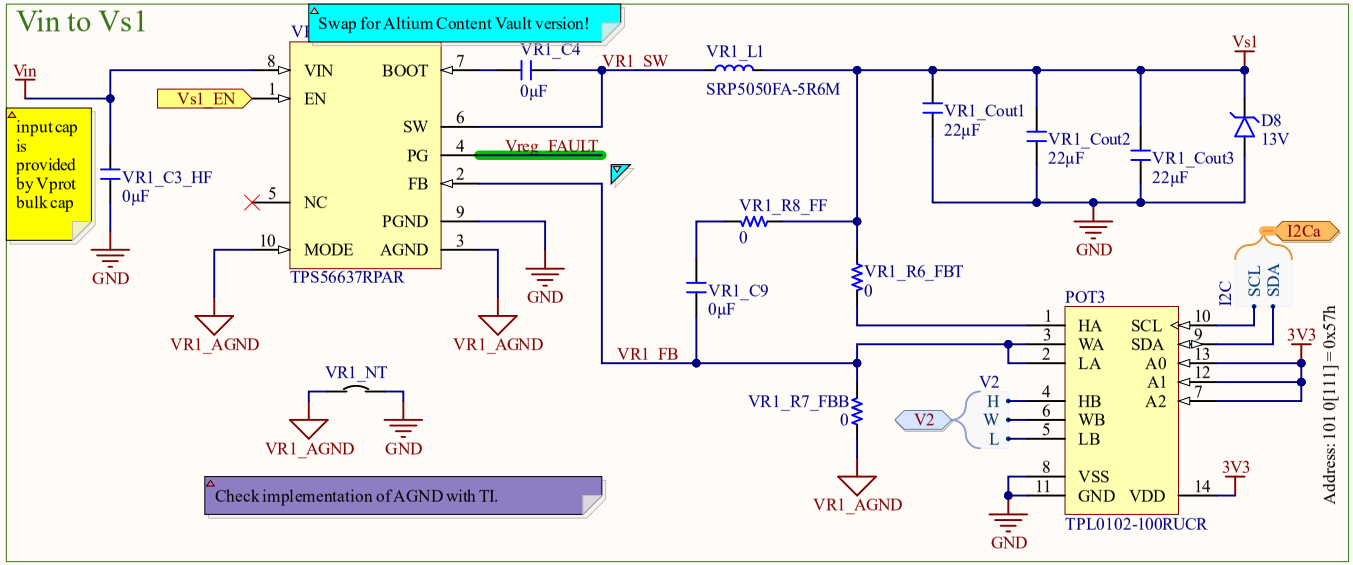

Header_RPi
SPImb
SPImb_G0
SPImb_G1
SPImb_G2
imu_INT1_DRDY_s
bar_INT_DRDY_s
Sensors
imu_INT1_DRDY
imu_INT2
bar_INT_DRDY
mag_DRDY
mag_INT
OSC32
SPIimu
I2Ca
Analog
amp_OTF_SLEEP
Vref
lpf
DAC[1..2]
ADC[1..2]
I2Ca
V2
Power
V2
I2Ca
power
Vreg_FAULT
Vs1_EN
MCU
SPImb
SPImb_G0
SPImb_G1
SPImb_G2
imu_INT1_DRDY_s
bar_INT_DRDY_s
imu_INT1_DRDY
imu_INT2
bar_INT_DRDY
mag_DRDY
mag_INT
OSC32
SPIimu
Vref
lpf
DAC[1..2]
ADC[1..2]
power
Vreg_FAULT
Vs1_EN
Vmon_H
Vmon_L
Vmon
S[1..10]
USB_N
USB_P
USARTb
USARTb_G5
UARTa
Encoders
CAN_RX
CAN_TX
CAN_SLEEP
RS485
RS485_SEL
SPIa
SPIa_G4
IR_OUT
Venc
I2Ca_G2
I2Ca_G3
SPIdrv
I2Ca
reset
buttons
drva_SLEEP
drvb_SLEEP
drv_FAULT
amp_OTF_SLEEP_GPIO
Vmon
Vmon_H
Vmon_L
Vmon
SignalHeaders
S[1..10]
USB_N
USB_P
Connectors
USB_N
USB_P
USARTb
USARTb_G5
UARTa
Encoders
CAN_RX
CAN_TX
CAN_SLEEP
RS485
RS485_SEL
SPIa
SPIa_G4
IR_OUT
Venc
I2Ca_G2
I2Ca_G3
I2Ca
Motors
SPIdrv
drva_SLEEP
drvb_SLEEP
drv_FAULT
UI
I2Ca
reset
buttons
Vs1_EN
Vreg_FAULT
drva_SLEEP
drvb_SLEEP
drv_FAULT
CAN_SLEEP
RS485
amp_OTF_SLEEP
amp_OTF_SLEEP_GPIO
Vreg_FAULT
Vs1_EN
amp_OTF_SLEEP
Raspberry
Beret
Master_Raspberry
Updated: 4/15/2021
Rev: 0.8
Sheet: 1


STM
STM32G474VEH6
STMA
1 of 2
STM32G474VEH6
GPIO Expander
PCAL6524HEAZ
I2C Address:
010 00[10]b
= 0x44h (W) or 0x43h (R)
Level Shifter
5V-3V3
TXB0108DQSR
Flash
flash
S25FL256LAGNFV010
Matched length - flash
OSC32
SIT1532AC-J5-DCC-32.768E
STMB
2 of 2
STM32G474VEH6
Encoders
UART
USART
I2C
SPI
SPI
SPI
lpf
DAC[1..2]
ADC[1..2]
S[1..10]
Vref
Raspberry, Black, White
Beret
MCU
Updated: 4/15/2021
Rev: 0.8
Sheet: 2

Power Input & Reverse Voltage Protection

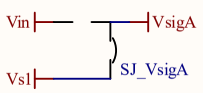


Vin to Vs1

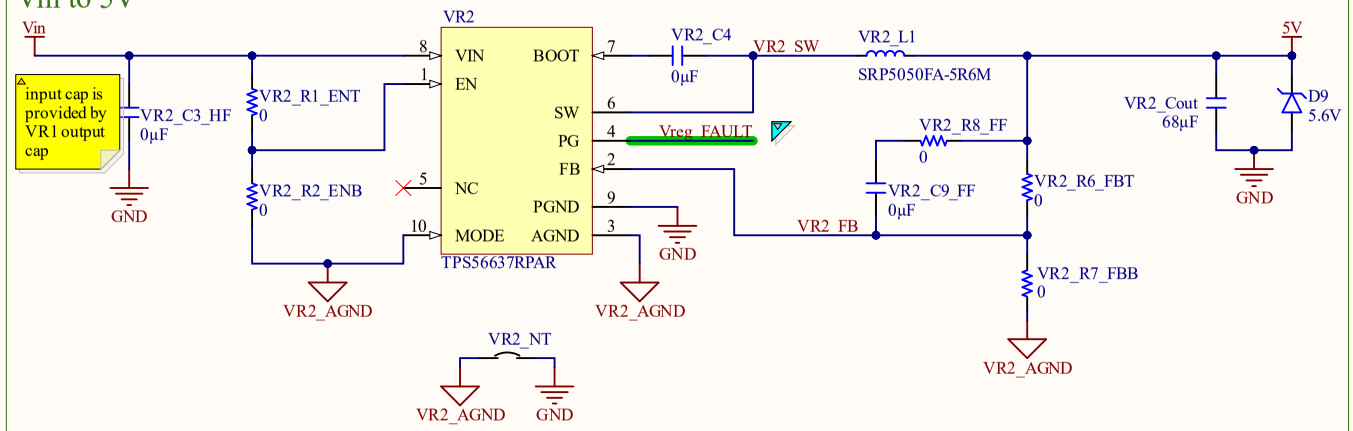


Signal Header A Power Selection

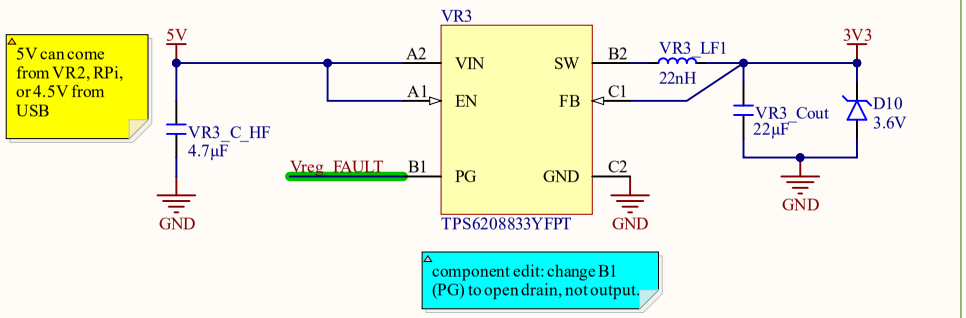
Signal Header A power pins are connected by default to Vs1 but can be connected to Vin instead by rotating the solder jumper 90 degrees.



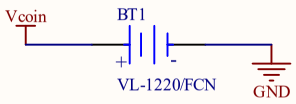
Vin to 5V



5V to 3.3V



Vcoin from Coin Cell

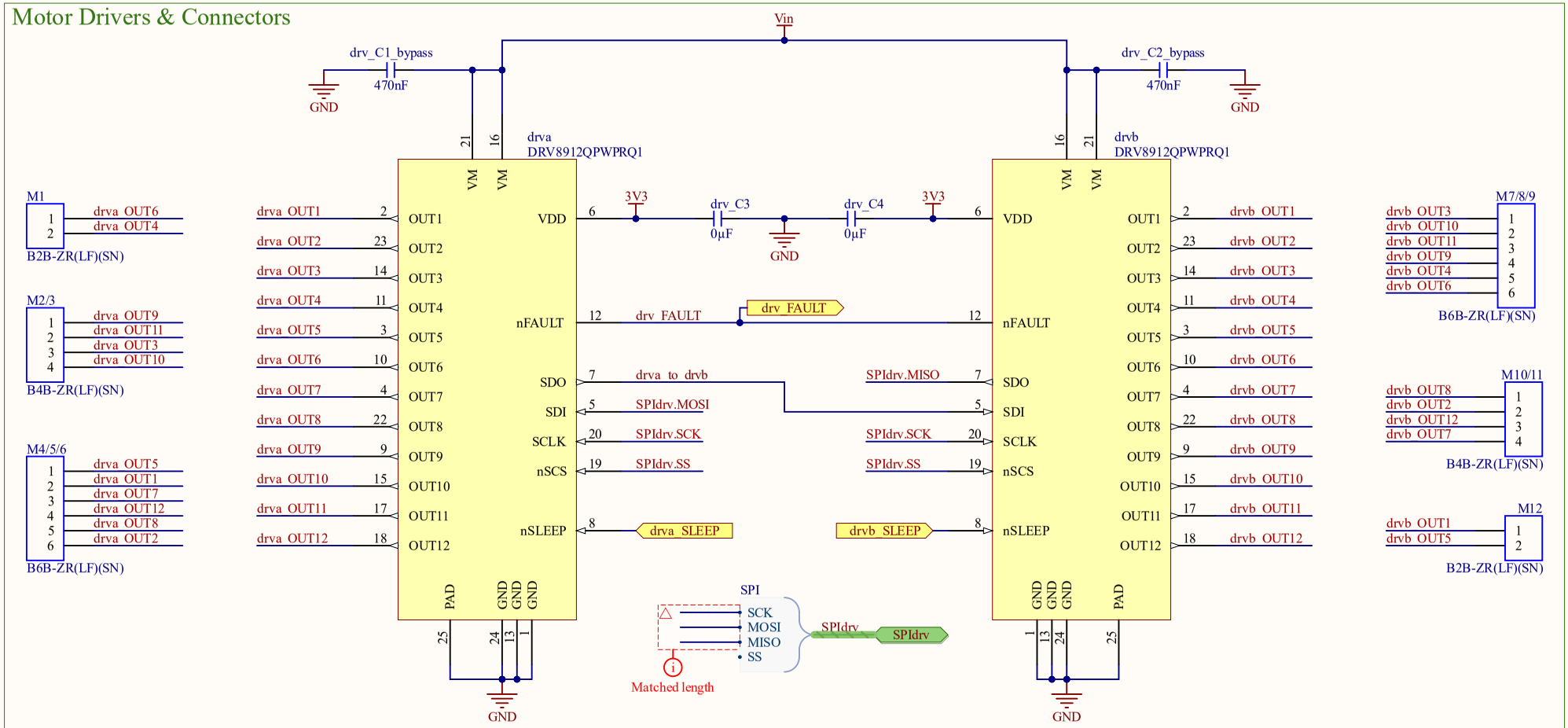


Vreg_FAULT

Raspberry, White
Beret

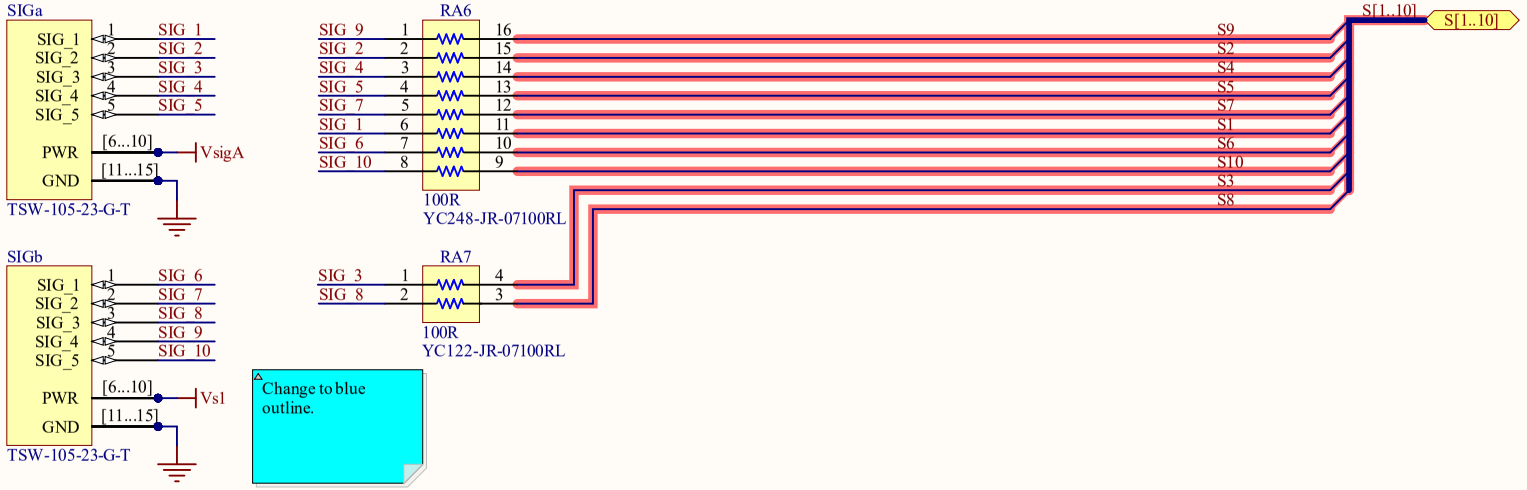
Power	
Updated:	4/15/2021
Rev:	0.8
Sheet:	3

Motor Drivers & Connectors

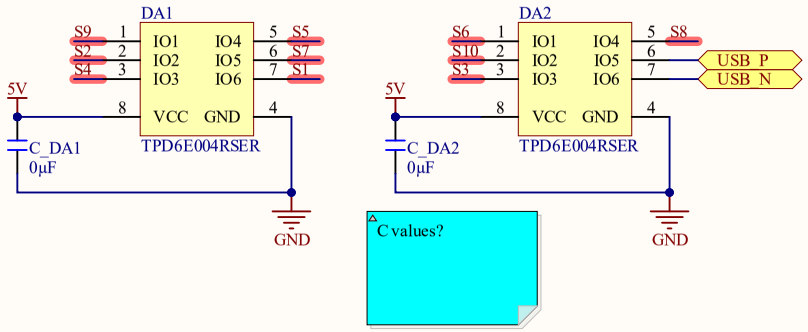


Rasp., Black, White Beret	Motors
	Updated: 4/15/2021
	Rev: 0.8
	Sheet: 4

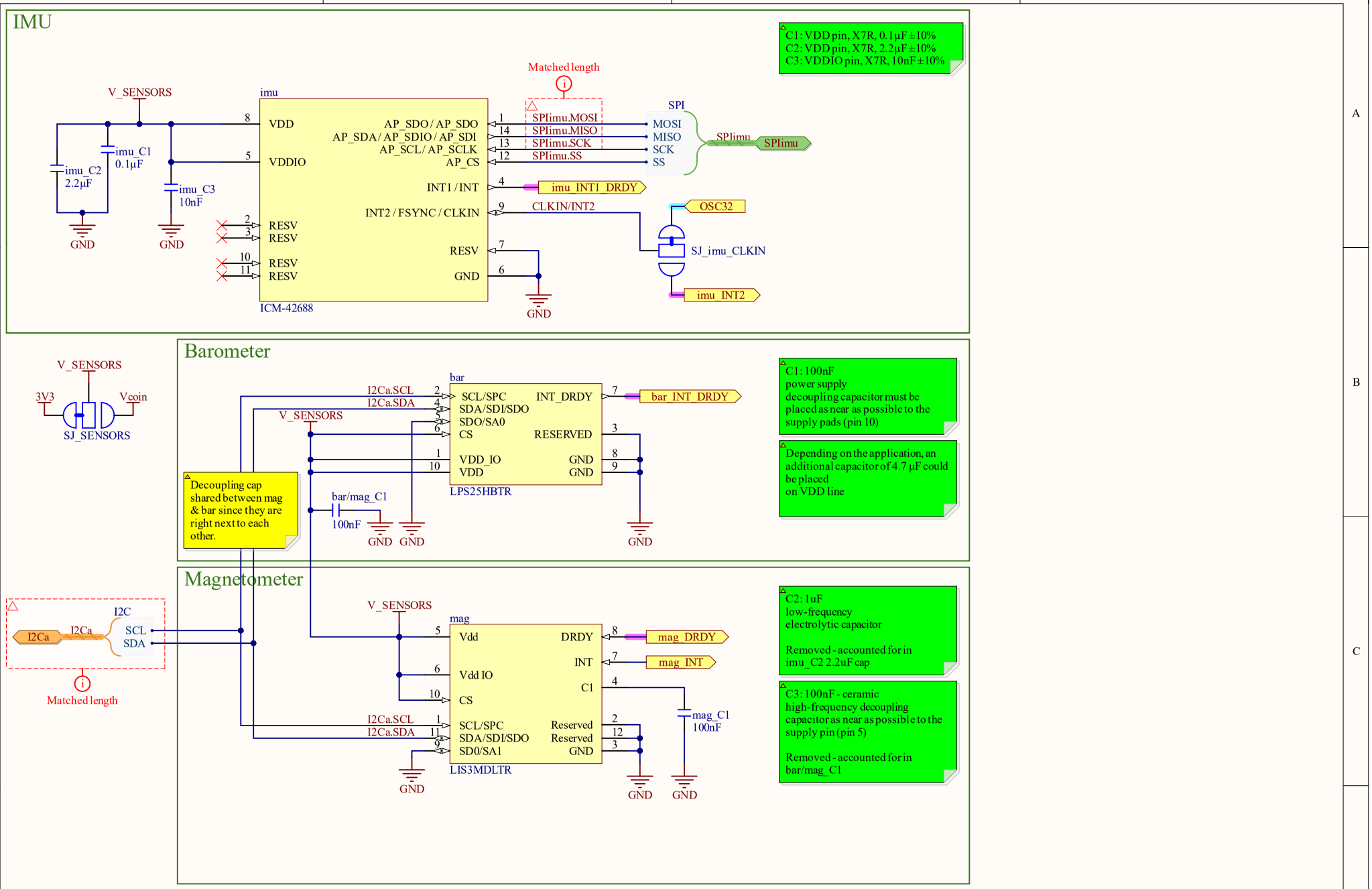
Signal Headers



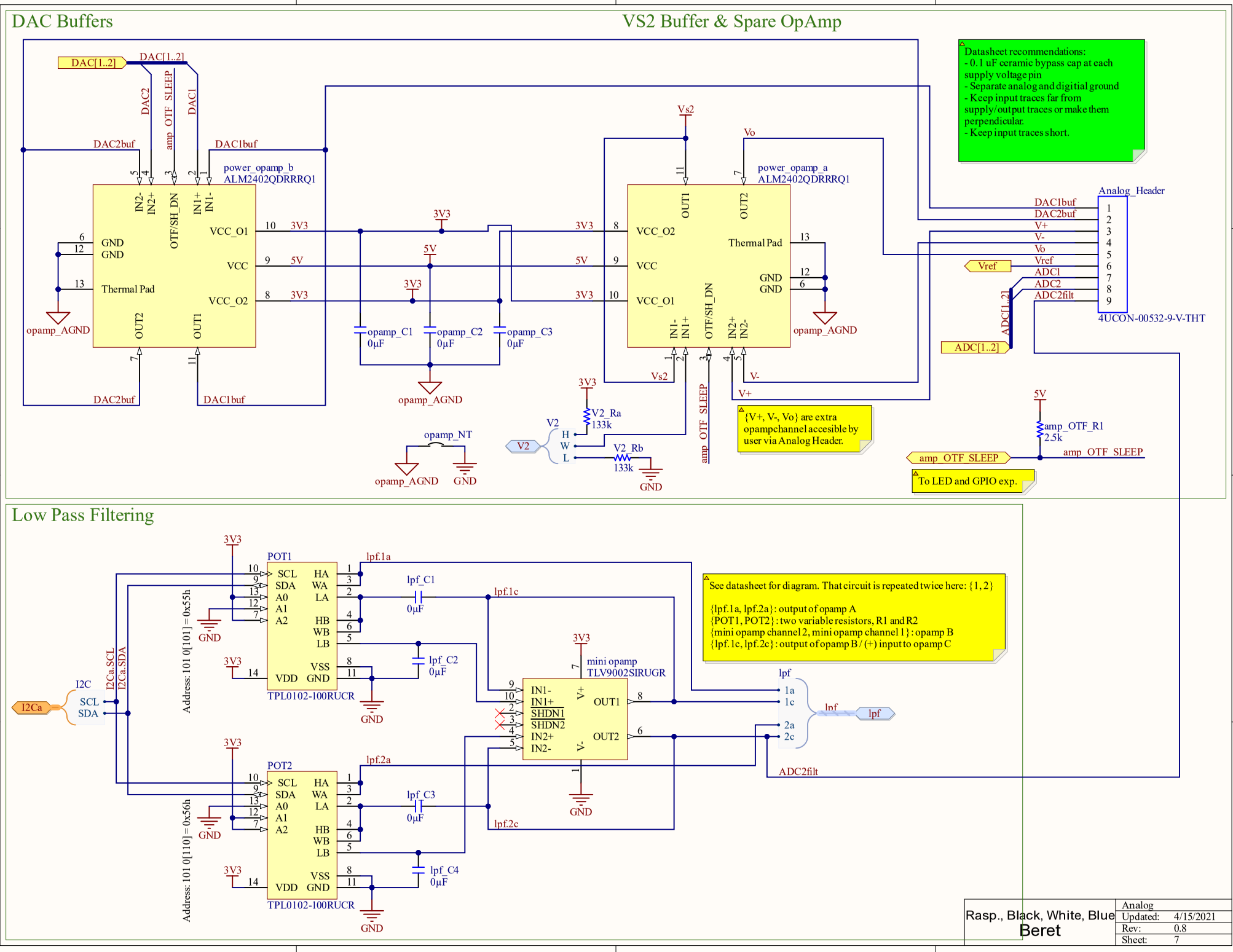
ESD Protection (Signal Headers & USB)

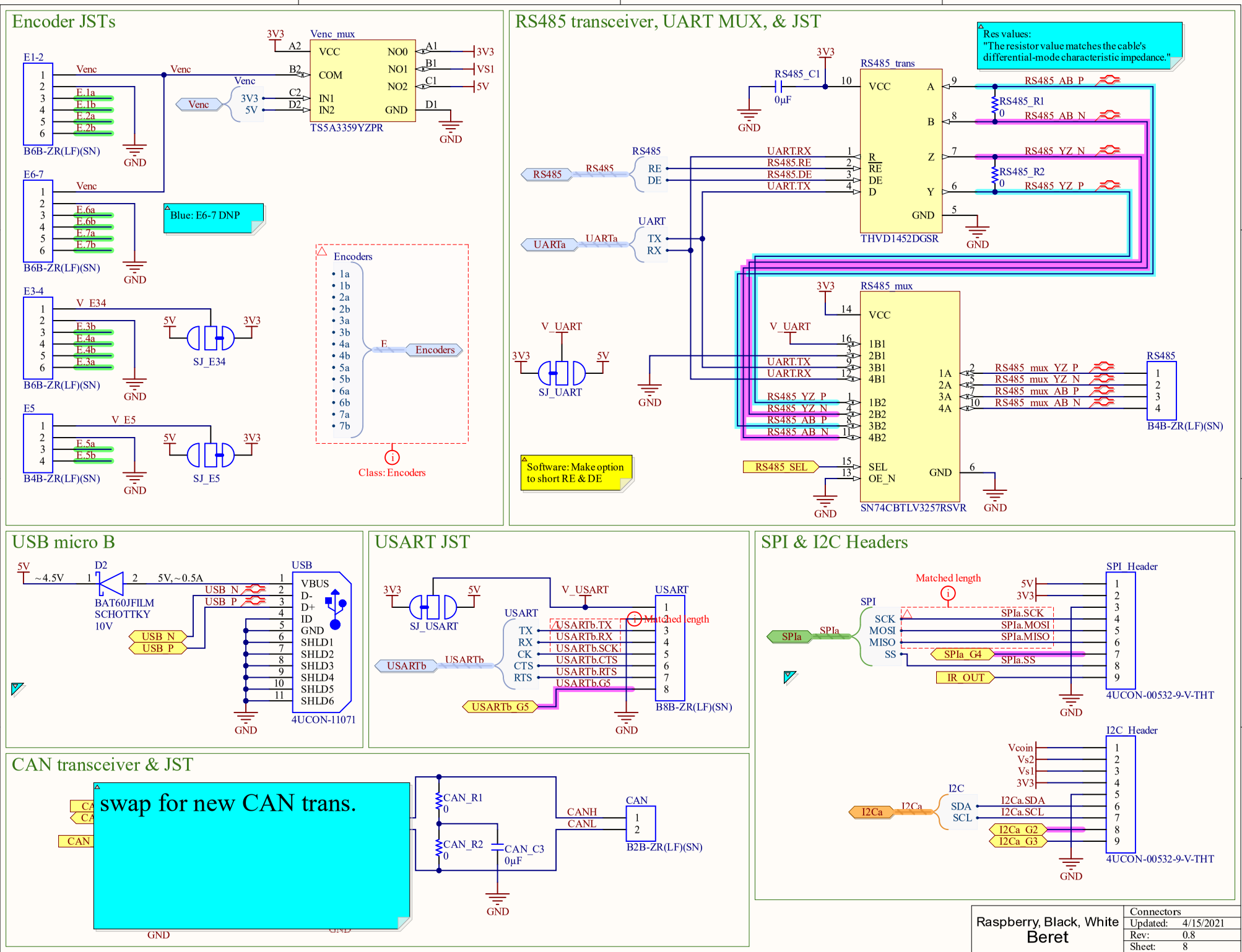


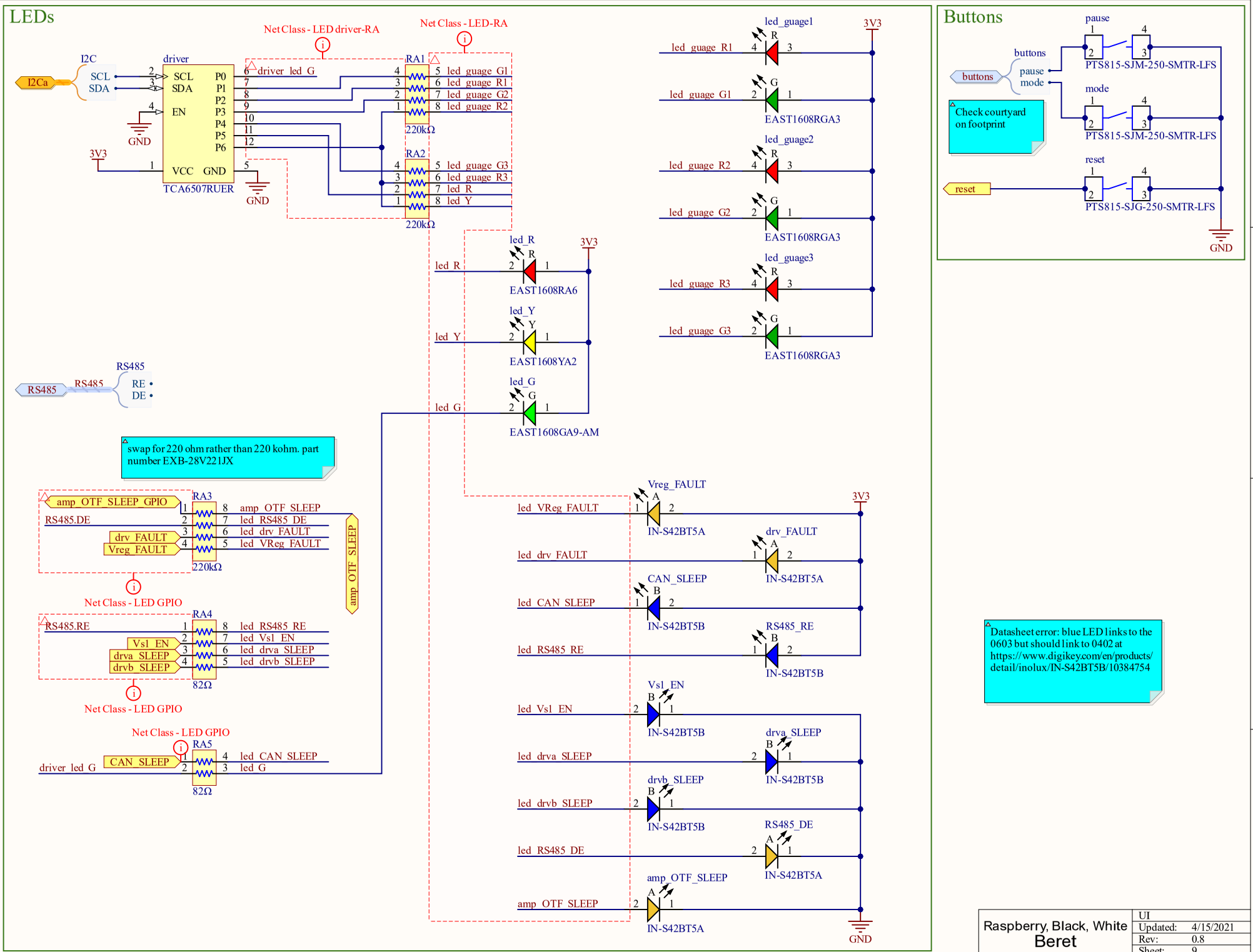
Rasp., Black, White, Blue Beret	Signal Headers
	Updated: 4/15/2021
	Rev: 0.8
	Sheet: 5



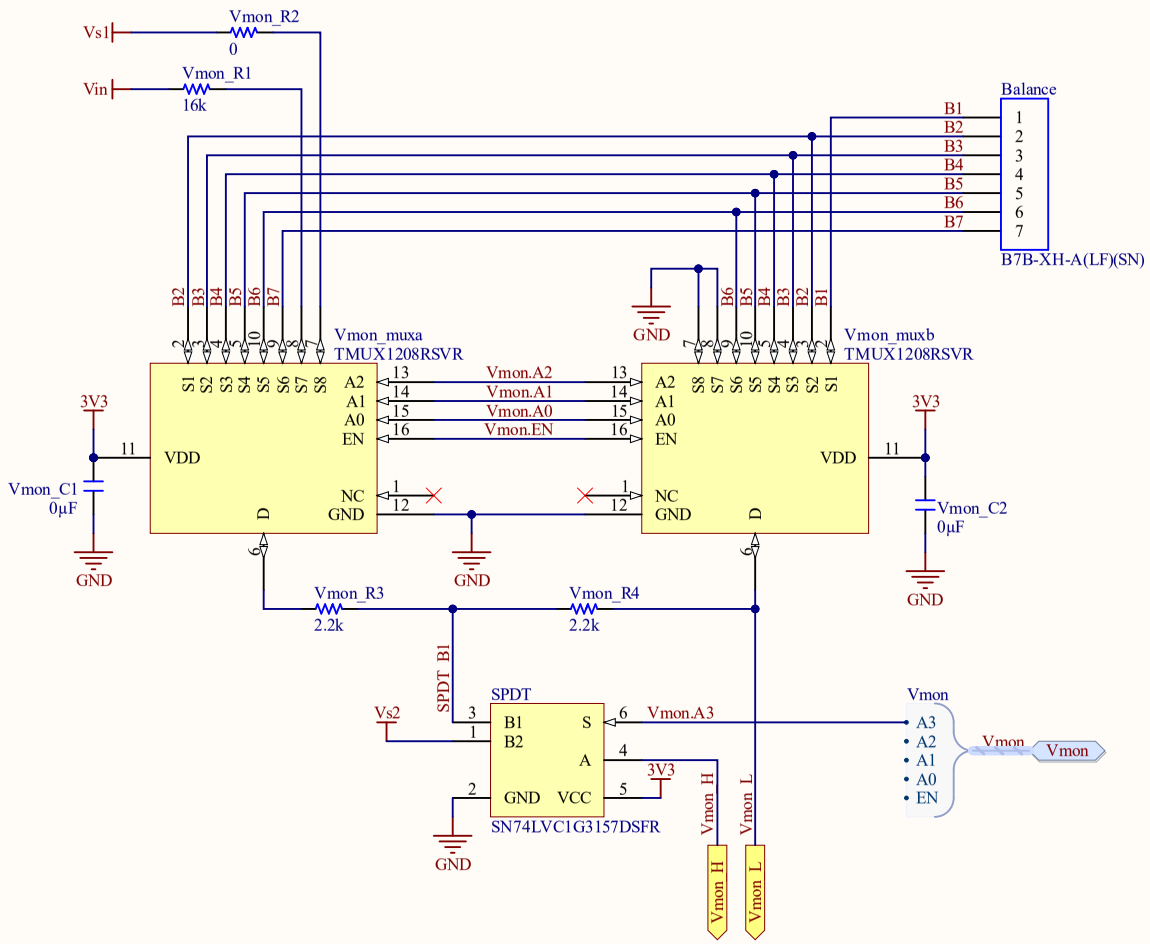
Rasp., Black, White, Blue Beret	Sensors
	Updated: 4/15/2021
	Rev: 0.8
	Sheet: 6


DAC Buffers
VS2 Buffer & Spare OpAmp
Datasheet recommendations:
- 0.1 uF ceramic bypass cap at each supply voltage pin
- Separate analog and digitial ground
- Keep input traces far from supply/output traces or make them perpendicular.
- Keep input traces short.
DAC[1..2]
DAC2
amp_OTF_SLEEP
DAC1
DAC2buf
DAC1buf
power_opamp_b
ALM2402QDRRRQ1
IN2-
IN2+
OTF/SH_DN
IN1+
IN1-
VCC_O1
VCC
VCC_O2
GND
GND
Thermal Pad
OUT2
OUT1
3V3
5V
opamp_AGND
opamp_C1
0µF
opamp_C2
0µF
opamp_C3
0µF
Vs2
Vo
power_opamp_a
ALM2402QDRRRQ1
Thermal Pad
V-
V+
{V+, V-, Vo} are extra opampchannel accesible by user via Analog Header.
Analog_Header
4UCON-00532-9-V-THT
Vref
ADC1
ADC2
ADC2filt
ADC[1..2]
ADC[1..2]
opamp_NT
V2
V2_Ra
133k
V2_Rb
133k
H
W
L
amp_OTF_R1
2.5k
amp_OTF_SLEEP
To LED and GPIO exp.
Low Pass Filtering
POT1
POT2
SCL
SDA
A0
A1
A2
HA
WA
LA
HB
WB
LB
VSS
VDD
GND
TPL0102-100RUCR
Address: 101 0[101] = 0x55h
Address: 101 0[110] = 0x56h
lpf.1a
lpf.1c
lpf.2a
lpf.2c
lpf_C1
lpf_C2
lpf_C3
lpf_C4
0µF
I2C
I2Ca
I2Ca.SCL
I2Ca.SDA
mini opamp
TLV9002SIRUGR
IN1-
IN1+
SHDN1
SHDN2
IN2+
IN2-
V+
V-
OUT1
OUT2
lpf
1a
1c
2a
2c
ADC2filt
See datasheet for diagram. That circuit is repeated twice here: {1, 2}
{lpf.1a, lpf.2a}: output of opamp A
{POT1, POT2}: two variable resistors, R1 and R2
{mini opamp channel2, mini opamp channel1}: opamp B
{lpf.1c, lpf.2c}: output of opamp B / (+) input to opamp C
Rasp., Black, White, Blue Beret
Analog
Updated: 4/15/2021
Rev: 0.8
Sheet: 7


Encoder JSTs
E1-2
B6B-ZR(LF)(SN)
E6-7
B6B-ZR(LF)(SN)
E3-4
B6B-ZR(LF)(SN)
E5
B4B-ZR(LF)(SN)
Venc
Venc_mux
TS5A3359YZPR
Blue: E6-7 DNP
SJ_E34
SJ_E5
Encoders
Class: Encoders
RS485 transceiver, UART MUX, & JST
Res values:
"The resistor value matches the cable's differential-mode characteristic impedance."
RS485_trans
THVD1452DGSR
RS485_mux
SN74CBTLV3257RSVR
SJ_UART
Software: Make option to short RE & DE
B4B-ZR(LF)(SN)
USB micro B
BAT60JFILM SCHOTTKY 10V
4UCON-11071
USART JST
SJ_USART
B8B-ZR(LF)(SN)
SPI & I2C Headers
Matched length
SPI_Header
4UCON-00532-9-V-THT
I2C_Header
4UCON-00532-9-V-THT
CAN transceiver & JST
swap for new CAN trans.
B2B-ZR(LF)(SN)
Raspberry, Black, White
Beret
Connectors
Updated: 4/15/2021
Rev: 0.8
Sheet: 8

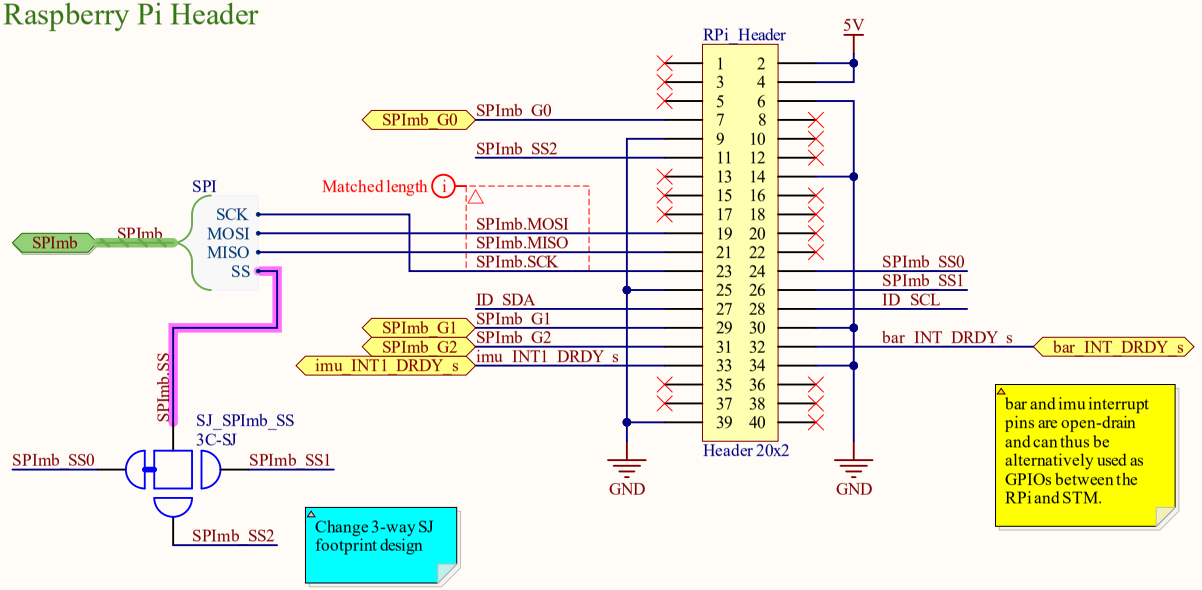

LEDs
Net Class - LED driver-RA
Net Class - LED-RA
I2C
driver
I2Ca
SCL
SDA
EN
P0
P1
P2
P3
P4
P5
P6
VCC
GND
TCA6507RUER
3V3
driver_led_G
led_guage_G1
led_guage_R1
led_guage_G2
led_guage_R2
led_guage_G3
led_guage_R3
led_R
led_Y
led_G
RA1
RA2
220kΩ
led_guage1
led_guage2
led_guage3
EAST1608RGA3
EAST1608RA6
EAST1608YA2
EAST1608GA9-AM
RS485
RE
DE
swap for 220 ohm rather than 220 kohm. part number EXB-28V221JX
RA3
amp_OTF_SLEEP_GPIO
amp_OTF_SLEEP
RS485.DE
led_RS485_DE
drv_FAULT
led_drv_FAULT
Vreg_FAULT
led_VReg_FAULT
Net Class - LED GPIO
RA4
RS485.RE
led_RS485_RE
Vs1_EN
led_Vs1_EN
drva_SLEEP
led_drva_SLEEP
drvb_SLEEP
led_drvb_SLEEP
82Ω
RA5
CAN_SLEEP
led_CAN_SLEEP
driver_led_G
led_G
Vreg_FAULT
IN-S42BT5A
CAN_SLEEP
IN-S42BT5B
RS485_RE
Vs1_EN
drva_SLEEP
drvb_SLEEP
RS485_DE
amp_OTF_SLEEP
Buttons
buttons
pause
mode
reset
PTS815-SJM-250-SMTR-LFS
PTS815-SJG-250-SMTR-LFS
Check courtyard on footprint
Datasheet error: blue LED links to the 0603 but should link to 0402 at https://www.digikey.com/en/products/detail/inolux/IN-S42BT5B/10384754
Raspberry, Black, White Beret
UI
Updated: 4/15/2021
Rev: 0.8
Sheet: 9

Voltage Monitoring (Vmon)

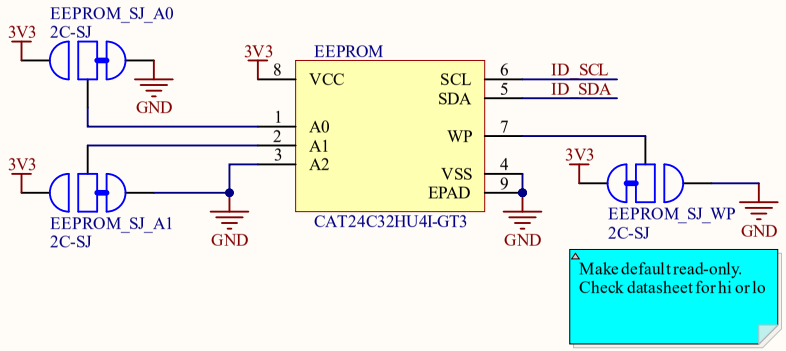


Rasp., Black, White Beret	Voltage Monitoring
	Updated: 4/15/2021
	Rev: 0.8
	Sheet: 10

Raspberry Pi Header



EEPROM



Raspberry Beret	Header_RPi
	Updated: 4/15/2021
	Rev: 0.8
	Sheet: 11

Part II

Theoretical Foundations

Chapter 6

Kinematics & Dynamics

The complete representation of a system's configuration, and the derivation of the full equations of motion governing how such a configuration evolves in time as a result of its initial motion, its internal connectivity, its contact with the ground and other stationary structures, and the applied forces and torques, is in general an involved and delicate subject. As a gentle introduction to this subject, in §6.1, we develop the equations of motion of some very simple dynamic systems that are used as examples in the remainder of this text. However, the reader is advised to not let the simple academic examples of §6.1 lull one into a false sense of complacency regarding this important and difficult subject, which is introduced only briefly in the remainder of this chapter.

The general problem of defining a system's configuration, and identifying the equations of motion governing how such a configuration evolves in time, is generally divided into two distinct subproblems. **Kinematics** is the study of properties of motion, such as the inherent relationships between linear & angular positions, velocities, and accelerations in different reference frames, and the constraints imposed by various types of contact between solid bodies. **Dynamics** is the study of the actual equations governing motion, including the effects of the forces and torques applied to a system. This chapter briefly considers both subjects. The presentation is inherently 3D, with essentially 2D problems treated as special cases.

We begin with the notion of a **particle**: that is, a body with **mass** whose dimensions are sufficiently small (compared with the rest of the system being considered) that its rotation may be neglected when modeling its motion. In this case, the problem of kinematics is essentially trivial: in Cartesian coördinates in 3D, the **position** of a particle is denoted by a vector[1] $\vec{r} \in \mathbb{R}^3$, its **velocity** $\vec{v} = d\vec{r}/dt$, and its **acceleration**[2] $\vec{a} = d\vec{v}/dt = d^2\vec{r}/dt^2$. The position of N particles in 3D is thus specified by $s = 3N$ **coördinates** (a.k.a. **degrees of freedom**). Any s quantities q_i for $i = 1, \ldots, s$ (collectively, $\mathbf{q}$) which uniquely specify the configuration of a system (e.g., in coördinate systems other than Cartesian) are referred to as **generalized coördinates**, and their derivatives $\dot{q}_i$ **generalized velocities**.

In §6.2, the dynamics of a system of N interacting particles is derived, starting from two basic axioms:

A. If the (Cartesian or generalized) coördinates and velocities (collectively, the **state**) of a mechanical system is specified, its subsequent motion can be calculated; that is, the accelerations $\ddot{q}_i$ may be determined uniquely from the coördinates q_i and the velocities $\dot{q}_i$, and thus the system may be marched in time with any of a variety of ODE time marching methods, such as the RK4 method discussed briefly in §7.

B. The motion of a mechanical system is characterized by a **principle of least action** (a.k.a. **Hamilton's principle**), in which an integral of some function of the coördinates and velocities is minimized.

[1] To disambiguate, we denote vectors defined in $\mathbb{R}^3$ with an arrow over the symbol, and more general vectors (including quaternions, introduced in §6.3.2.2) with boldface. Also, we sometimes denote differentiation with respect to time with a dot, e.g., $\dot{\mathbf{q}} = d\mathbf{q}/dt$.

[2] The quantity $d^3\vec{r}/dt^3$ is called the **jerk**, and the quantity $d^4\vec{r}/dt^4$ is sometimes called the **jounce**; alternatively, the quantities $d^4\vec{r}/dt^4$, $d^5\vec{r}/dt^5$, and $d^6\vec{r}/dt^6$ are sometimes humorously referred to as **snap**, **crackle**, and **pop**.

From these axioms, via a formulation known as **Lagrangian mechanics**, the laws of **classical mechanics** are derived; note that classical mechanics neglects the **relativistic effects** that arise when the characteristic velocities are a significant fraction of the speed of light[3]. The consequences of homogeneity and isotropy of the equations of motion, in both space and time, are also considered, leading to the conservation of momentum, angular momentum, and energy, and to the reversibility of trajectories in the absence of frictional losses.

In §6.3, we discuss the notion of a **solid body**, first by approximation as several particles rigidly connected by massless rods, then by passing to the limit as the number of particles approaches infinity. In both cases, it is shown that the dynamic properties of a solid body is characterized completely by its total mass and its inertial tensor, both of which are easy to compute. The configuration of a solid body is specified uniquely by the position of its center of mass together with its orientation, the latter of which may be described as a rotation of the body, as specified by three degrees of freedom, from some reference orientation. There are a number of different ways to describe the orientation of a body and how it changes in time; this subject is somewhat delicate, and requires some care.

Once the notions of a solid body and its orientation and rotation are at hand, the equations governing the dynamics of solid bodies are developed in §6.4, building from the dynamics of particles discussed previously. This development includes the derivation of the equations of motion themselves, using various descriptions of rotations and accounting for various types of contact, as well as a description the conservation of momentum, angular momentum, and energy, and the consequences of these conservation properties.

6.1 The equations of motion of some simple physical systems

We now introduce a few simple mechanical, fluid, chemical, automotive, structural, and aerospace systems, and develop the low-dimensional ODEs governing their dynamics. These canonical systems are considered further, in the analysis and control settings, in the remainder of this text, and in the companion volume *NR*.

Example 6.1 A (linear) mass/spring/damper system

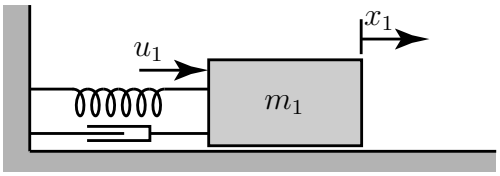


Figure 6.1: The single mass/spring/damper system set up in Example 6.1.

Recalling **Newton's second law**, $f = ma$ where m is the mass of the body, f is the force applied to the body, and a is the resulting linear acceleration of the body, the motion of the simple **mass/spring/damper**[4] system illustrated in Figure 6.1 is governed by

$$m_1 \frac{d^2 x_1}{dt^2} = u_1 - k\, x_1 - c \frac{dx_1}{dt}, \tag{6.1}$$

where x_1 is the deflection of the mass from its rest position, u_1 is the applied force, $\{m_1, k, c\}$ are constants, and the spring and damper have been modeled as linear in the deflection and velocity, respectively.

Identifying a SISO model by taking the **output** of the system as $y = x_1$, the **input** to the system as $u = u_1$, and defining $a_1 = c/m_1$, $a_0 = k/m_1$, and $b_0 = 1/m_1$, we may rewrite (6.1) in a standard **input/output ODE**

[3] Such relativistic effects occur, e.g., in the everyday setting of an electron moving along a wire, giving rise to the **magnetic field**.

[4] A.k.a. **dashpot** or **shock absorber**. Note that shock absorbers often exhibit significant nonlinear characteristics for large or fast motions, in which case the linear model used here should be considered as only approximate.

form as

$$\frac{d^2y}{dt^2} + a_1\frac{dy}{dt} + a_0\, y = b_0\, u. \tag{6.2}$$

If the spring and damper are removed ($k = c = 0$), the system reduces to the **double integrator** $d^2y/dt^2 = b_0\, u$.

Recalling **Newton's second law of rotation**, $\tau = I\alpha$ where I is the moment of inertia of the about the axis of rotation, τ is the applied torque, and α is the resulting angular acceleration, analogous rotational systems are easily identified that are governed by the same ODEs as those identified above. Hard disk read/write head/arm assemblies are an important engineering example system that fit such a model. △

Example 6.2 A (linear) cascade mass/spring system with viscous friction

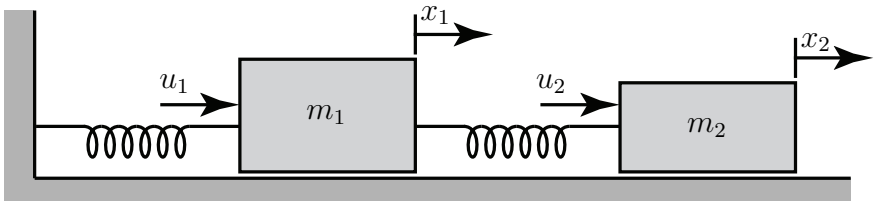


Figure 6.2: The cascade mass/spring system with viscous friction set up in Example 6.2.

The equations governing the motion of each mass in the cascade system illustrated in Figure 6.2 also follow immediately from Newton's second law:

$$m_1\frac{d^2x_1}{dt^2} = u_1 - k_1x_1 + k_2(x_2 - x_1) - \mu_1 m_1 g\,\frac{dx_1}{dt}, \tag{6.3a}$$

$$m_2\frac{d^2x_2}{dt^2} = u_2 \qquad - k_2(x_2 - x_1) - \mu_2 m_2 g\,\frac{dx_2}{dt}, \tag{6.3b}$$

where x_1 and u_1 are the deflection of and applied force on the first mass, x_2 and u_2 are the deflection of and applied force on the second mass, $\{m_1, m_2, k_1, k_2, \mu_1, \mu_2\}$ are constants, and $g = 9.8$ m/sec^2. Note that the linear damping in this case is modeled as arising from the **viscous friction** between the blocks and the horizontal surface, assuming this interface is lubricated; in this case, the friction force is accurately modeled as proportional to both the weight of the respective block[5] and the velocity of the relative motion at the interface (i.e., roughly independent of the contact area!), and is of a sign that opposes this motion.

To manipulate a set of ODEs like (6.3) algebraically, it is convenient to first express it in operator form:

$$\Big[m_1\frac{d^2}{dt^2} + \mu_1 m_1 g\frac{d}{dt} + k_1 + k_2\Big]x_1 + \Big[-k_2\Big]x_2 = u_1, \quad \Rightarrow \quad \mathcal{L}_1x_1 + \mathcal{L}_2x_2 = u_1, \tag{6.4a}$$

$$\Big[-k_2\Big]x_1 + \Big[m_2\frac{d^2}{dt^2} + \mu_2 m_2 g\frac{d}{dt} + k_2\Big]x_2 = u_2, \quad \Rightarrow \quad \mathcal{L}_3x_1 + \mathcal{L}_4x_2 = u_2. \tag{6.4b}$$

Identifying a SISO model by taking, for example, the **output** of the system as $y = x_2$ and the **input** to the system as $u = u_1$ (and, for the moment, taking $u_2 = 0$), we may thus rewrite (6.3) by subtracting $\mathcal{L}_3$ times (6.4a) from $\mathcal{L}_1$ times (6.4b), noting that, e.g., $\mathcal{L}_1\mathcal{L}_3x_1 = \mathcal{L}_3\mathcal{L}_1x_1$, thus leading again to the standard ODE form:

$$(\mathcal{L}_1\mathcal{L}_4 - \mathcal{L}_3\mathcal{L}_2)x_2 = -\mathcal{L}_3u_1 \quad \Rightarrow \quad \frac{d^4y}{dt^4} + a_3\frac{d^3y}{dt^3} + a_2\frac{d^2y}{dt^2} + a_1\frac{dy}{dt} + a_0\, y = b_0\, u \tag{6.5}$$

where $a_3 = (\mu_1 + \mu_2)g$, $a_2 = k_2/m_2 + (k_1 + k_2)/m_1 + \mu_1\mu_2\, g^2$, $a_1 = \mu_1 g\, k_2/m_2 + \mu_2 g\,(k_1 + k_2)/m_1$, $a_0 = k_1k_2/(m_1m_2)$, and $b_0 = k_2/(m_1m_2)$.

Finally, as in (6.2), note that there are more derivatives on the output y than there are on the input u in the SISO ODE model given in (6.5); this property is essentially ubiquitous in mechanical systems with inertia, and is discussed further in §8.2.3.1. △

[5] Or, the component of this weight normal to the interface if the interface is at an incline.

Example 6.3 A (nonlinear) mass/elastic-conveyer-belt system with dry friction

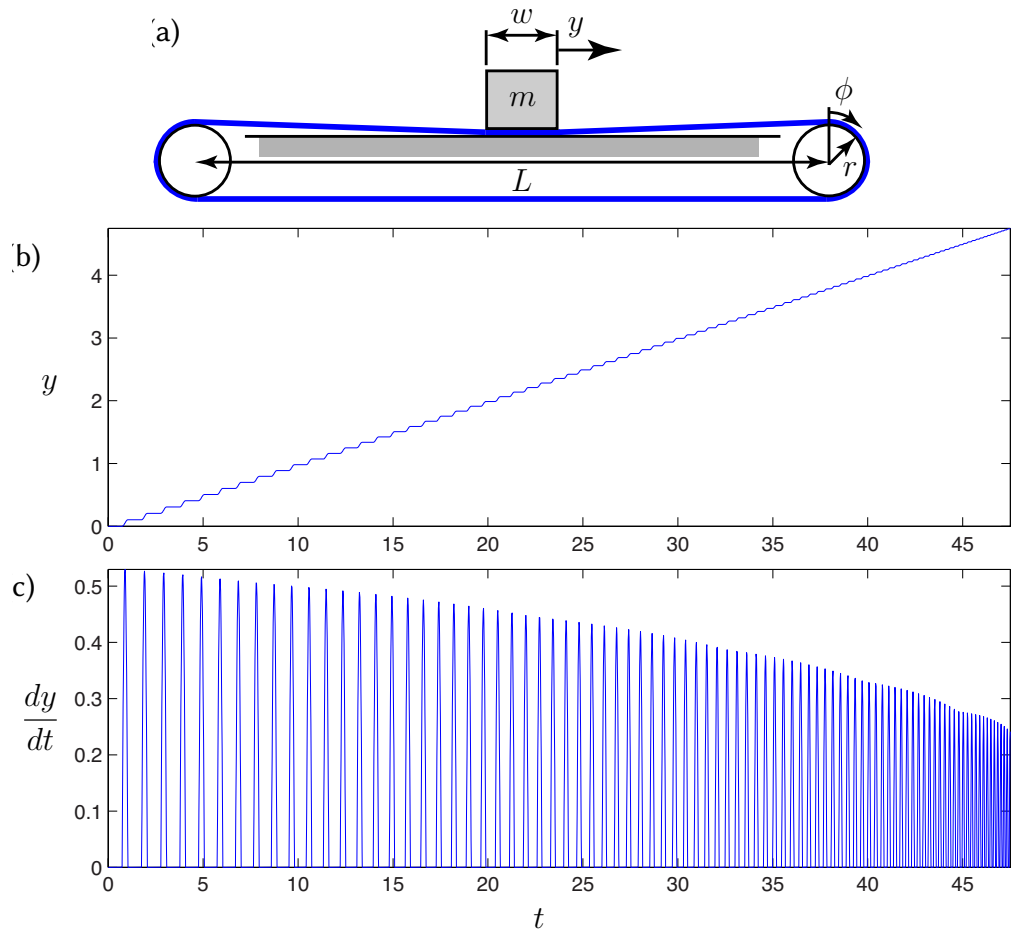


Figure 6.3: (a) The elastic conveyer belt system, with dry friction, in Example 6.3. To simplify the analysis, we assume that $w \ll L$ and $r \ll L$. The (b) position and (c) velocity components of the step response of this system as a function of time are also shown; note the **stick/slip** behavior that results from the nonlinear friction model. Note also that the frequency of the resulting **jerks** increases as the mass approaches the end of the belt.

We now consider the more problematical elastic conveyer belt system illustrated in Figure 6.3a.

In this system, the sections of the pretensioned elastic belt (that is, the "springs") acting to pull the mass to the left and right are effectively changing in "length" as the driven pulley, on the right end of the system, drags the mass across the table to the right. The modeling of the force applied by the belt thus requires some care. We first assume that the belt does not slip on the driven pulley and that the mass doesn't slip on the belt, though the idler pulley on the left end of the system is free to rotate and the belt, though it makes contact with the table under the mass, and slides (with friction in the region of this contact patch) across the table. We will refer to the "length" of the portion of the belt tending to pull the mass to the right as the distance from the mass directly to the driven pulley, $\ell_1 = L/2 - y$, and the "length" of the portion of the belt tending to pull the mass to the left as the distance from the mass to the driven pulley around the idler, $\ell_2 = 3L/2 + y$. For any given amount of rotation of the driven pulley $\phi(t)$, measured in radians, there is a corresponding nominal position of the mass $\overline{y}(t)$ at which the force applied by the pretensioned belt to the left and right sides of the mass is equal. The actual position of the mass, $y(t)$, may thus be written

$$y = \overline{y} + y' \quad \text{where} \quad \overline{y} = r\,\phi, \tag{6.6a}$$

where $y'(t)$ denotes the (small) perturbation of the mass from the nominal position $\overline{y}(t)$. The total force applied

by the belt to the mass, which opposes the perturbation y', is then given by

$$f_{\text{belt}} = -y'(k_0/\ell_1 + k_0/\ell_2) \tag{6.6b}$$

where k_0, the spring constant per unit length of the belt, is a constant.

The friction force f_{friction} caused by the dry contact of the portion of the belt under the mass with the table is accurately modeled in two parts. If you have ever tried to push a heavy object without wheels across a level surface[6], you probably recall that it takes more force to get the object moving than it takes to keep it moving, and that once the object is moving, the force required to keep it moving is approximately independent of the speed at which it is moving (this latter property is know as **Coulomb's law**). That is,

• if the velocity of the mass is zero (i.e., the system is **stuck**), the magnitude of the friction force f_{friction} precisely matches the force applied to the mass by the belt, with a sign that opposes the force applied by the belt, up to a maximum absolute value of $\mu_s mg$, where mg is the weight of the mass[7,8], whereas

• if the velocity of the mass is nonzero (i.e., the system is **unstuck**), the magnitude of the friction force f_{friction} is $\mu_k mg$, with a sign that opposes the motion of the belt (and the mass that sits thereon), where μ_s is the **coefficient of static friction** and μ_k is the **coefficient of kinetic friction**; thus,

$$f_{\text{friction}} = \begin{cases} -\min(|f_{\text{belt}}|, \mu_s\, m\, g)\, \text{sgn}\ (f_{\text{belt}}) & \text{if } dy/dt = 0, \\ -\mu_k\, m\, g\, \text{sgn}\ (dy/dt) & \text{if } dy/dt \neq 0. \end{cases} \tag{6.6c}$$

Typically, $\mu_s > \mu_k$; representative values[9] of these two coefficients for a rubber belt and a metal surface are $\mu_s \approx 1.0$ and $\mu_k \approx 0.5$.

The motion of the mass is thus governed by

$$m\frac{d^2y}{dt^2} = f_{\text{belt}} + f_{\text{friction}}, \tag{6.6d}$$

where f_{belt} and f_{friction} are given above, with $\overline{y} = y' = \phi = 0$ corresponding to the mass at the center of the conveyer belt with no net force applied by the belt to the mass.

The motion that the above system exhibits is illustrated in Figure 6.3b-c; for the purpose of this numerical simulation, we take $m = 1$ kg, $r = 0.1$ m, $k_0 = 500$ N, and $L = 10$ m. This system may be simulated accurately using, e.g., the standard RK4 technique (see §7); however, care must be taken in order to switch accurately between the "stuck" and "unstuck" conditions. The code used to perform the simulation illustrated in Figure 6.3b-c is available as RR_Example_Conveyer_Belt.m.

The **stick/slip** behavior illustrated in Figure 6.3b-c is a nonlinear phenomenon that defies any reasonably accurate linear approximation. Some physical systems are like this, with systems exhibiting dry friction being particularly "sticky" to deal with. Fortunately, many[10] "highly nonlinear"[11] systems are *not* like this, and can be treated adequately via **linearization** about an operating point of interest, as illustrated in the several examples presented next. △

[6] Most students attempt this at least once when moving into or out of college and/or graduate school...

[7] This type of frictional force is often referred to as **stiction**.

[8] If the belt is at an angle θ from horizontal, the normal force $mg\cos(\theta)$ across the interface should be used instead.

[9] Tables of such coefficients, for different materials in contact, are broadly available on the web.

[10] Indeed, it is our experience that *most* control problems encountered in practice may be treated effectively with linear methods.

[11] The phrase "highly nonlinear", like "mostly dead" and "very unique", should be avoided in scientific writing.

Example 6.4 A rolling cart system, and its linearization

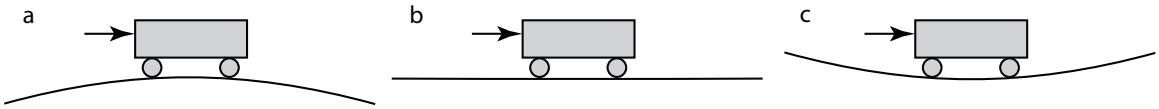


Figure 6.4: A simple second-order system in: (a) an unstable configuration with $\beta < 0$, (b) a neutrally-stable configuration with $\beta = 0$, and (c) an oscillatory configuration with $\beta > 0$.

We now consider now the dynamics of a simple rolling cart, as shown in Figure 6.4, governed by

$$m\,\frac{d^2x}{dt^2} + c\,\frac{dx}{dt} + m\,g\,\sin\left(\frac{dy}{dx}\right) = u, \qquad \text{where} \qquad y = \beta\,x^2.$$

Combining these two equations, applying the identity $\sin(\epsilon) = \epsilon - \epsilon^3/3! + \ldots$, and **linearizing** (i.e., performing the necessary Taylor series expansions and, assuming x and u are small, neglecting all terms that are quadratic or higher in x and/or u) leads to

$$\frac{d^2x}{dt^2} + a_1\frac{dx}{dt} + a_0\,x = b_0\,u \quad \text{where} \quad a_1 = \frac{c}{m},\;\; a_0 = 2\beta g,\;\; b_0 = \frac{1}{m}. \qquad \triangle$$

Example 6.5 Inverted and hanging pendulum/cart systems, and their linearization

It is straightforward to derive the full nonlinear equations of motion of the inverted and hanging pendulum/cart systems illustrated in Figure 6.5. Define P_x and P_y as the forces the pendulum exerts on the cart in the $\mathbf{e}^1$ and $\mathbf{e}^2$ directions (and, thus, the cart exerts the opposite forces on the pendulum), $x(t)$ as the horizontal position of the cart, $\theta(t)$ as the angle of the pendulum (measured counterclockwise from upright), and $\mathbf{r}(t)$ as a vector from a (stationary) coördinate system origin to the center of mass of the pendulum. Writing $\mathbf{r}(t)$ as a function of $x(t)$ and $\theta(t)$ (known as a **kinematic** relationship), differentiating twice, and reärranging gives

$$\mathbf{r} = [\,x - \ell\,\sin\theta\,]\,\mathbf{e}^1 + [\,\ell\,\cos\theta\,]\,\mathbf{e}^2, \tag{6.7a}$$

$$\frac{d^2\mathbf{r}}{dt^2} = \left[\frac{d^2x}{dt^2} - \ell\,\cos\theta\,\frac{d^2\theta}{dt^2} + \ell\,\sin\theta\left(\frac{d\theta}{dt}\right)^2\right]\mathbf{e}^1 - \left[\ell\,\sin\theta\,\frac{d^2\theta}{dt^2} + \ell\,\cos\theta\left(\frac{d\theta}{dt}\right)^2\right]\mathbf{e}^2 \tag{6.7b}$$

$$= \left[\cos\theta\,\frac{d^2x}{dt^2} - \ell\,\frac{d^2\theta}{dt^2}\right]\mathbf{e}^\perp - \left[\sin\theta\,\frac{d^2x}{dt^2} + \ell\left(\frac{d\theta}{dt}\right)^2\right]\mathbf{e}^\parallel, \tag{6.7c}$$

where $\mathbf{e}^\perp = \mathbf{e}^1\cos\theta + \mathbf{e}^2\sin\theta$ is the direction perpendicular to the pendulum, and $\mathbf{e}^\parallel = \mathbf{e}^2\cos\theta - \mathbf{e}^1\sin\theta$ is the direction parallel to the pendulum (see Figure 6.5a). We then write Newton's second law for the acceleration in the $\mathbf{e}^1$ direction of the cart, and the pendulum, and Newton's second law of rotation for the pendulum:

$$m_c\frac{d^2x}{dt^2} = P_x + u, \tag{6.8a}$$

$$m_p\left[\frac{d^2\mathbf{r}}{dt^2}\cdot\mathbf{e}^1\right] = m_p\left[\frac{d^2x}{dt^2} - \ell\,\cos\theta\,\frac{d^2\theta}{dt^2} + \ell\,\sin\theta\left(\frac{d\theta}{dt}\right)^2\right] = -P_x \tag{6.8b}$$

$$I_p\,\frac{d^2\theta}{dt^2} = -P_y\,\ell\,\sin\theta - P_x\,\ell\,\cos\theta; \tag{6.8c}$$

we are also interested in Newton's second law for the acceleration of the pendulum in the $\mathbf{e}^\perp$ direction:

$$m_p\left[\frac{d^2\mathbf{r}}{dt^2}\cdot\mathbf{e}^\perp\right] = m_p\left[\cos\theta\,\frac{d^2x}{dt^2} - \ell\,\frac{d^2\theta}{dt^2}\right] = -m_p\,g\,\sin\theta - P_y\,\sin\theta - P_x\,\cos\theta. \tag{6.8d}$$

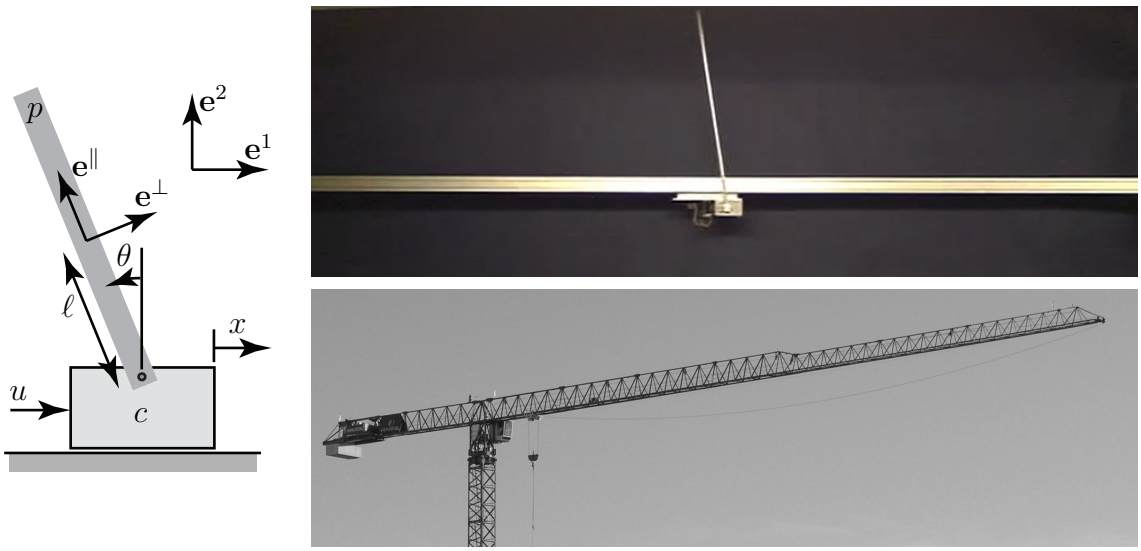


Figure 6.5: Pendulum/cart systems: (a) schematic, (b) lab realization in the **inverted** configuration $\theta(t) \approx 0$ [see (6.10)], and (c) a large-scale realization in the **hanging** configuration $\theta(t) \approx \pi$ [see (6.11)].

Note that $\{m_p, m_c\}$ are the masses of the pendulum and cart, I_p is the moment of inertia of the pendulum about its center of mass, ℓ is the distance from the center of mass of the pendulum to the point where it is pivotally attached to the cart, and g is the acceleration due to gravity; all of these parameters are positive.

First combining (6.8a) and (6.8b), then combining (6.8c) and (6.8d), leads to the two nonlinear equations of motion:

$$(m_c + m_p)\,\frac{d^2x}{dt^2} \quad - m_p\,\ell\,\cos\theta\,\frac{d^2\theta}{dt^2} + m_p\,\ell\,\sin\theta\Big(\frac{d\theta}{dt}\Big)^2 = u, \tag{6.9a}$$

$$-m_p\,\ell\,\cos\theta\,\frac{d^2x}{dt^2} + (I_p + m_p\,\ell^2)\,\frac{d^2\theta}{dt^2} - m_p\,g\,\ell\,\sin\theta \qquad = 0. \tag{6.9b}$$

Linearization of this system is performed by taking $x = \overline{x} + x'$, $\theta = \overline{\theta} + \theta'$, and $u = \overline{u} + u'$ in (6.9), expanding with Taylor series, multiplying out, applying the fact that the **nominal** condition $\{\overline{x}, \overline{\theta}, \overline{u}\}$ is itself also a solution of (6.9), and keeping only those terms which are linear in the perturbation (primed) quantities, as terms that are quadratic or higher in the perturbations are negligible if the perturbations are sufficiently small. Often, a nonlinear system is linearized about a *stationary* (a.k.a. **equilibrium**) nominal condition; such an equilibrium condition might be **stable**, such as the **hanging pendulum** configuration with $\{\overline{x} = 0, \overline{\theta} = \pi, \overline{u} = 0\}$, or **unstable**, such as the **inverted pendulum** configuration with $\{\overline{x} = 0, \overline{\theta} = 0, \overline{u} = 0\}$. More generally, the nominal condition about which small perturbations of a nonlinear system are modeled in a linearization may also be an *unsteady* trajectory of the system considered, which we denote $\{\overline{x}(t), \overline{\theta}(t), \overline{u}(t)\}$ for the problem considered in (6.9); this is called a **tangent linear** approximation of the equations of motion governing perturbations $\{x'(t), \theta'(t), u'(t)\}$ of the system from the "target" nominal trajectory $\{\overline{x}(t), \overline{\theta}(t), \overline{u}(t)\}$.

Taking $\{\overline{x} = 0, \overline{\theta} = 0, \overline{u} = 0\}$, the linearized equations of motion of the **inverted pendulum** are

$$(m_c + m_p)\,\frac{d^2x'}{dt^2} \qquad - m_p\,\ell\,\frac{d^2\theta'}{dt^2} \qquad = u', \tag{6.10a}$$

$$-m_p\,\ell\,\frac{d^2x'}{dt^2} + (I_p + m_p\,\ell^2)\,\frac{d^2\theta'}{dt^2} - m_p\,g\,\ell\,\theta' = 0. \tag{6.10b}$$

Taking $\{\overline{x}=0, \overline{\theta}=\pi, \overline{u}=0\}$, the linearized equations of motion of the **hanging pendulum** are

$$(m_c+m_p)\,\frac{d^2x'}{dt^2} \qquad +m_p\,\ell\,\frac{d^2\theta'}{dt^2} \qquad = u', \tag{6.11a}$$

$$m_p\,\ell\,\frac{d^2x'}{dt^2}+(I_p+m_p\,\ell^2)\,\frac{d^2\theta'}{dt^2}+m_p\,g\,\ell\,\theta' = 0. \tag{6.11b}$$

Finally, considering an unsteady nominal trajectory $\{\overline{x}(t), \overline{\theta}(t), \overline{u}(t)\}$ gives

$$\begin{aligned}(m_c+m_p)\,\frac{d^2(\overline{x}+x')}{dt^2}-m_p\,\ell\,\cos(\overline{\theta}+\theta')\,\frac{d^2(\overline{\theta}+\theta')}{dt^2}+m_p\,\ell\,\sin(\overline{\theta}+\theta')\Big(\frac{d(\overline{\theta}+\theta')}{dt}\Big)^2 &= \overline{u}+u',\\ -m_p\,\ell\,\cos(\overline{\theta}+\theta')\,\frac{d^2(\overline{x}+x')}{dt^2} \qquad +(I_p+m_p\,\ell^2)\,\frac{d^2(\overline{\theta}+\theta')}{dt^2}-m_p\,g\,\ell\,\sin(\overline{\theta}+\theta') &= 0;\end{aligned}$$

leveraging (B.52) and (B.53), multiplying out, applying the condition that $\{\overline{x}(t), \overline{\theta}(t), \overline{u}(t)\}$ itself satisfies (6.9), and keeping only those terms linear in the perturbation (primed) quantities then gives the **tangent linear** approximation of the equations of motion governing perturbations of the pendulum system, $\{x'(t), \theta'(t), u'(t)\}$, in the vicinity of any "target" nominal trajectory $\{\overline{x}(t), \overline{\theta}(t), \overline{u}(t)\}$:

$$\begin{aligned}(m_c+m_p)\,\frac{d^2x'}{dt^2}\,-m_p\,\ell\,\cos(\overline{\theta})\,\frac{d^2\theta'}{dt^2}+m_p\,\ell\Big[\frac{d^2\overline{\theta}}{dt^2}\sin(\overline{\theta})\,\theta'+\Big(\frac{d\overline{\theta}}{dt}\Big)^2(\cos\overline{\theta})\theta'+2\,\frac{d\overline{\theta}}{dt}\,\sin(\overline{\theta})\,\frac{d\theta'}{dt}\Big] &= u'.\\ -m_p\,\ell\,\cos(\overline{\theta})\,\frac{d^2x'}{dt^2}+(I_p+m_p\,\ell^2)\,\frac{d^2\theta'}{dt^2}-m_p\,\ell\Big[g\,(\cos\overline{\theta})\,\theta'-\frac{d^2\overline{x}}{dt^2}\sin(\overline{\theta})\,\theta'\Big] &= 0. \qquad \triangle\end{aligned}$$

Example 6.6 The Mobile Inverted Pendulum problem, and its linearization

The derivation of the equations of motion of the Mobile Inverted Pendulum (MIP; see Figure 6.6) is related to that of the classical inverted pendulum (Example 6.5). Define P_x and P_y as the forces that the MIP body exerts on the wheels in the $\mathbf{e}^1$ and $\mathbf{e}^2$ directions, $x(t)$ as the horizontal position of the center of the wheels, $\phi(t)$ as the angle of rotation of the wheels as they roll (measured counterclockwise from a reference position), $\theta(t)$ as the angle of the MIP body (measured counterclockwise from upright, with $-\pi/2 < \theta(t) < \pi/2$), and $\mathbf{r}(t)$ as a vector from a stationary coördinate system origin to the center of mass of the MIP body. Writing $\mathbf{r}(t)$ as a function of $x(t)$ and $\theta(t)$, differentiating twice, and re"arranging again gives (6.7).

A motor is attached which applies an input torque τ that tends to rotate the body in one direction and the wheels in the other. We assume for the moment that the two wheels of the vehicle move together (that is, the vehicle isn't turning), and that a stiction force between the wheels and the ground is generated such that the wheels do not slip, and thus

$$r\,\phi = x. \tag{6.12}$$

We then write Newton's second law for the acceleration in the $\mathbf{e}^1$ direction of the wheel centers and the center-of-mass of the MIP body, and Newton's second law of rotation for the MIP body and the wheels:

$$m_w\frac{d^2x}{dt^2} = P_x - f, \tag{6.13a}$$

$$m_b\Big[\frac{d^2\mathbf{r}}{dt^2}\cdot\mathbf{e}^1\Big] = m_b\Big[\frac{d^2x}{dt^2}-\ell\,\cos\theta\,\frac{d^2\theta}{dt^2}+\ell\,\sin\theta\Big(\frac{d\theta}{dt}\Big)^2\Big] = -P_x, \tag{6.13b}$$

$$I_b\,\frac{d^2\theta}{dt^2} = -\tau - P_y\,\ell\,\sin\theta - P_x\,\ell\,\cos\theta, \tag{6.13c}$$

$$I_w\,\frac{d^2\phi}{dt^2} = \tau - r\,f; \tag{6.13d}$$

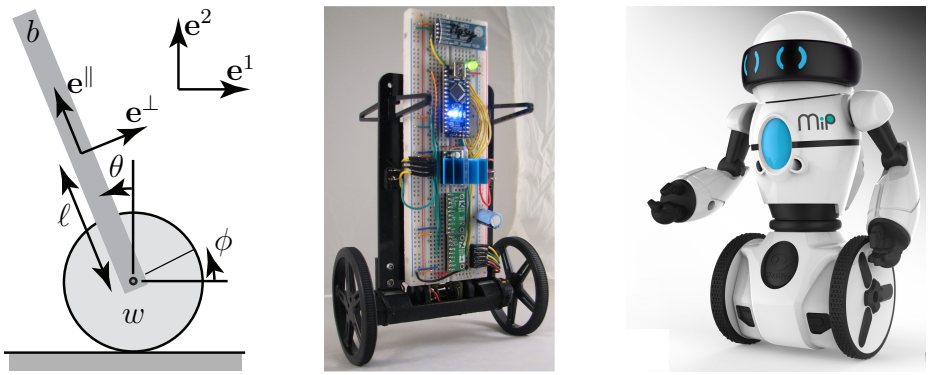


Figure 6.6: MIP, a Mobile Inverted Pendulum; (a) schematic, (b) prototype, (c) toy product commercialized by WowWee Robotics and the UCSD Coördinated Robotics Lab.

we are also interested in Newton's second law for the acceleration of the MIP body in the $\mathbf{e}^\perp$ direction:

$$m_b\Big[\frac{d^2\mathbf{r}}{dt^2}\cdot\mathbf{e}^\perp\Big] = m_b\Big[\cos\theta\,\frac{d^2x}{dt^2} - \ell\,\frac{d^2\theta}{dt^2}\Big] = -m_b\,g\,\sin\theta - P_y\,\sin\theta - P_x\,\cos\theta. \tag{6.13e}$$

Note that $\{m_b, m_w, I_b, I_w\}$ are the masses and moments of inertia (about their respective centers of mass) of the body and the sum of both wheels moving together, r is the radius of the wheels, ℓ is the distance from the center of mass of the MIP body to the axis of rotation of the wheels, g is the acceleration due to gravity; all of these parameters are positive. First combining (6.12), (6.13a), (6.13b) and (6.13d), then combining (6.13c) and (6.13e), leads to the nonlinear equations of motion of the MIP:

$$[I_w + (m_w + m_b)r^2]\,\frac{d^2\phi}{dt^2} + m_b\,r\,\ell\,\cos\theta\,\frac{d^2\theta}{dt^2} - m_b\,r\,\ell\,\sin\theta\Big(\frac{d\theta}{dt}\Big)^2 = \tau, \tag{6.14a}$$

$$m_b\,r\,\ell\,\cos\theta\,\frac{d^2\phi}{dt^2} + (I_b + m_b\,\ell^2)\,\frac{d^2\theta}{dt^2} - m_b\,g\,\ell\,\sin\theta = -\tau. \tag{6.14b}$$

Linearization of this system is performed by taking $\phi = \overline{\phi} + \phi'$, $\theta = \overline{\theta} + \theta'$, and $u = \overline{u} + u'$ in (6.14), applying the fact that the **nominal** condition $\{\overline{\theta}, \overline{\phi}, \overline{u}\}$ is itself also a solution of (6.14), and keeping only those terms which are linear in the perturbation (primed) quantities.

Taking $\{\overline{\phi} = 0, \overline{\theta} = 0, \overline{u} = 0\}$, the linearized equations of motion of the MIP about its upright state are

$$[I_w + (m_w + m_b)r^2]\,\frac{d^2\phi'}{dt^2} + m_b\,r\,\ell\,\frac{d^2\theta'}{dt^2} = \tau', \tag{6.15a}$$

$$m_b\,r\,\ell\,\frac{d^2\phi'}{dt^2} + (I_b + m_b\,\ell^2)\,\frac{d^2\theta'}{dt^2} - m_b\,g\,\ell\,\theta' = -\tau'. \tag{6.15b}$$

Considering an unsteady nominal trajectory $\{\overline{\phi}(t), \overline{\theta}(t), \overline{u}(t)\}$ and applying the same manipulations as before gives the **tangent linear** approximation of the equations governing the perturbations $\{\phi'(t), \theta'(t), u'(t)\}$ of the MIP in the vicinity of the nominal trajectory $\{\overline{\phi}(t), \overline{\theta}(t), \overline{u}(t)\}$:

$$[I_w + (m_w + m_b)r^2]\,\frac{d^2\phi'}{dt^2} + m_b\,r\,\ell\,\cos\overline{\theta}\,\frac{d^2\theta'}{dt^2} + m_b\,r\,\ell\,\Big[2\,\sin\overline{\theta}\,\frac{d\overline{\theta}}{dt}\,\frac{d\theta'}{dt} - \sin\overline{\theta}\,\frac{d^2\overline{\theta}}{dt^2}\theta' + \cos\overline{\theta}\,\Big(\frac{d\overline{\theta}}{dt}\Big)^2\theta'\Big] = \tau',$$

$$m_b\,r\,\ell\cos\overline{\theta}\,\frac{d^2\phi'}{dt^2} + (I_b + m_b\,\ell^2)\,\frac{d^2\theta'}{dt^2} - m_b\,g\,\ell\,(\cos\overline{\theta})\,\theta' + m_b\,r\,\ell\sin\overline{\theta}\,\frac{d^2\overline{\phi}}{dt^2}\,\theta' = -\tau'. \qquad \triangle$$

Example 6.7 A (linear) temperature bath system with a transport delay

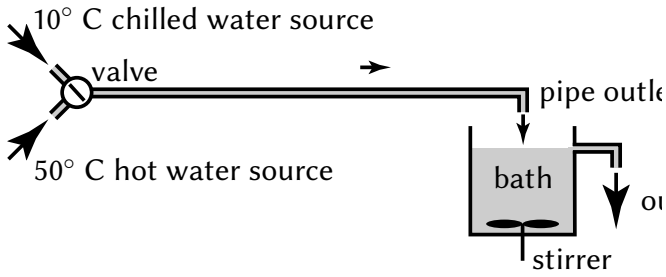


Figure 6.7: The temperature bath system set up in Example 6.7. The valve allows an adjustment of the flow temperature between $T_{\text{valve, min}} = 10^\circ$ C and $T_{\text{valve, max}} = 50^\circ$ C, and maintains a constant flow rate of $dV/dt = 6$ liters per minute in the inflow pipe; the wastewater flows out from the bath at precisely the same rate. The volume of fluid in the inflow pipe and the bath at any given time are $V_{\text{pipe}} = 1.2$ liters and $V_{\text{bath}} = 20$ liters. We assume further that (i) the inflow pipe is perfectly insulated, (ii) its walls have negligible thermal capacity, (iii) there is negligible heat diffusion within the fluid as it flows through the inflow pipe, and (iv) the bath is stirred quickly enough that it is maintained at essentially uniform temperature.

Performing a **control volume** analysis of the temperature bath system illustrated in Figure 6.7 to compute the **thermal energy** of the bath at time $t + \Delta t$ (for small Δt), at which time the bath has lost ΔV of the liquid it had at time t and gained ΔV of the new liquid from the pipe outlet, it follows that

$$V_{\text{bath}}\, T_{\text{bath}}(t + \Delta t) = (V_{\text{bath}} - \Delta V) T_{\text{bath}}(t) + \Delta V\, T_{\text{outlet}}(t) = (V_{\text{bath}} - \Delta V) T_{\text{bath}}(t) + \Delta V\, T_{\text{valve}}(t - d),$$

where $d = V_{\text{pipe}} \,/\, dV/dt = 12$ s represents the **convective transport delay** (that is, the time it takes the fluid to flow from the valve to the pipe outlet), and thus

$$T_{\text{bath}}(t) + \frac{dT_{\text{bath}}(t)}{dt}\Delta t + \ldots = T_{\text{bath}}(t) + \frac{\Delta V}{V_{\text{bath}}}[T_{\text{valve}}(t - d) - T_{\text{bath}}(t)];$$

taking $y(t) = T_{\text{bath}}(t)$, $u(t) = T_{\text{valve}}(t)$, and $a_0 = b_0 = dV/dt \,/\, V_{\text{bath}} = 0.005$ s^{-1}, in the limit of small Δt we have

$$\frac{dT_{\text{bath}}(t)}{dt} = \frac{dV/dt}{V_{\text{bath}}}\Big[T_{\text{valve}}(t - d) - T_{\text{bath}}(t)\Big] \qquad \Rightarrow \qquad \frac{dy(t)}{dt} + a_0\, y(t) = b_0\, u(t - d). \qquad \triangle$$

Example 6.8 An automobile with a throttle delay, and its linearization

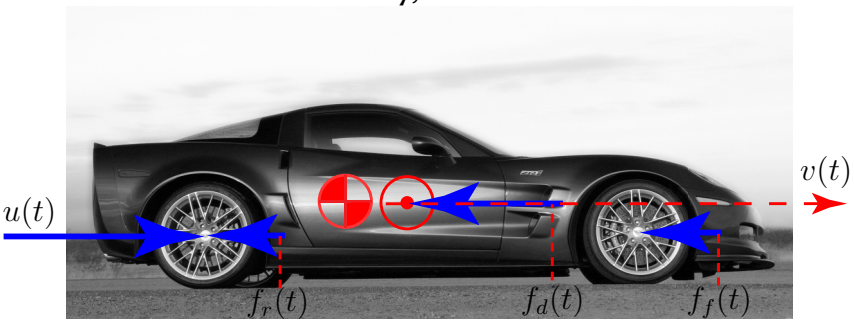


Figure 6.8: Coördinate system for the analysis of an automobile at cruise, as considered in Example 6.8 (to clarify the drawing, the vertical and lateral forces are not marked).

The equations of motion for the velocity $v(t)$ an automobile at cruise (see Figure 6.8) may be written

$$\begin{aligned} m\,\frac{dv(t)}{dt} &= u(t-d) - f_d(t) - [f_r(t) + f_f(t)] + w(t) \\ &= u(t-d) - C_d\,A\,\cdot 0.5\,\rho\,v(t)^2 - m\,g[C_0 + C_1 v(t)^{2.5}] + w(t), \end{aligned} \tag{6.16}$$

where

- $u(t)$ denotes the "control" force applied to accelerate the vehicle by the engine,
- $f_d(t) = C_d A \cdot 0.5\,\rho\,v(t)^2$ models the aerodynamic drag,
- $f_r(t) + f_f(t) = m\,g[C_0 + C_1 v(t)^{2.5}]$ models the rolling drag (see p. 117 of Gillespie 1992), and
- $w(t)$ denotes the "disturbances" (caused by headwind/tailwind, road inclination, modeling errors, etc.).

Note that this model accounts for a slight delay d between the actuation of the throttle and its effect on the force applied to accelerate the vehicle. In our model of the vehicle depicted in Figure 6.8, we will take $C_d = 0.36$, $A = 2.06$, $\rho = 1.2$, $m = 1520$, $g = 9.8$, $C_0 = .01$, $C_1 = 1.2 \cdot 10^{-6}$, and $d = 0.04$, where all variables are in **SI units** (length in meters, mass in kilograms, time in seconds, force in Newtons, etc.)

At an equilibrium target car velocity, $v(t) = \overline{v}$, the corresponding throttle position $u(t) = \overline{u}$ is given by

$$d\overline{v}/dt = d\overline{u}/dt = 0, \quad \overline{u} = C_d A \cdot 0.5\,\rho\,\overline{v}^2 + m\,g[C_0 + C_1 \overline{v}^{2.5}]. \tag{6.17}$$

We now show how to linearize the dynamics of this system, taking $v(t) = \overline{v} + v'(t)$ and $u(t) = \overline{u} + u'(t)$ where $v'(t)$ and $u'(t)$ denote perturbations to the equilibrium car velocity and throttle position respectively.

Recall that any smooth function $f(x)$ may be expanded about $x = \overline{x}$ via a Taylor Series as follows:

$$f(\overline{x} + x') = f(x)\Big|_{x=\overline{x}} + \frac{df(x)}{dx}\Big|_{x=\overline{x}}(x') + \frac{d^2 f(x)}{dx^2}\Big|_{x=\overline{x}}\frac{(x')^2}{2!} + \frac{d^3 f(x)}{dx^3}\Big|_{x=\overline{x}}\frac{(x')^3}{3!} + \ldots;$$

thus, the function $f(v) = v^{2.5}$ may be expanded about $v = \overline{v}$ as follows:

$$[\overline{v} + v']^{2.5} = \overline{v}^{2.5} + 2.5\,\overline{v}^{1.5}\,v' + O[(v')^2]. \tag{6.18}$$

Considering small perturbations about the equilibrium condition $\{\overline{v}, \overline{u}\}$, by substituting $v(t) = \overline{v} + v'(t)$ and $u(t) = \overline{u} + u'(t)$ into (6.16), applying (6.18), then applying (6.17), then finally eliminating all terms which are quadratic or higher in the perturbation (primed) quantities, leads to a linear equation as follows:

$$\begin{aligned} m\,\frac{d\{\overline{v} + v'(t)\}}{dt} &= \{\overline{u} + u'(t-d)\} - C_d\,A\,\cdot 0.5\,\rho\,\{\overline{v} + v'(t)\}^2 - m\,g[C_0 + C_1\{\overline{v} + v'(t)\}^{2.5}], \\ m\,\frac{d\{\overline{v} + v'(t)\}}{dt} &= \overline{u} + u'(t-d) - 0.5\,C_d\,A\,\rho\,\{\overline{v}^2 + 2\overline{v}[v'(t)] + [v'(t)]^2\} \\ &\qquad - m\,g[C_0 + C_1\{\overline{v}^{2.5} + 2.5\overline{v}^{1.5}v'(t) + O[(v'(t))^2]\}], \\ \Rightarrow \quad m\,\frac{d\{v'(t)\}}{dt} &= u'(t-d) - 0.5\,C_d\,A\,\rho\,\{2\,\overline{v}\,v'(t) + [v'(t)]^2\} - m\,g[C_1\{2.5\,\overline{v}^{1.5}\,v'(t) + O[(v'(t))^2]\}], \\ m\,\frac{d\{v'(t)\}}{dt} &= u'(t-d) - C_d\,A\,\rho\,\overline{v}\,v'(t) - m\,g\,C_1\,2.5\,\overline{v}^{1.5}\,v'(t), \end{aligned}$$

thus resulting in the linear ODE

$$\left(\frac{d}{dt} + a_0\right) v'(t) = b_0\,u'(t-d) \qquad \text{where} \qquad a_0 = C_d\,A\,\rho\,\overline{v}/m + 2.5\,C_1\;g\,\overline{v}^{1.5} > 0. \qquad \triangle$$

Example 6.9 A three-story building during an earthquake, and its linearization

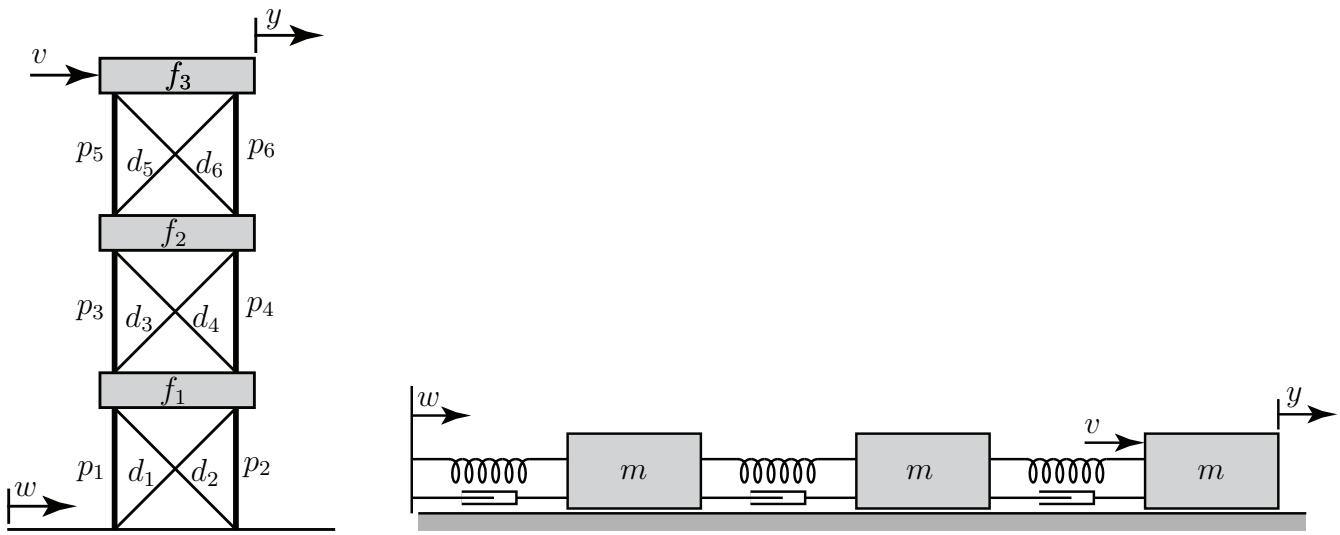


Figure 6.9: (a) The three-story building considered in Example 6.9, and (b) a cascade spring/mass/damper system which provides an equivalent model for the linearized horizontal dynamics of this structure.

We now analyze the dynamics of the three-story building illustrated in Figure 6.9 during an earthquake.

Each of the three floors is of mass $m = 1000$ kg, and the diagonals are nominally at 45° angles. The lengths of the pillars and floors are nominally $\ell_p = \ell_f \triangleq \ell = 5$ m; by the Pythagorean theorem, the lengths of the diagonals are nominally $\ell_d = \ell\sqrt{2}$. All joints are assumed to be **pinned**, so no members bear bending loads. The vertical pillars are under compression. The diagonal stabilizers are under tension, and each has a spring constant k and damping coefficient c; note that the structure is **pretensioned**, so the diagonal members remain under tension even as the building is deformed. An earthquake is modeled as horizontal motion of the ground, $w(t)$. We are primarily interested in the horizontal motion of the top floor, $y(t)$, which, we will see, can deflect a lot even for relatively small ground motion $w(t)$ if the building is forced at a critical **resonant frequency** of the structure by the earthquake.

We now model the horizontal motion of the ground floor, $x_1(t)$, the second floor, $x_2(t)$, and the top floor $x_3(t) = y(t)$, as a function of both the horizontal motion of the ground, $w(t)$, and the force applied to the top floor, $v(t)$. It will be seen that we can neglect the vertical motions of the floors, which (for small deflections) are small as compared with the horizontal motions.

Assume first that the horizontal position of the third floor is perturbed a small amount to the right of the horizontal position of the second floor; that is, $0 < (x_3 - x_2)/\ell \ll 1$. Denote by $\theta_6 = \sin^{-1}[(x_3 - x_2)/\ell] \approx (x_3 - x_2)/\ell$ the (clockwise) angle that the sixth pillar is deflected from its nominally vertical orientation. Note that, since $\cos\theta_6 = 1 + O([(x_3 - x_2)/\ell]^2)$, the perturbations in the vertical forces and vertical deflections of the floor are *quadratic* in the horizontal perturbation quantities; that is, they are negligible as compared with the horizontal forces and deflections if these quantities are small. The (clockwise) angles that the other pillars are deflected from their nominally vertical orientations may be defined and computed similarly,

$$\theta_5 \approx \theta_6 \approx (x_3 - x_2)/\ell, \quad \theta_3 \approx \theta_4 \approx (x_2 - x_1)/\ell, \quad \theta_1 \approx \theta_2 \approx (x_1 - w)/\ell, \tag{6.19}$$

and also result in negligible perturbations in the vertical forces and vertical positions of the floors; we thus focus on the horizontal dynamics in the remainder of this example.

Denote by δ_5 the changes in length of the fifth diagonal member from its nominal (pretentioned) length ℓ_d. Noting that $|x_3 - x_2|/\ell \ll 1$ [and, therefore, $|\delta_5|/\ell \ll 1$], we again appeal to the Pythagorean theorem:

$$\ell^2 + [\ell + (x_3 - x_2)]^2 = [\ell_d + \delta_5]^2 \;\Rightarrow\; 2\,\ell\,(x_3 - x_2) + (x_3 - x_2)^2 = 2\sqrt{2}\,\ell\,\delta_5 + \delta_5^2 \;\Rightarrow\; \delta_5 \approx (x_3 - x_2)/\sqrt{2}.$$

Again, when performing linearizations of this sort, terms which are quadratic in the perturbation quantities are negligible as compared with terms which are linear in the perturbation quantities, which are assumed to be small. The changes in lengths of the other diagonal members may be computed similarly:

$$\begin{aligned} \delta_5 &\approx \quad (x_3 - x_2)/\sqrt{2}, \quad \delta_3 \approx \quad (x_2 - x_1)/\sqrt{2}, \quad \delta_1 \approx \quad (x_1 - w)/\sqrt{2}, \\ \delta_6 &\approx -(x_3 - x_2)/\sqrt{2}, \quad \delta_4 \approx -(x_2 - x_1)/\sqrt{2}, \quad \delta_2 \approx -(x_1 - w)/\sqrt{2}. \end{aligned} \tag{6.20}$$

Finally, denote by $\phi_5 = \tan^{-1}[(\ell + x_3 - x_2)/\ell] - \pi/4$ the angle that the fifth diagonal member is rotated from its nominal $\pi/4$ radian orientation (again, measured clockwise from vertical). Noting the identities (B.55), (B.90), and (B.87),

$$\tan(x+y) = \frac{\tan x + \tan y}{1 - \tan x \, \tan y}, \quad \frac{1}{1-\epsilon} = 1 + \epsilon + \epsilon^2 + \ldots, \quad \tan(\epsilon) = \epsilon + \frac{\epsilon^3}{3} + \ldots,$$

when $\phi_5 \ll 1$ it follows that

$$\tan(\pi/4 + \phi_5) = \frac{1 + \tan\phi_5}{1 - \tan\phi_5} = [1 + \phi_5 + O(\phi_5^3)][1 + \phi_5 + O(\phi_5^2)] = 1 + 2\phi_5 + O(\phi_5^2);$$

thus, $\phi_5 \approx (x_3 - x_2)/(2\ell)$. The (clockwise) angles that the other diagonal members are rotated from their nominal orientations ($\pm\pi/4$ radians from vertical) may be computed similarly:

$$\phi_5 \approx \phi_6 \approx (x_3 - x_2)/(2\ell), \quad \phi_3 \approx \phi_4 \approx (x_2 - x_1)/(2\ell), \quad \phi_1 \approx \phi_2 \approx (x_1 - w)/(2\ell). \tag{6.21}$$

We are now in a position to add up all of the horizontal forces on the floors when the structure undergoes small horizontal movements. The horizontal acceleration of the top floor is acted upon by the external force v, the horizontal component of the force from the two rotated pillars [see (6.19)], and the horizontal component of the force from the two extended [see (6.20)] and rotated [see (6.21)] diagonal members; noting the nominal loading computed in (**??**) and the identity (B.52) [$\sin(x+y) = \sin x \, \cos y + \cos x \, \sin y$], neglecting terms that are quadratic or higher in the perturbations, this may be summed up as follows,

$$\begin{aligned} m\frac{d^2x_3}{dt^2} &= v + p_5 \sin\theta_5 + p_6 \sin\theta_6 - \left(d_5 + k\,\delta_5 + c\,\frac{d\delta_5}{dt}\right)\sin\left(\frac{\pi}{4} + \theta_5\right) - \left(d_6 + k\,\delta_6 + c\,\frac{d\delta_5}{dt}\right)\sin\left(-\frac{\pi}{4} + \theta_6\right) \\ &\approx v + 2p_5\frac{x_3 - x_2}{\ell} - \left[d_5 + \left(k + c\frac{d}{dt}\right)\frac{x_3 - x_2}{\sqrt{2}}\right]\frac{1 + (x_3 - x_2)/\ell}{\sqrt{2}} \\ &\qquad - \left[d_6 - \left(k + c\frac{d}{dt}\right)\frac{x_3 - x_2}{\sqrt{2}}\right]\frac{-1 + (x_3 - x_2)/\ell}{\sqrt{2}} \\ &\approx v + 2p_5\frac{x_3 - x_2}{\ell} - 2d_5\frac{x_3 - x_2}{\sqrt{2}\ell} - 2\left(k + c\frac{d}{dt}\right)\frac{x_3 - x_2}{2} \\ &\approx v - \overline{k}_3(x_3 - x_2) - c\left(\frac{dx_3}{dt} - \frac{dx_2}{dt}\right); \end{aligned} \tag{6.22a}$$

note that the horizontal forces of the other two floors may be summed up in a similar fashion,

$$m\frac{d^2x_2}{dt^2} = -\overline{k}_2(x_2 - x_1) + \overline{k}_3(x_3 - x_2) - c\left(\frac{dx_2}{dt} - \frac{dx_1}{dt}\right) + c\left(\frac{dx_3}{dt} - \frac{dx_2}{dt}\right), \tag{6.22b}$$

$$m\frac{d^2x_1}{dt^2} = -\overline{k}_1(x_1 - w) + \overline{k}_2(x_2 - x_1) - c\left(\frac{dx_1}{dt} - \frac{dw}{dt}\right) + c\left(\frac{dx_2}{dt} - \frac{dx_1}{dt}\right), \tag{6.22c}$$

where, noting the solution of the statics of this building derived in Example **??**,

$$\overline{k}_1 = k - 2p_1/\ell + \sqrt{2}d_1/\ell = k - 5880 \text{ kg/sec}^2, \tag{6.23a}$$
$$\overline{k}_2 = k - 2p_3/\ell + \sqrt{2}d_3/\ell = k - 3919 \text{ kg/sec}^2, \tag{6.23b}$$
$$\overline{k}_3 = k - 2p_5/\ell + \sqrt{2}d_5/\ell = k - 1960 \text{ kg/sec}^2. \tag{6.23c}$$

For the purpose of clear visualization of the essence of this problem, we may thus ignore the vertical *separation* of the floors, and model small horizontal motions of this building linearly as a cascade spring/mass/damper system, as illustrated in Figure 6.9b, with spring constants $\overline{k}_1$, $\overline{k}_2$, and $\overline{k}_3$ and damping $\overline{c}_1 = \overline{c}_2 = \overline{c}_3 = c$.

The three second-order equations governing x_1, x_2, and x_3 may thus be rewritten as

$$\left(m\frac{d^2}{dt^2} + 2c\frac{d}{dt} + \overline{k}_1 + \overline{k}_2\right)x_1 = \left(c\frac{d}{dt} + \overline{k}_1\right)w + \left(c\frac{d}{dt} + \overline{k}_2\right)x_2 \quad \Rightarrow \quad \mathcal{L}_1 x_1 = \mathcal{L}_2 w + \mathcal{L}_3 x_2, \tag{6.24a}$$
$$\left(m\frac{d^2}{dt^2} + 2c\frac{d}{dt} + \overline{k}_2 + \overline{k}_3\right)x_2 = \left(c\frac{d}{dt} + \overline{k}_2\right)x_1 + \left(c\frac{d}{dt} + \overline{k}_3\right)x_3 \quad \Rightarrow \quad \mathcal{L}_4 x_2 = \mathcal{L}_5 x_1 + \mathcal{L}_6 x_3, \tag{6.24b}$$
$$\left(m\frac{d^2}{dt^2} + c\frac{d}{dt} + \overline{k}_3\right)x_3 = \left(c\frac{d}{dt} + \overline{k}_3\right)x_2 + v \quad \Rightarrow \quad \mathcal{L}_7 x_3 = \mathcal{L}_8 x_2 + v. \tag{6.24c}$$

The task of eliminating x_1 and x_2 from these three second-order ODEs, thereby determining a single sixth-order ODE relating $y = x_3$ to v and w, is algebraically involved; it is thus helpful (as in Example 6.2) to use the streamlined notation introduced above right for the scalar linear differential operators $\mathcal{L}_i$. Premultiplying (6.24a) by $\mathcal{L}_5$ and (6.24b) by $\mathcal{L}_1$ and combining to eliminate x_1 (noting, e.g., that $\mathcal{L}_1\mathcal{L}_5 = \mathcal{L}_5\mathcal{L}_1$) leads to

$$\mathcal{L}_1\mathcal{L}_4 x_2 = (\mathcal{L}_5\mathcal{L}_2 w + \mathcal{L}_5\mathcal{L}_3 x_2) + \mathcal{L}_1\mathcal{L}_6 x_3 \quad \Rightarrow \quad (\mathcal{L}_1\mathcal{L}_4 - \mathcal{L}_5\mathcal{L}_3)x_2 = \mathcal{L}_5\mathcal{L}_2 w + \mathcal{L}_1\mathcal{L}_6 x_3; \tag{6.25}$$

premultiplying (6.25) by $\mathcal{L}_8$ and (6.24c) by $(\mathcal{L}_1\mathcal{L}_4 - \mathcal{L}_5\mathcal{L}_3)$ and combining to eliminate x_2 then leads to

$$(\mathcal{L}_1\mathcal{L}_4\mathcal{L}_7 - \mathcal{L}_5\mathcal{L}_3\mathcal{L}_7 - \mathcal{L}_8\mathcal{L}_1\mathcal{L}_6)x_3 = (\mathcal{L}_1\mathcal{L}_4 - \mathcal{L}_5\mathcal{L}_3)v + (\mathcal{L}_8\mathcal{L}_5\mathcal{L}_2)w,$$

which, denoting $y = x_3$, may be rewritten as

$$\begin{aligned}\left(\frac{d^6}{dt^6} + a_5\frac{d^5}{dt^5} + a_4\frac{d^4}{dt^4} + a_3\frac{d^3}{dt^3} + a_2\frac{d^2}{dt^2} + a_1\frac{d}{dt} + a_0\right)y = \\ \left(b_4\frac{d^4}{dt^4} + b_3\frac{d^3}{dt^3} + b_2\frac{d^2}{dt^2} + b_1\frac{d}{dt} + b_0\right)v + \left(\overline{b}_3\frac{d^3}{dt^3} + \overline{b}_2\frac{d^2}{dt^2} + \overline{b}_1\frac{d}{dt} + \overline{b}_0\right)w.\end{aligned} \tag{6.26}$$

Symbolic manipulation tools may now be used to do the necessary (but tedious) algebraic simplifications (for Matlab implementation, see `RR_Example_06_8.m`) in order to determine the coefficients. As seen by running this code, for $k = 10000$ and $c = 10$, the coefficients work out to be:

$$a_5 = .05, \quad a_4 = 32.361, \quad a_3 = .76881, \quad a_2 = 237.95, \quad a_1 = 1.0706, \quad a_0 = 201.40, \tag{6.27a}$$
$$b_4 = .001, \quad b_3 = .00004, \quad b_2 = .024320, \quad b_1 = .00036480, \quad b_0 = .10706, \tag{6.27b}$$
$$\overline{b}_3 = .000001, \quad \overline{b}_2 = .0018240, \quad \overline{b}_1 = 1.0706, \quad \overline{b}_0 = 201.40; \tag{6.27c}$$

the values of the coefficients for other values of k and c may be determined similarly. The Bode plot of this system, the design of a passive vibration damper for this system, and the conversion of this system is to state-space form, are all problems that are, in due course, considered in the exercises. △

Example 6.10 The launch of a rocket, and its linearization

The dynamics of a Saturn V rocket during liftoff (see Figure 6.10) may be considered in the $x-z$ plane and the $y-z$ plane separately, as the rocket is not spinning. Considering the dynamics in one of these planes, there are three equations governing the motion of the vehicle, two of the form $d^2x/dt^2 = f/m$ and one of the form $d^2\theta/dt^2 = \tau/J$:

$$m\,\frac{d^2z(t)}{dt^2} = f_t\,\cos[\theta(t)-u(t)] - f_d(t)\,\cos[\alpha(t)+\theta(t)] + w(t)\,\sin[\alpha(t)+\theta(t)] - f_g, \tag{6.28a}$$

$$m\,\frac{d^2x(t)}{dt^2} = f_t\,\sin[\theta(t)-u(t)] - f_d(t)\,\sin[\alpha(t)+\theta(t)] - w(t)\,\cos[\alpha(t)+\theta(t)], \tag{6.28b}$$

$$J\,\frac{d^2\theta(t)}{dt^2} = f_t\,D\,\sin[u(t)] \qquad - f_d(t)\,L\,\sin[\alpha(t)] \qquad - w(t)\,L\,\cos[\alpha(t)], \tag{6.28c}$$

which may be taken together with the **kinematic** condition

$$\frac{v_x(t)}{|\mathbf{v}(t)|} = \sin[\alpha(t)+\theta(t)]. \tag{6.29}$$

Note that the three second-order ODES in (6.28) may easily be rewritten as six first-order equations, and describe the evolution in time of the six state variables listed in Figure 6.10.

Taking the disturbance force $w(t)$, the horizontal velocity $v_x(t)$, and the angles $\{\theta(t), \alpha(t), u(t)\}$ to be small, the equation for the vertical acceleration, (6.28a), reduces upon linearization to

$$m\,\frac{d^2z(t)}{dt^2} = f_t - 10\left|\frac{dz(t)}{dt}\right|^2 - f_g, \tag{6.30a}$$

whereas the equations for the horizontal and angular acceleration, (6.28b)-(6.28c), reduce to

$$m\,\frac{d^2x(t)}{dt^2} = f_t\,[\theta(t)-u(t)] - f_d(t)\,[\alpha(t)+\theta(t)] - w(t). \tag{6.30b}$$

$$J\,\frac{d^2\theta(t)}{dt^2} = f_t\,D\,u(t) \qquad - f_d(t)\,L\,\alpha(t) \qquad - L\,w(t). \tag{6.30c}$$

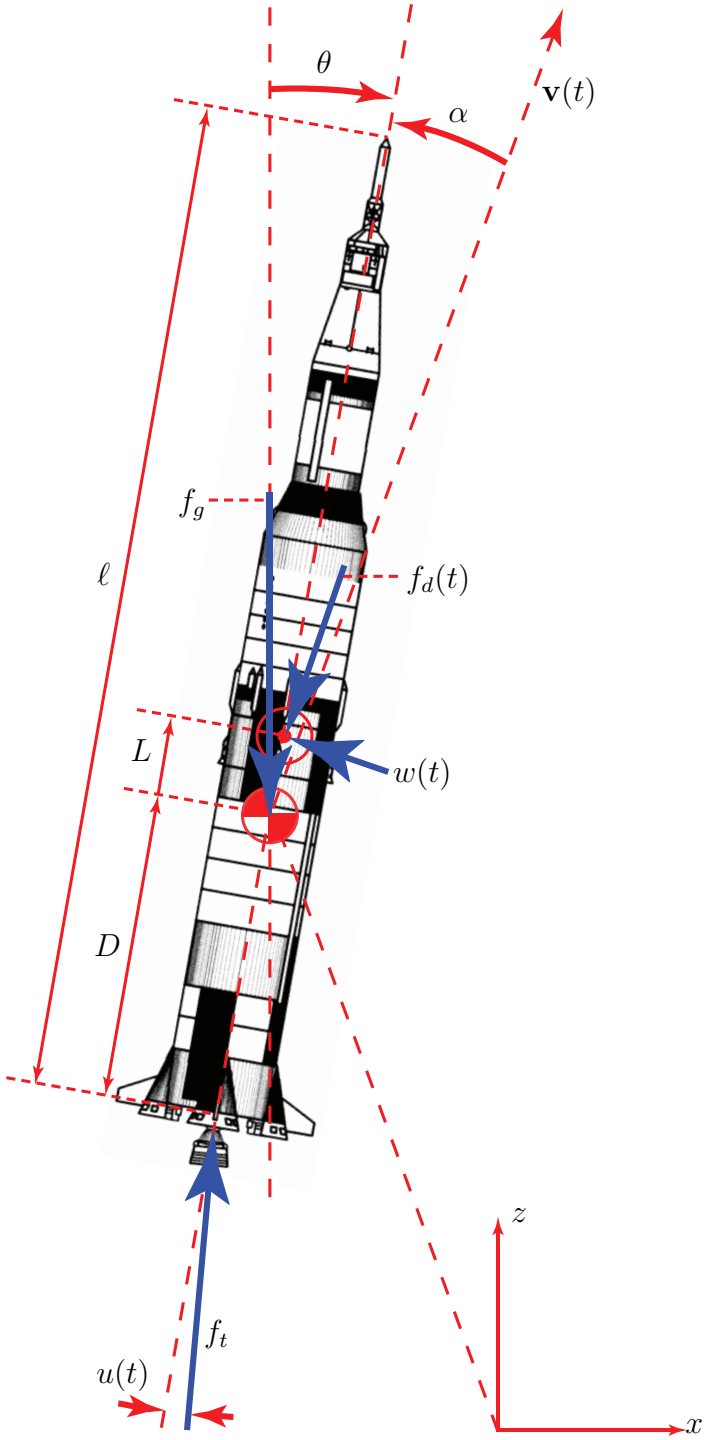
Note that (6.30a) can be marched in order to compute $dz(t)/dt = v_z(t) = |\mathbf{v}(t)|$ at any instant t_1. Given this value of $\overline{v} = |\mathbf{v}(t_1)|$, which varies only slowly in time due to the large mass of the rocket (and the fact that its thrust only slightly exceeds its weight), the linearized form of the auxiliary equation (6.29) may be written

$$\frac{dx(t)}{dt} = \overline{v}\,[\alpha(t)+\theta(t)]. \tag{6.31}$$

Considering $\overline{v}$ as essentially constant, defining $\overline{f}_d = 10\,\overline{v}^2$, and combining (6.30b)-(6.30c) and (6.31) to eliminate α leads to

$$m\,\frac{d^2x(t)}{dt^2} + \frac{\overline{f}_d}{\overline{v}}\,\frac{dx(t)}{dt} - f_t\,\theta(t) = -f_t\,u(t) - w(t), \tag{6.32a}$$

$$\frac{\overline{f}_d\,L}{\overline{v}}\,\frac{dx(t)}{dt} + J\,\frac{d^2\theta(t)}{dt^2} - \overline{f}_d\,L\,\theta(t) = f_t\,D\,u(t) - L\,w(t). \tag{6.32b}$$



State variables:
$z(t)$ = vertical position
$v_z(t) = dz(t)/dt$ = vertical velocity
$x(t)$ = horizontal position
$v_x(t) = dx(t)/dt$ = horizontal velocity
$\theta(t)$ = angle (clockwise from vertical)
$\omega(t) = d\theta(t)/dt$ = angular velocity

Auxiliary variables:
$f_d(t) = 10\,|\mathbf{v}(t)|^2$ = aerodynamic drag
$\alpha(t)$ = angle of attack [see (6.29)]

Control input:
$u(t)$ = angle of thruster

Disturbance input:
$w(t)$ = wind + aerodynamic lift

Constants:
$\ell = 110$ = length
$m = 3 \times 10^6$ = mass
$J = m\,\ell^2/20$ = moment of inertia
$L = 10 \pm 5\%$ = distance C_p is ahead of C_m
$D = 40$ = distance from nozzle pivot to C_m
$f_t = 34 \times 10^6$ = thrust
$f_g = m\,g$ = weight ($g = 9.8$)

Figure 6.10: Coördinate system for a rocket stabilization problem. There are four forces acting on the rocket, directed as indicated: thrust f_t, weight f_g, drag $f_d(t)$, and "disturbances" (lift + wind) $w(t)$; the control $u(t)$ is the angle of the thruster. The rocket is assumed to be not spinning, and all angles indicated are assumed to be small, which decouples the control problem in the x-z plane (shown) from that in the y-z plane. All variables in SI units. [To clarify the diagram, only one of the five thrusters is shown in this sketch of the Saturn V.] △

Example 6.11 Linearized dynamic models of aircraft

The state of an aircraft in flight may be defined by twelve variables: three to identify the position $\{X, Y, Z\}$, three to identify the velocity $\{U, V, W\}$, three to identify the orientation $\{\phi, \theta, \psi\}$, and three to identify the rate of change of the orientation, $\{p, q, r\}$. To identify the orientation, we first denote:

• the **body-fitted coördinates** of the aircraft as three orthogonal vectors from the center of mass out the nose, out the right wingtip, and out the bottom of the aircraft for the x, y, and z axes, respectively, and
• a reference set of **inertial coördinates** (that is, a non-accelerating and non-rotating reference frame) as north, east, and down (NED) from the aircraft center of mass for the x_1, x_2, and x_3 axes, respectively.

Starting from the reference configuration of the aircraft, with its body-fitted coördinates aligned with the inertial (NED) coördinates, the orientation of the aircraft may then be identified unambiguously by three successive rotations[12] about its body-fitted coördinates, the most common choice in the aerodynamics literature being the **3-2-1 Tait-Bryan rotation sequence**[13] (a.k.a. **3-2-1 Euler rotation sequence**[14]) given by:

3 **yaw** the aircraft by an angle ψ about the z (down) axis (positive ψ yaws the nose to the right),
2 **pitch** the aircraft by an angle θ about the y (out-the-right-wing) axis (positive θ pitches the nose up),
1 **roll** the aircraft by an angle ϕ about the x (out-the-nose) axis (positive ϕ rolls the right wing down).

In the reference frame of the aircraft, three convenient auxiliary variables used to describe the dynamics are

• the **airspeed** v_T the magnitude of the **relative wind** past the aircraft,
• the **angle of attack** (**AOA**) α, the angle between the x axis and the component of the relative wind in the $x-y$ plane, and
• the **sideslip angle** β, the angle between the x axis and the component of the relative wind in the $x-z$ plane.

The airspeed, angle of attack, and sideslip angle, $\{v_T, \alpha, \beta\}$, may be determined from the absolute velocity of the aircraft, $\{U, V, W\}$, together with the local wind velocity and the aircraft orientation as defined by the roll, pitch, and yaw variables, $\{\phi, \theta, \psi\}$, of the 3-2-1 rotation sequence described above (alternatively, $\{U, V, W\}$ may be determined from $\{v_T, \alpha, \beta\}$ and $\{\phi, \theta, \psi\}$ and the local wind velocity). For the purpose of describing the dynamics of flight, of course, $\{v_T, \alpha, \beta\}$ are the natural variables to consider.

Next, an ODE model for how the state of the aircraft evolves in time must be developed. The process of developing accurate linearized dynamic models of an aircraft in flight is quite involved; this process may be started using simplified aerodynamic models and small-scale wind-tunnel tests, but generally must be subsequently refined using high-fidelity computational fluid dynamics simulations, large-scale wind-tunnel tests, and flight tests. Almost all models of aircraft dynamics today are based on **static stability derivatives**; that is, the forces and moments on the aircraft and the effectiveness of the control surfaces for any given state of the aircraft within its **flight envelope**[15] are determined assuming the aircraft is maintained in equilibrium in this configuration; that is, a *dynamic* model accounting for the unsteadiness of the flow itself is not accounted for with this approach. Certain dynamic maneuvers, such as the so-called **dynamic lift** available right before vortex separation and stall of a rapidly pitching airfoil moving at low speed (e.g., during spot landings with a flapping wing) are thus not accounted for well with such static models of the flow evolution. Nonetheless, a static model of the flow is in fact quite adequate for most fixed wing aircraft throughout most of their flight envelope.

[12] Even though these three rotations are usually not the actual rotations that brought the aircraft into this configuration!

[13] Note that *order matters* (that is, such rotations are **noncommutative**), as the latter steps rotate the aircraft about the body-fitted coördinates only after the former steps are complete. Various alternative rotation sequences may also be used to unambiguously identify the orientation of an aircraft, spacecraft, or other solid body; which rotation sequence is most convenient depends on the application.

[14] Note that the 3-2-1 rotation sequence used here is often casually referred to as an **Euler rotation sequence** though, strictly speaking, an Euler rotation sequence repeats a rotation around one of the axes, a common choice being the **3-1-3 Euler rotation sequence** (in the present setting, yaw, then roll, then yaw again).

[15] A **flight envelope** is the set of states of an aircraft deemed safe for flight.

Linearized dynamic models governing the time evolution of the 12 variables describing an aircraft in cruise approximately decouple[16] into three essentially independent subsystems:

• the **lateral/directional dynamics** of the aircraft model, which relates the **yaw** (a.k.a. **heading angle**) ψ, the **roll** ϕ, the **yaw rate** $r = d\psi/dt$, the **body-axis roll rate** $p = d\phi/dt$, and the **sideslip angle** β,
• the **longitudinal dynamics** of the aircraft model, which relates the **pitch** θ, the **pitch rate** $q = d\theta/dt$, the **angle of attack** α, and the **airspeed** v_T, and
• the **navigation equations** $dX/dt = U$, $dY/dt = V$, $dZ/dt = W$; as mentioned previously, $\{U, V, W\}$ may, via simple geometry, be determined from the orientation angles $\{\phi, \theta, \psi\}$ together with knowledge of the local wind velocity and measurements of the relative wind past the aircraft, $\{v_T, \alpha, \beta\}$.

The navigation equations are straightforward to integrate in time (see §??) to track changes in the vehicle's absolute position in order to navigate; we thus focus our attention below on the more complex problems of the lateral/directional dynamics and the longitudinal dynamics of some representative aircraft.

Defining the deflection of the elevator, aileron, rudder, and throttle from their **trimmed** flight positions as δ_e, δ_a, δ_r, and δ_{th}, respectively, a representative linearized model of the lateral/directional dynamics of a large transport aircraft on approach to landing (see Minto, Chow, & Beseler 1989) is

$$\begin{matrix}\text{yaw:}\\ \text{roll:}\\ \text{yaw rate:}\\ \text{roll rate:}\\ \text{sideslip:}\end{matrix} \quad \frac{d}{dt}\begin{pmatrix}\psi\\ \phi\\ p\\ r\\ \beta\end{pmatrix} = \underbrace{\begin{pmatrix}0 & 0 & 1 & 0 & 0\\ 0 & 0 & .199 & 1 & 0\\ 0 & -.002 & -.194 & -.167 & .748\\ 0 & -.003 & .636 & -2.02 & -5.37\\ 0 & .136 & -.970 & .198 & -.148\end{pmatrix}}_{A_1}\begin{pmatrix}\psi\\ \phi\\ p\\ r\\ \beta\end{pmatrix} + \underbrace{\begin{pmatrix}0 & 0\\ 0 & 0\\ .053 & -.74\\ .865 & .904\\ .002 & .047\end{pmatrix}}_{B_1}\begin{pmatrix}\delta_a\\ \delta_r\end{pmatrix}. \tag{6.33}$$

A representative linearized model of the longitudinal dynamics of a large transport aircraft on approach to landing (see Stevens & Lewis 2003, Example 4.6-4) is

$$\begin{matrix}\text{airspeed:}\\ \text{AOA:}\\ \text{pitch:}\\ \text{pitch rate:}\end{matrix} \quad \frac{d}{dt}\begin{pmatrix}v_T\\ \alpha\\ \theta\\ q\end{pmatrix} = \underbrace{\begin{pmatrix}-.0386 & 19.0 & -32.1 & 0\\ -.00103 & -.633 & .0056 & 1\\ 0 & 0 & 0 & 1\\ -.00008 & -.76 & -.0008 & -.52\end{pmatrix}}_{A_2}\begin{pmatrix}v_T\\ \alpha\\ \theta\\ q\end{pmatrix} + \underbrace{\begin{pmatrix}10 & 0\\ -.00015 & 0\\ 0 & 0\\ .025 & -.011\end{pmatrix}}_{B_2}\begin{pmatrix}\delta_{th}\\ \delta_e\end{pmatrix}. \tag{6.34}$$

A representative linearized model of the longitudinal dynamics of an F-16 in cruise (300 knots at sea level; see Stevens & Lewis 2003, Example 4.4-1) is

$$\begin{matrix}\text{airspeed:}\\ \text{AOA:}\\ \text{pitch:}\\ \text{pitch rate:}\end{matrix} \quad \frac{d}{dt}\begin{pmatrix}v_T\\ \alpha\\ \theta\\ q\end{pmatrix} = \underbrace{\begin{pmatrix}-.0193 & 8.82 & -32.2 & -.575\\ -.000254 & -1.02 & 0 & .905\\ 0 & 0 & 0 & 1\\ 0 & .822 & 0 & -1.08\end{pmatrix}}_{A_3}\begin{pmatrix}v_T\\ \alpha\\ \theta\\ q\end{pmatrix} + \underbrace{\begin{pmatrix}.174\\ -.00215\\ 0\\ -.176\end{pmatrix}}_{B_3}\begin{pmatrix}\delta_e\end{pmatrix}. \tag{6.35}$$

The systems given above are written in the ubiquitous **state-space** form, $d\mathbf{x}/dt = A\mathbf{x} + B\mathbf{u}$, the characterization of which is studied in §??, and the control of which is considered in §??. We will also develop a variety of convenient ways to convert back and forth between first-order state-space forms and **single input, single output** (**SISO**) higher-order ODE forms; note that state space forms have the significant advantage of easily

[16] That is, if the linearized dynamics of these 12 variables is written in the **state-space** form $d\mathbf{x}/dt = A\mathbf{x} + B\mathbf{u}$ with the components of $\mathbf{x}$ appropriately ordered, A may written in a 3×3 block upper-triangular form.

handling **multiple input, multiple output** (**MIMO**) systems. Further, state-space models reveal the inherent *coupling* present as the several states of a system (e.g., yaw, roll, yaw rate, roll rate, and sideslip) evolve in time, which often leads to significant practical insight regarding the physical system (see Exercise **??**). △

6.2 The dynamics of systems of N interacting particles

6.2.1 The principle of least action, and Lagrange's equations

Our first task is to establish from first principles the equations of motion of a mechanical system, starting from axioms A and B above. Following Landau & Lifshitz (1976), the **principle of least action** asserts that the motion of a system from some initial position $\mathbf{q}(t_1)$ to some final position $\mathbf{q}(t_2)$ minimizes an integral

$$S = \int_{t_1}^{t_2} L(\mathbf{q}, \dot{\mathbf{q}}, t)\, dt,$$

where the function $L(\mathbf{q}, \dot{\mathbf{q}}, t)$, called the **Lagrangian** of the system considered, is, so far, unspecified. Starting from this ansatz, we now develop a key equation relating various derivatives of L, assuming that $\mathbf{q}(t)$ is the trajectory from a specified $\mathbf{q}(t_1)$ to a specified $\mathbf{q}(t_2)$ that minimizes the **action integral** S. To proceed, consider an infinitesimal perturbation $\delta\mathbf{q}(t)$ to the trajectory $\mathbf{q}(t)$ such that $\delta\mathbf{q}(t_1) = \delta\mathbf{q}(t_2) = 0$. The modified value of S corresponding this perturbed trajectory is

$$S + \delta S = \int_{t_1}^{t_2} L(\mathbf{q} + \delta\mathbf{q}, \dot{\mathbf{q}} + \delta\dot{\mathbf{q}}, t)\, dt.$$

Assuming the dependence of S on $\mathbf{q}$ is smooth, a necessary condition for $\mathbf{q}(t)$ to minimize S is that the first variation $\delta S = 0$; that is, in summation notation,

$$\delta S = \int_{t_1}^{t_2} \Big(\frac{\partial L}{\partial q_i}\, \delta q_i + \frac{\partial L}{\partial \dot{q}_i}\, \delta \dot{q}_i \Big)\, dt = \int_{t_1}^{t_2} \Big(\frac{\partial L}{\partial q_i} - \frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} \Big)\, \delta q_i \; dt - \Big[\frac{\partial L}{\partial \dot{q}_i} \delta q_i \Big]_{t_1}^{t_2} = 0,$$

where the second expression follows from the first via integration by parts. Noting $\delta q_i(t_1) = \delta q_i(t_2) = 0$, and that the above result holds for any set of infinitesimal perturbations δq_i, we obtain **Lagrange's equation**

$$\boxed{\frac{d}{dt}\Big(\frac{\partial L}{\partial \dot{q}_i}\Big) = \frac{\partial L}{\partial q_i}.} \tag{6.36}$$

Once the Lagrangian L of a system is identified, (6.36) provides the **equation of motion** governing the dynamics of the system, relating the accelerations to the velocities and the coördinates.

The Lagrangian L characterizing the dynamics of a system is not unique. Any constant times L is also a valid Lagrangian for the same system; this happens, e.g., when changing units of measurement. Further, consider two Lagragians that differ by a total time derivative of a function of the coördinates and time:

$$L_2(\mathbf{q}, \dot{\mathbf{q}}, t) = L_1(\mathbf{q}, \dot{\mathbf{q}}, t) + df(\mathbf{q}, t)/dt. \tag{6.37a}$$

The action integral S corresponding to these two Lagrangians are related as follows:

$$S_2 = \int_{t_1}^{t_2} L_2(\mathbf{q}, \dot{\mathbf{q}}, t)\, dt = \int_{t_1}^{t_2} L_1(\mathbf{q}, \dot{\mathbf{q}}, t)\, dt + \int_{t_1}^{t_2} \frac{df}{dt}\, dt = S_1 + f(\mathbf{q}(t_2), t_2) - f(\mathbf{q}(t_1), t_1); \tag{6.37b}$$

that is, S_1 and S_2 differ by an amount which is not affected by variation of the trajectory $\mathbf{q}(t)$ between $\mathbf{q}(t_1)$ and $\mathbf{q}(t_2)$, and thus the Lagrangians L_1 and L_2 characterize the same motion. Note also the Lagrangian of a system C composed of two non-interacting subsystems A and B is additive: $L_C = L_A + L_B$.

An **inertial** reference frame (a.k.a. a **Galilean** reference frame) can always be chosen such that the equations governing motion are[17] homogeneous and isotropic in space, and homogeneous in time; that is, the equations of

[17] **Homegeneous** in this setting means **translation invariant**, whereas **isotropic** means **direction invariant**.

motion don't change if the coördinate system origin is shifted or rotated in space, or if the clock is reset. Note that, in an arbitrary (that is, noninertial) reference frame (e.g., a reference frame that spins), the equations governing motion are inhomogeneous and anisotropic, which is much less convenient.

Thus, in an inertial reference frame, the Lagrangian L of a single particle can not explicitly depend on the position vector[18] $\vec{r}$, the direction of the velocity vector $\vec{v}$, or the time t. Rather, L can only depend on the *magnitude* of the velocity, $v = \|\vec{v}\|$; we may thus write $L = L(v^2)$. In Cartesian coördinates, Lagrange's equation (6.36) thus reduces to $d/dt(\partial L/\partial v) = 0$, and thus $\partial L/\partial v$ is constant. Since $\partial L/\partial v$ depends only on v, it follows that $\vec{v}$ itself is constant for a single free particle, a fact which is known as **Newton's first law**.

Indeed, in classical mechanics, the equations of motion in any two inertial reference frames, one of which may be rotated, translated, and moving uniformly in a straight line with respect to the other, are entirely equal; this important principle is known as **Galileo's relativity principle**; it implies that there is no "one reference frame to rule them all" (cf. Tolkien 1954). The position of a particle in a frame of reference G which moves relative to another frame of reference G' at a constant velocity $\vec{V}$, and the corresponding times in these two different reference frames, are related by the **Galilean transformation**:

$$\vec{r}\,' = \vec{r} + \vec{V}\,t, \qquad t' = t. \tag{6.38}$$

Differentiating the above expression with respect to time, we have $\vec{v}\,' = \vec{v} + \vec{V}$. Assuming $\vec{V} = \vec{\epsilon}$ is small and expanding in powers of $\vec{\epsilon}$, neglecting powers higher than first, we may write, in Cartesian coördinates,

$$L(v'^2) = L(\vec{v}\,' \cdot \vec{v}\,') = L(v^2 + 2\,\vec{v}\cdot\vec{\epsilon} + \vec{\epsilon}\cdot\vec{\epsilon}) \approx L(v^2) + \frac{\partial L}{\partial v^2}\,2\,\vec{v}\cdot\vec{\epsilon}.$$

Since the equations of motion themselves must be the same in the two different frames, by (6.37), $L(v'^2)$ and $L(v^2)$ must differ at most by a function that may be written $df(\vec{r},t)/dt$. The last term on the RHS above may be written in this form only if it is linear in $\vec{v}$. Therefore, $\partial L/\partial v^2$ is independent of v, and L is proportional to v^2; we thus write $L = mv^2/2$ (this step, in fact, may be said to be that which *defines* the mass m, with $m > 0$). Indeed, even if $\vec{V}$ is not small, $L(v'^2)$ and $L(v^2)$ differ only by a function that may be written $df(\vec{r},t)/dt$:

$$L(v'^2) = \frac{1}{2}m\,v'^2 = \frac{1}{2}\,m\,\|\vec{v}+\vec{V}\|^2 = \frac{1}{2}\,m\,v^2 + m\,\vec{v}\cdot\vec{V} + \frac{1}{2}\,m\,V^2 = L(v^2) + \frac{d}{dt}(m\,\vec{r}\cdot\vec{V} + m\,V^2\,t/2). \tag{6.39}$$

By the additive property mentioned previously, the Lagrangian of a system of noninteracting particles is thus given, in Cartesian coördinates, by

$$L = \sum\nolimits_a m_a\,\|\vec{v}_a\|^2/2. \tag{6.40}$$

For a **system of N interacting particles**, a term is added to the Lagrangian to model their interaction: in Cartesian coördinates,

$$\boxed{L = \sum\nolimits_a m_a\,\|\vec{v}_a\|^2/2 - U(\mathbf{r}) = T(\mathbf{v}) - U(\mathbf{r})} \tag{6.41}$$

with $\mathbf{v} = \{\vec{v}_1, \vec{v}_2, \ldots\}$ and $\mathbf{r} = \{\vec{r}_1, \vec{r}_2, \ldots\}$, where $T(\mathbf{v})$ is called the **kinetic energy** and $U(\mathbf{r})$ the **potential energy**. Note that $T(\mathbf{v})$ is quadratic in $\mathbf{v}$. The form of $U(\mathbf{r})$ is problem specific; as the distance between each pair of particles gets large, U approaches an arbitrary constant (usually taken as zero), and (6.40) is recovered. Given the form of L in (6.41), in Cartesian coördinates, the equations of motion may be derived from (6.36):

$$\frac{d}{dt}\Big(\frac{\partial L}{\partial \vec{v}_a}\Big) = \frac{\partial L}{\partial \vec{r}_a} \quad \Rightarrow \quad \boxed{m_a\frac{d\vec{v}_a}{dt} = -\frac{\partial U}{\partial \vec{r}_a} \triangleq \vec{f}_a.} \tag{6.42}$$

[18] Reminder: see footnote 1 on page 6-1 regarding the disambiguating notation used for vectors in $\mathbb{R}^3$ in this chapter.

The term $\vec{f}_a$, which depends only on the coördinates of the particles, is called the **force** on the a'th particle, and (6.42) is known as **Newton's second law**.

One example of a system of N interacting particles is given by **Newton's law of universal gravitation**:

$$U = -\sum_a \sum_{b \neq a} \frac{G\, m_a\, m_b}{\|\vec{r}_b - \vec{r}_a\|} \quad \text{and} \quad \vec{f}_a = -\frac{\partial U}{\partial \vec{r}_a} = \sum_{b \neq a} \frac{G\, m_a\, m_b (\vec{r}_b - \vec{r}_a)}{\|\vec{r}_b - \vec{r}_a\|^3}, \tag{6.43}$$

where $G = 6.67384 \times 10^{-11}$ N m^2 / kg^2; this force is attractive between bodies. Another example is given by **Coulomb's law** between charged bodies:

$$U = \sum_a \sum_{b \neq a} \frac{k_e\, q_a\, q_b}{\|\vec{r}_b - \vec{r}_a\|} \quad \text{and} \quad \vec{f}_a = -\frac{\partial U}{\partial \vec{r}_a} = -\sum_{b \neq a} \frac{k_e\, q_a\, q_b (\vec{r}_b - \vec{r}_a)}{\|\vec{r}_b - \vec{r}_a\|^3}, \tag{6.44}$$

where the charges q_a and q_b are measured in coulombs (note that 1 coulomb is the charge of 6.24151×10^{18} protons), and $k_e = 8.98755 \times 10^9$ N m^2 / C^2; this force is attractive between bodies of opposite charge, and repulsive between bodies of like charge.

For a **system of N interacting particles in generalized coördinates**, writing $\vec{r}_a = \vec{r}_a(\mathbf{q})$, it follows that $\vec{v}_a = \sum_i (\partial \vec{r}_a(\mathbf{q})/\partial q_i)\, \dot{q}_i$. Substituting this expression into (6.41), it follows that

$$\boxed{L = \sum\nolimits_{i,k} a_{ik}(\mathbf{q})\, \dot{q}_i\, \dot{q}_k/2 - U(\mathbf{q}) = T(\mathbf{q}, \dot{\mathbf{q}}) - U(\mathbf{q}).} \tag{6.45}$$

Note that $T(\mathbf{q}, \dot{\mathbf{q}})$ is quadratic in $\dot{\mathbf{q}}$, though the coefficients a_{ik} are, in general, functions of $\mathbf{q}$.

Consider a closed system C with two interacting subsystems, denoted A and B. By (6.45), we may write $L_C = T_A(\mathbf{q}_A, \dot{\mathbf{q}}_A) + T_B(\mathbf{q}_B, \dot{\mathbf{q}}_B) - U(\mathbf{q}_A, \mathbf{q}_B)$. If subsystem B (e.g., the Earth) is much bigger than subsystem A (e.g., a rocket), then $\mathbf{q}_B$ may be considered as a specified function of time, as the motion of subsystem B is effectively independent of the motion of subsystem A. When deriving the motion of subsystem A, referred to as an **open system**, we may thus consider the simplified Lagrangian

$$L_A(\mathbf{q}_A, \dot{\mathbf{q}}_A, t) = T_A(\mathbf{q}_A, \dot{\mathbf{q}}_A) - U(\mathbf{q}_A, \mathbf{q}_B(t)), \tag{6.46}$$

where $\mathbf{q}_B(t)$ is a specified function of time, and thus L_A also depends explicitly on time.

Considering subsystem A in an open system as described above as a single particle a (and, for simplicity, taking subsystem B as stationary), and performing a path integral of (6.42) from $\vec{r}_a^{\,1}$ to $\vec{r}_a^{\,2}$, leads to

$$U(\vec{r}_a^{\,2}) - U(\vec{r}_a^{\,1}) = \int_{\vec{r}_a^{\,1}}^{\vec{r}_a^{\,2}} (-\vec{f}_a) \cdot d\vec{r}_a; \tag{6.47}$$

that is, the change in the potential energy of the particle over the path considered is precisely the **work** (i.e., the integral of the force overcome over the distance travelled) required to move the particle from $\vec{r}_a^{\,1}$ to $\vec{r}_a^{\,2}$.

As an even simpler example, let $\vec{r}$ denote the position of an object of mass m in Cartesian coördinates in the lab, where r_3 measures the height of the object above the floor, and let $\vec{R} = \{0, 0, -R\}$ denote the position of the center of the earth (which is independent of the motion of subsystem A), noting that $R = 6.371 \times 10^6$ m and that the mass of the Earth is $M = 5.972 \times 10^{24}$ kg. It follows from (6.43), taking each object as acting on the other like a particle, and noting (B.90), that the force $\vec{f}$ on the object, and its potential energy U, are

$$\vec{f} = \frac{G\, m\, M(\vec{R} - \vec{r})}{\|\vec{R} - \vec{r}\|^3} \approx -m\, g\, \vec{e}^{\,3}, \quad U = -\frac{G\, m\, M}{\|\vec{R} - \vec{r}\|} = -\frac{G\, m\, M/R}{\|R + r_3\|/R} \approx -m\, g\, R\Big(1 - \frac{r_3}{R}\Big) = C - \vec{f} \cdot \vec{r}, \tag{6.48}$$

where $g = G\, M/R^2 \approx 9.8$ m / sec^2, $\vec{e}^{\,3}$ is a unit vector pointed up, and C is constant and may thus be ignored. Note that $\vec{f}$ is essentially constant everywhere in the lab (that is, the gravitational field over this domain is **uniform**), and that the vector from the center of the earth to the object is aligned in the $\vec{e}^{\,3}$ direction; thus, the expression for U in (6.48) is a special case of the more general expression given in (6.47).

6.2.2 Consequences of the homogeneity and isotropy of space and time

6.2.2.1 Conservation of momentum, and the center of mass

Due to the **homogeneity of space**, the Lagrangian of a closed system is unchanged by a (fixed) infinitesimal displacement of the entire system in space, $\delta\vec{r}_a = \vec{\epsilon}\,\forall a$. As $\vec{\epsilon}$ is arbitrary, noting (6.41)-(6.42), we have

$$L(\mathbf{r},\mathbf{v}) = L(\mathbf{r}+\delta\mathbf{r},\mathbf{v}) = L(\mathbf{r},\mathbf{v}) + \sum_a \frac{\partial L}{\partial \vec{r}_a}\cdot\delta\vec{r}_a = L(\mathbf{r},\mathbf{v}) + \vec{\epsilon}\cdot\sum_a \frac{\partial L}{\partial \vec{r}_a} \quad\Rightarrow\quad \sum_a \frac{\partial L}{\partial \vec{r}_a} = \sum_a \vec{f}_a = 0.$$

[In the special case of two particles, it is thus seen that $\vec{f}_1 = -\vec{f}_2$ (that is, the forces are equal in magnitude and opposite in direction); this is known as **Newton's third law**.] It thus follows from (6.36) that

$$\sum_a \frac{d}{dt}\frac{\partial L}{\partial \vec{v}_a} = 0 \quad\Rightarrow\quad \boxed{\frac{d\vec{P}}{dt} = 0,} \tag{6.49}$$

where $\vec{P} = \sum_a \partial L/\partial\vec{v}_a$. Noting (6.45), and defining the **momentum of each particle** $\vec{p}_a = m_a\,\vec{v}_a$, it follows that the **total momentum** $\vec{P} = \sum_a m_a\,\vec{v}_a = \sum_a \vec{p}_a$ of a closed system is conserved.

Define the **total mass** of the system $\mu = \sum_a m_a$ and the **center of mass** $\vec{R} = \sum_a m_a\vec{r}_a/\mu$, and note by (6.38) that the velocity of a particle in a frame of reference G which moves relative to another frame of reference G' at a constant velocity $-\vec{V}$ is related by $\vec{v}' = \vec{v} - \vec{V}$. The momentum in reference frame G' is thus given by $\vec{P}' = \sum_a m_a\,\vec{v}'_a = \vec{P} - \mu\vec{V}$. Selecting $\vec{V} = \vec{P}/\mu = \sum_a m_a\,\vec{v}_a/\mu$, the total system is said to be **at rest** in reference frame G', in which the total momentum $\vec{P}' = 0$ (and, thus, the center of mass is stationary), and the total system is said to be moving at velocity $\vec{V}$ in reference frame G, in which the total momentum is $\vec{P} = \mu\vec{V}$.

6.2.2.2 Conservation of energy

Due to the **homogeneity of time**, the Lagrangian of closed systems does depend explicitly on time. Indeed, as discussed above, even in open systems for which the external field is constant (that is, if the subsystem B is stationary), the Lagrangian does not depend explicitly on time. In either case, noting (6.36), we may write

$$\frac{dL}{dt} = \sum_i \frac{\partial L}{\partial q_i}\dot{q}_i + \sum_i \frac{\partial L}{\partial \dot{q}_i}\ddot{q}_i = \sum_i \dot{q}_i\frac{d}{dt}\Big(\frac{\partial L}{\partial \dot{q}_i}\Big) + \sum_i \frac{\partial L}{\partial \dot{q}_i}\ddot{q}_i = \frac{d}{dt}\Big(\sum_i \dot{q}_i\frac{\partial L}{\partial \dot{q}_i}\Big) \quad\Rightarrow\quad \frac{d}{dt}\Big(\sum_i \dot{q}_i\frac{\partial L}{\partial \dot{q}_i} - L\Big) = 0.$$

Note from (6.45) that $L = T(\mathbf{q},\dot{\mathbf{q}}) - U(\mathbf{q})$ where T is quadratic in $\dot{\mathbf{q}}$; it follows that

$$\sum_i \dot{q}_i\frac{\partial L}{\partial \dot{q}_i} = \sum_i \dot{q}_i\frac{\partial T}{\partial \dot{q}_i} = 2T \quad\Rightarrow\quad \boxed{\frac{dE}{dt} = 0,} \tag{6.50}$$

where $E = T(\mathbf{q},\dot{\mathbf{q}}) + U(\mathbf{q})$ in generalized coördinates, or $E = T(\mathbf{v}) + U(\mathbf{r})$ in Cartesian coördinates. That is, in such **conservative systems**, the **total energy** E (kinetic energy *plus* potential energy) is conserved.

Also, using the definitions of μ, $\vec{V}$, G, and the rest frame G' given in §6.2.2.1, noting the definition of $T(\mathbf{v})$ in (6.41) and that $\vec{v}_a = \vec{V} + \vec{v}'_a$, the total energy E derived above, in the G frame, may be written

$$E = \frac{1}{2}\sum_a m_a\,(\vec{V}+\vec{v}'_a)^2 + U(\mathbf{r}) = \frac{1}{2}\mu V^2 + \vec{V}\cdot\cancelto{0}{\sum_a m_a\,(\vec{v}'_a)} + \frac{1}{2}\sum_a m_a\,(\vec{v}'_a)^2 + U(\mathbf{r}') = \frac{1}{2}\mu V^2 + E',$$

where $\mu V^2/2$ is the energy due to the motion of the center, and E' is the **internal energy** of the system in the frame G', where the center of mass is at rest.

6.2.2.3 Conservation of angular momentum

Due to the **isotropy of space**, the Lagrangian $L(\mathbf{r}, \mathbf{v})$ of a closed system is unchanged by a (fixed) infinitesimal rotation of the entire system in space. Denote this rotation by the vector $\delta\vec{\phi}$, where the magnitude of this vector is the (infinitesimal) angle of rotation $\delta\phi$, and the direction of this vector is the axis of rotation, using the right-hand rule. For the radius vector $\vec{r}_a$ from the origin to each particle in the system, the increment $\delta\vec{r}_a$ due to this rotation is $\delta\vec{r}_a = \delta\vec{\phi} \times \vec{r}_a$, and thus $\delta\vec{v}_a = \delta\vec{\phi} \times \vec{v}_a$. Noting from §6.2.2.1 that $\partial L/\partial\vec{v}_a = \vec{p}_a$, from (6.42) that $\partial L/\partial\vec{r}_a = \dot{\vec{p}}_a$, applying (B.20), and noting that the infinitesimal variation $\delta\vec{\phi}$ is arbitrary, we have

$$\begin{aligned}\delta L &= \sum_a \left(\frac{\partial L}{\partial \vec{r}_a} \cdot \delta\vec{r}_a + \frac{\partial L}{\partial \vec{v}_a} \cdot \delta\vec{v}_a\right) = \sum_a \left(\dot{\vec{p}}_a \cdot \delta\vec{\phi} \times \vec{r}_a + \vec{p}_a \cdot \delta\vec{\phi} \times \vec{v}_a\right) \\ &= \delta\vec{\phi} \cdot \sum_a \left(\vec{r}_a \times \dot{\vec{p}}_a + \vec{v}_a \times \vec{p}_a\right) = \delta\vec{\phi} \cdot \frac{d}{dt} \sum_a \vec{r}_a \times \vec{p}_a = 0 \quad \Rightarrow \quad \boxed{\frac{d\vec{M}}{dt} = 0,}\end{aligned} \tag{6.51}$$

that is, the **total angular momentum** $\vec{M} = \sum_a \vec{r}_a \times \vec{p}_a$ of a closed system is conserved.

Also, using the definitions of μ, $\vec{R}$, $\vec{V}$, $\vec{P}$, G, and the rest frame G' given in §6.2.2.1, assuming the origins of the reference frames G and G' coincide at the instant considered, the total angular momentum $\vec{M}$ derived above, in reference frame G, may be written

$$\vec{M} = \sum_a m_a \vec{r}_a \times \vec{v}_a = \sum_a m_a \vec{r}_a \times \vec{V} + \sum_a m_a \vec{r}_a \times \vec{v}'_a = \mu\vec{R} \times \vec{V} + \vec{M}' = \vec{R} \times \vec{P} + \vec{M}',$$

where $\vec{R} \times \vec{P}$ is the angular momentum due to the motion of the center, and $\vec{M}'$ is the **intrinsic angular momentum** of the system in the frame G', where the center of mass is at rest.

6.2.2.4 Reversibility of trajectories

Due to the **isotropy of time**, the Lagrangian of a closed system is unchanged by a reversal of the trajectories of the system in time. That is, in the purest form of Lagrangian mechanics, there are no losses of energy (for example, to heat or sound), and thus trajectories are reversible. Methods to generalize this setting to account for frictional losses are discussed in §6.4.4.

6.2.3 Hamiltonian and Routhian formulations†

Define the **generalized momenta** $p_i = \partial L/\partial\dot{q}_i$ and the **Hamiltonian** $H = \sum_i \dot{q}_i p_i - L$. Consider now the variation of $L(q_i, \dot{q}_i, t)$ arising from an arbitrary infinitesimal variation of its arguments:

$$\begin{aligned}\delta L &= \sum_i \left(\frac{\partial L}{\partial q_i}\delta q_i + \frac{\partial L}{\partial \dot{q}_i}\delta\dot{q}_i\right) + \frac{\partial L}{\partial t}\delta t = \sum_i \left(\frac{\partial L}{\partial q_i}\delta q_i + p_i\delta\dot{q}_i\right) + \frac{\partial L}{\partial t}\delta t \\ &= \sum_i \left(\frac{\partial L}{\partial q_i}\delta q_i + \delta(p_i\dot{q}_i) - \dot{q}_i\delta p_i\right) + \frac{\partial L}{\partial t}\delta t \quad \Rightarrow \quad \delta H = \sum_i \left(-\frac{\partial L}{\partial q_i}\delta q_i + \dot{q}_i\delta p_i\right) - \frac{\partial L}{\partial t}\delta t.\end{aligned}$$

Consider also the variation of $H(q_i, p_i, t)$ arising from an arbitrary infinitesimal variation of its arguments:

$$\delta H = \sum_i \left(\frac{\partial H}{\partial q_i}\delta q_i + \frac{\partial H}{\partial p_i}\delta p_i\right) + \frac{\partial H}{\partial t}\delta t.$$

Setting δH equal in the two previous expressions, for arbitrary δq_i, δp_i, and δt, results in:

$$\frac{\partial H}{\partial q_i} = -\frac{\partial L}{\partial q_i}, \qquad \frac{\partial H}{\partial p_i} = \dot{q}_i, \qquad \frac{\partial H}{\partial t} = -\frac{\partial L}{\partial t}.$$

Finally, incorporating Lagrange's equation (6.36) with the definition of p_i above, it follows that $\partial L/\partial q_i = \dot{p}_i$, thus leading to **Hamilton's equations**

$$\boxed{\frac{d\mathbf{p}}{dt} = -\frac{\partial H}{\partial \mathbf{q}}, \qquad \frac{d\mathbf{q}}{dt} = \frac{\partial H}{\partial \mathbf{p}}, \qquad \frac{\partial H}{\partial t} = -\frac{\partial L}{\partial t}.} \tag{6.52}$$

For a closed system, $\partial L/\partial t = 0$, and thus, noting (6.50) and (6.45), H is simply the (conserved) total energy of the system, $H = 2T - L = T + U = E$. Writing a system in this **symplectic form** leads to numerical advantages in the long-time integration of such conservative systems, as discussed in §??. To recap, the equations of motion in the Lagrangian approach, given in (6.36), are fundamentally second order, whereas the equations of motion in the Hamiltonian approach, given in (6.52), are fundamentally first order and exhibit special structure which may be exploited when performing long-time integration of conservative systems.

A hybrid approach, called the **Routhian** formulation, develops the evolution equations for the coördinates explicitly appearing in the Lagrangian in the (second-order) Lagrangian manner, and develops the evolution equations for the coördinates not explicitly appearing in the Lagrangian (called **cyclic coördinates**) in the (first-order) Hamiltonian manner. In certain problems with many coördinates, some of which are cyclic, this approach can significantly simplify the resulting computations required to march the system in time.

6.3 Solid bodies and their kinematics

We now turn to the defintion of a **solid body**, and characterize its mass distribution, orientation, and rate of rotation. Once this subject is well at hand, the dynamics of solid bodies is considered in §6.4. Much of the machinery of §6.2 carries over directly to this discussion as, for the purpose of derivation, solid bodies may be considered simply as a cloud of particles constrained to move together.

6.3.1 Description of a solid body and its mass distribution

Consider first a cloud of particles rigidly connected by, in effect, massless rods; the **total mass** μ, **center of mass** $\vec{R}$, and **inertial tensor** $I_{ik} \triangleq \sum_a m_a(r_j^2\delta_{ik} - r_i r_k)_a$ (where the sum is taken over each particle a) of this cloud of particles are defined (denoting r_1 as x, r_2 as y, and r_3 as z) by:

$$\mu = \sum_a m_a, \quad \vec{R} = \sum_a \frac{m_a \vec{r}_a}{\mu}, \quad I = \begin{pmatrix} \sum_a m_a(y_a^2+z_a^2) & -\sum_a m_a\, x_a\, y_a & -\sum_a m_a\, x_a\, z_a \\ -\sum_a m_a\, y_a\, x_a & \sum_a m_a(x_a^2+z_a^2) & -\sum_a m_a\, y_a\, z_a \\ -\sum_a m_a\, z_a\, x_a & -\sum_a m_a\, z_a\, y_a & \sum m_a(x_a^2+y_a^2) \end{pmatrix}. \tag{6.53a}$$

Passing to the limit of an infinite number of infinitesimal particles, a **solid body** is characterized by:

$$\mu = \int_\Omega \rho\, dV, \quad \vec{R} = \int_\Omega \frac{\rho\,\vec{r}}{\mu}\, dV, \quad I = \begin{pmatrix} \int_\Omega \rho(y^2+z^2)\, dV & -\int_\Omega \rho\, x\, y\, dV & -\int_\Omega \rho\, x\, z\, dV \\ -\int_\Omega \rho\, y\, x\, dV & \int_\Omega \rho(x^2+z^2)\, dV & -\int_\Omega \rho\, y\, z\, dV \\ -\int_\Omega \rho\, z\, x\, dV & -\int_\Omega \rho\, z\, y\, dV & \int_\Omega \rho(x^2+y^2)\, dV \end{pmatrix}. \tag{6.53b}$$

In the sections that follow, it is shown that the equations of motion of any solid body are built on these simple aggregate functions of its mass distribution. It follows from (6.53) that, if the origin is shifted such that $\vec{r}' = \vec{r}+\vec{s}$, then $I'_{ik} = I_{ik} + \mu(s^2\delta_{ik} - s_i s_k)$. Note also that, by construction, the inertial tensor is symmetric positive semi-definite, $I \geq 0$. Thus, by Fact **??**, its eigenvalues are non-negative, its eigenvectors may chosen to be orthonormal, and we may decompose the inertial tensor as $I = S\Lambda S^H$. The three eigenvalues of I, denoted $\{I_1, I_2, I_3\}$ and usually ordered $I_1 \geq I_2 \geq I_3 \geq 0$, are known as the **principal moments of inertia** of the body, and the corresponding eigenvectors identify, in the initial reference frame considered, the **principal axes** of the body. The equations of motion of a body will simplify significantly when considered in these coördinates. Note that $I_1 \leq I_2 + I_3$. The following names and properties are associated with solid bodies:

(a) the case with $I_1 = I_2 = I_3$ is called **spherical top** (e.g., a European football),

(b) the case with $I_1 = I_2 > I_3$ is called an **elongated symmetric top** (e.g., an American football),

- further, the limit of case (b) with all particles **colinear** (a.k.a. a **rotator**, with $r_1 = r_2 = 0$) has $I_3 = 0$,

(c) the case with $I_1 > I_2 = I_3$ is called a **flattened symmetric top** (e.g., a frisbee), and

(d) the case with $I_1 > I_2 > I_3$ is called an **asymmetric top** (e.g., a textbook or cellphone),

- further, the limit of case (c) or (d) with all particles **coplanar** (with $r_1 = 0$) has $I_1 = I_2 + I_3$.

To describe a solid body's orientation, we use two orthogonal frames of reference, an **inertial** (a.k.a. **Galilean**) frame $\{\mathbf{i}, \mathbf{j}, \mathbf{k}\}$ that is non-accelerating and non-rotating, and a **Body** frame $\{x, y, z\}$ fixed to the solid body (usually with its origin at the center of mass, and often with its axes aligned with the principal axes of the body), translating and rotating with the solid body itself. Right-handed coördinate systems are used everywhere. A solid body has six degrees of freedom: three to describe the location of its center of mass, and three to describe its orientation as a 3D rotation (see §6.3.2) from some reference orientation.

6.3.2 3D Rotations

The orientation of a solid body may be defined by a real orthogonal 3×3 matrix B whose columns specify the orientation of the principle axes of the body frame within the inertial frame. A finite **3D rotation** from any orientation B_1 to any other orientation B_2 may be related by a real, orthogonal **rotation matrix** R such that $B_2 = R\,B_1$ (that is, $R = B_2\,B_1^T$); via this rotation, any vector $\vec{p}$ affixed to a point on the body maps to a corresponding vector $\vec{p}\,'$ such that $\vec{p}\,' = R\vec{p}$. It follows that $R\,R^T = I$ (i.e., R must also be orthogonal), which imposes six constraints on the nine components of R, and thus R has three degrees of freedom; taking the determinant (see §??) of this expression, it is seen that $|R| = \pm 1$. The case with $|R| = 1$ is called a **proper rotation**; the 3D Givens rotation matrix G (see §??), which performs a rotation in a single coördinate plane and is formed as 2×2 rotation matrix embedded within a 3×3 identity matrix, is a special case. The case with $|R| = -1$ is called an **improper rotation**, and can be formed as the product of a proper rotation matrix with a real Householder reflector matrix H (see §??) with $|H| = -1$; as it is generally not possible to "reflect" a solid body through itself, we restrict our attention to proper rotations with $|R| = 1$.

As shown in §6.3.2.1, any proper rotation R may be expressed as a single rotation of they body by some angle θ about some unit vector $\vec{u}$ via **Rodrigues' rotation formula**; this may be represented as a single vector in the direction of $\vec{u}$ of length θ or, as shown in §6.3.2.2, as a single unit vector in $\mathbb{R}^4$, interpreted as a **unit quaternion**. Alternatively, as shown in §6.3.2.3, any rotation of a body may be represented as a sequence of three distinct rotations of the body around it's own body-fitted axes, called an **Euler** or **Tait-Bryan rotation sequence**. All of these representations of a rotation have exactly three degrees of freedom.

6.3.2.1 Euler's rotation theorem and Rodrigues' rotation formula

Fact 6.1 (Euler's rotation theorem) *Any orientation of a 3D solid body can be expressed as a single rotation of the body by a certain angle around a certain unit vector from a reference orientation.*

Proof: By the properties of the determinant (see §??), it follows for any proper rotation matrix R that

$$|I-R| = |(I-R)^T| = |I-R^T| = |I-R^{-1}| = |-R^{-1}(I-R)| = -|R^{-1}|\,|I-R| = -|I-R| \quad \Rightarrow \quad |I-R| = 0.$$

Thus, $|\lambda_1 I - R| = 0$ for the eigenvalue $\lambda_1 = 1$, and $R\,\vec{s}^{\,1} = \lambda_1 \vec{s}^{\,1} = \vec{s}^{\,1}$ for the corresponding eigenvector $\vec{s}^{\,1}$; $\vec{s}^{\,1}$ is identified as the **rotation axis** of R, as any vector in this direction is unchanged via premultiplication by R.

For any eigenvalue/eigenvector pair $\{\lambda, \vec{s}\}$ of the orthogonal matrix R, we have $\vec{s}^{\,H}\vec{s} = \vec{s}^{\,H}R^H R\,\vec{s} = |\lambda|^2\,\vec{s}^{\,H}\vec{s}$, and thus $|\lambda| = 1$. Since R is orthogonal, $|R| = \lambda_1 \cdot \lambda_2 \cdot \lambda_3 = 1$. Since R is real, $\{\lambda_2, \lambda_3\}$ are either real, or come as a complex-conjugate pair; if they are real, it follows that $\lambda_2 = \lambda_3 = 1$, or $\lambda_2 = \lambda_3 = -1$. In either case, we may write $\lambda(R) = \{1, c + s\mathrm{i}, c - s\mathrm{i}\}$ where $c^2 + s^2 = 1$, with c and s real. The most general way to achieve this is by taking $c = \cos\theta$ and $s = \sin\theta$ for some angle θ. Writing the real Schur decomposition of R (see §??), it follows that

$$R = U\hat{T}U^T, \quad \hat{T} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{pmatrix}, \quad U\,U^T = I, \quad U = \begin{pmatrix} | & | & | \\ \vec{s}^{\,1} & \vec{u}^{\,2} & \vec{u}^{\,3} \\ | & | & | \end{pmatrix}.$$

It is seen that $\hat{T}$ is just a Givens rotation matrix; that is, in the coordinate system defined by the (orthonormal) columns of U, any rotation matrix R has a rotation axis $\vec{s}^{\,1}$, and represents a regular 2D rotation by the angle θ in the plane $\vec{u}^{\,2}$-$\vec{u}^{\,3}$. □

Fact 6.2 (Rodrigues' rotation formula) *Given a real 3D vector $\vec{p}$, define $\vec{p}\,'$ as the rotation of $\vec{p}$ about a unit vector $\vec{u}$ by an angle θ (counterclockwise positive, according to the right-hand rule); $\vec{p}\,'$ is given by*

$$\vec{p}\,' = \vec{p}\cos\theta + (\vec{u}\cdot\vec{p})\,\vec{u}\,(1-\cos\theta) + (\vec{u}\times\vec{p})\sin\theta. \tag{6.54}$$

Proof: Decompose $\vec{p} = \vec{p}_{\parallel} + \vec{p}_{\perp}$ into components parallel and perpendicular to $\vec{u}$:

$$\vec{p}_{\parallel} = (\vec{u} \cdot \vec{p})\,\vec{u}, \quad \vec{p}_{\perp} = \vec{p} - \vec{p}_{\parallel} = \vec{p} - (\vec{u} \cdot \vec{p})\,\vec{u}.$$

The $\vec{p}_{\parallel}$ component of $\vec{p}$ is unaffected by the rotation. Define $\vec{w} = \vec{u} \times \vec{p}_{\perp} = \vec{u} \times (\vec{p}_{\parallel} + \vec{p}_{\perp}) = \vec{u} \times \vec{p}$; since $\vec{u}$ is a unit vector, $\vec{w}$ represents a $90°$ clockwise rotation of $\vec{p}_{\perp}$ around $\vec{u}$. Thus,

$$\begin{aligned}\vec{p}' &= \vec{p}_{\parallel} + \vec{p}_{\perp}\cos\theta + \vec{w}\,\sin\theta \\ &= (\vec{u} \cdot \vec{p})\,\vec{u} + (\vec{p} - (\vec{u} \cdot \vec{p})\,\vec{u})\cos\theta + \vec{u} \times \vec{p}\,\sin\theta = \vec{p}\cos\theta + (\vec{u} \cdot \vec{p})\,\vec{u}\,(1 - \cos\theta) + \vec{u} \times \vec{p}\,\sin\theta.\end{aligned}$$ □

Noting the definition of $\left[\vec{u}\right]_{\times}$ in (B.18), Rodrigues' rotation formula may be written in matrix form as

$$\vec{p}' = R\,\vec{p} \quad \text{where} \quad R = I\,\cos\theta + (1 - \cos\theta)\,\vec{u}\,\vec{u}^{\,T} + \sin\theta\left[\vec{u}\right]_{\times}. \tag{6.55}$$

6.3.2.2 Quaternions

In the early 1700s, Leonhard Euler established the theory of complex numbers, as summarized in §B.1. Starting with the construct $\mathrm{i} = \sqrt{-1}$, complex numbers $z = a + b\,\mathrm{i}$ are said to have a real part a and an imaginary part b. When plotting $z = a + b\,\mathrm{i}$ in the "complex plane", the real dimension is taken as horizontal and the imaginary dimension as vertical; in a sense, in the expression $z = a + b\,\mathrm{i}$, i is a unit vector in the imaginary dimension, and the unit vector in the real dimension is only implied. The product of two complex numbers treats i like an ordinary algebraic variable, noting that $\mathrm{i}^2 = -1$. For example, if we take $q = c + s\,\mathrm{i}$ and $z = a + b\,\mathrm{i}$, the product $q\,z = (c + s\,\mathrm{i})(a + b\,\mathrm{i}) = (ca - sb) + (cb + sa)\mathrm{i}$; note in particular that complex arithmetic is commutative (that is, $q\,z = z\,q$). Using this definition of complex arithmetic, Euler identified that

$$e^{\mathrm{i}\phi} = \cos\phi + \mathrm{i}\,\sin\phi = c + s\,\mathrm{i} \tag{6.56}$$

Fact 6.3 (Rotation using complex numbers) *Taking $q = e^{i\phi} = c + s\,i$ for some angle ϕ, any complex number $z = a + b\,i$ may be rotated counterclockwise by ϕ in the complex plane by taking the product*

$$z' = q\,z = (c + s\,i)(a + b\,i) = (ca - sb) + (cb + sa)i = a' + b'\,i \tag{6.57}$$

Proof: Follows from the definition of 2D vector rotation $\mathbf{z}' = G^T\mathbf{z}$ (§??), with $\begin{pmatrix} a' \\ b' \end{pmatrix} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix}$. □

In a flash of inspiration[19] in 1843, Sir William Hamilton extended Euler's definitions of complex numbers by defining *three* distinct square roots[20] of -1, and the following noncommutative relations between them:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{i}\,\mathbf{j}\,\mathbf{k} = -1 \quad \Rightarrow \quad \mathbf{i}\,\mathbf{j} = -\mathbf{j}\,\mathbf{i} = \mathbf{k}, \;\; \mathbf{j}\,\mathbf{k} = -\mathbf{k}\,\mathbf{j} = \mathbf{i}, \;\; \mathbf{k}\,\mathbf{i} = -\mathbf{i}\,\mathbf{k} = \mathbf{j}. \tag{6.58}$$

A quaternion is a 4D generalization of a complex number, with a real part and three imaginary parts. The **Hamilton product** of two quaternions $\mathbf{p} = p_0 + p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k}$ and $\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$ treats $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ like noncommutative algebraic variables, noting (6.58). Each of the four components of the resulting vector $\mathbf{r} = \mathbf{p}\,\mathbf{q} = r_0 + r_1\mathbf{i} + r_2\mathbf{j} + r_3\mathbf{k}$ has four terms, as summarized by the following equivalent matrix forms:

$$\mathbf{r} = \mathbf{p}\,\mathbf{q} = \begin{pmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{pmatrix} = \begin{pmatrix} p_0 & -p_1 & -p_2 & -p_3 \\ p_1 & p_0 & -p_3 & p_2 \\ p_2 & p_3 & p_0 & -p_1 \\ p_3 & -p_2 & p_1 & p_0 \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}. \tag{6.59a}$$

Denoting $\mathbf{p} = p_0 + \vec{p}$ and $\mathbf{q} = q_0 + \vec{q}$ where $\vec{p} = p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k}$ and $\vec{q} = q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$, we may also write

$$\mathbf{p}\,\mathbf{q} = (p_0 + \vec{p})(q_0 + \vec{q}) = (p_0 q_0 - \vec{p}\cdot\vec{q}) + (p_0\vec{q} + q_0\vec{p} + \vec{p}\times\vec{q}), \tag{6.59b}$$

where $\vec{p}\cdot\vec{q}$ and $\vec{p}\times\vec{q}$ denote 3D dot and cross products (see §B.3). In particular, it follows from (6.59b) that

$$\mathbf{p}\,\mathbf{q} = -\vec{p}\cdot\vec{q} + \vec{p}\times\vec{q} \quad \text{if } p_0 = q_0 = 0. \tag{6.59c}$$

The Hamilton product is noncommutative [that is, $\mathbf{p}\,\mathbf{q} \neq \mathbf{q}\,\mathbf{p}$; note, e.g., the cross products in (6.59b) and (6.59c), and the fact that $\vec{p}\times\vec{q} = -\vec{q}\times\vec{p}$]. Further, akin to (6.56), it follows that

$$e^{\vec{u}\,\phi} = e^{(u_1\mathbf{i}+u_2\mathbf{j}+u_3\mathbf{k})\,\phi} = \cos\phi + (u_1\mathbf{i} + u_2\mathbf{j} + u_3\mathbf{k})\sin\phi \triangleq \mathbf{q}. \tag{6.60}$$

The **conjugate** of a quaternion $\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$ is defined as $\mathbf{q}^* = q_0 - q_1\mathbf{i} - q_2\mathbf{j} - q_3\mathbf{k}$. Thus,

$$(\mathbf{q}\mathbf{p})^* = \mathbf{p}^*\mathbf{q}^* \;\; (6.61a) \qquad (\mathbf{q}^*)^* = \mathbf{q} \;\; (6.61b) \qquad q_0 = (\mathbf{q} + \mathbf{q}^*)/2 \;\; (6.61c) \qquad \vec{q} = (\mathbf{q} - \mathbf{q}^*)/2 \;\; (6.61d)$$

The **norm** of $\mathbf{q}$ is defined as $\|\mathbf{q}\| = \sqrt{\mathbf{q}\mathbf{q}^*} = \sqrt{\mathbf{q}^*\mathbf{q}} = (q_0^2 + q_1^2 + q_2^2 + q_3^2)^{1/2}$. If $\|\mathbf{q}\| = 1$ (referred to as a **unit quaternion** or **versor**), $\mathbf{q}$ may be written in the form of (6.60) for some angle ϕ and some unit vector $\vec{u}$. This representation is not unique: an angle of $-\phi$ and a unit vector of $-\vec{u}$ results in the same quaternion $\mathbf{q}$.

Fact 6.4 *The* **inverse** $\mathbf{q}^{-1}$ *of the quaternion* $\mathbf{q}$*, for which* $\mathbf{q}\,\mathbf{q}^{-1} = 1$*, is given simply by* $\mathbf{q}^{-1} = \mathbf{q}^*/\|\mathbf{q}\|^2$*. In particular, if* $\mathbf{q}$ *is a* **unit quaternion***, then* $\mathbf{q}^{-1} = \mathbf{q}^*$*.*

Proof: Follows directly from (6.59a). □

Fact 6.5 (Rotation using quaternions) *If* $\vec{u} = u_1\mathbf{i} + u_2\mathbf{j} + u_3\mathbf{k}$ *is a 3D unit vector (that is,* $u_1^2 + u_2^2 + u_3^2 = 1$*), and thus* $\mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$ *defined by* (6.60)*, with* $\phi = \theta/2$ *for some angle* θ*, is a unit quaternion (that is,* $q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$*), then any 3D vector* $\vec{p} = p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k}$ *may be rotated by the angle* $\theta = 2\phi$ *around the vector* $\vec{u}$ *by taking the product*

$$\vec{p}\,' = \mathbf{q}\,\vec{p}\,\mathbf{q}^*. \tag{6.62}$$

[19] In fact, upon this inspiration, Hamilton carved the defining relations on the left in (6.58) into the Broom Bridge in Dublin.

[20] It is common to denote the various square roots of -1 in a quaternion representation with boldface, to emphasize their interpretation as three unit vectors in a four-dimensional space. We thus adopt that convention here.

Proof: Noting the formula for the Hamilton product given in (6.59c) [interpreting $\vec{p}$ and $\vec{u}$ as quaternions $\mathbf{p}$ and $\mathbf{u}$ with zero real part, i.e., $p_0 = u_0 = 0$, for the purpose of performing multiplication], as well as the identities (B.21), (B.54), and (B.56), Fact 6.5 may be verified by comparing with Rodrigues' rotation formula (Fact 6.2) as follows:

$$\begin{aligned}
\vec{p}' &= \mathbf{q}\,\mathbf{p}\,\mathbf{q}^* = \left(\cos\frac{\theta}{2} + \mathbf{u}\,\sin\frac{\theta}{2}\right)\mathbf{p}\left(\cos\frac{\theta}{2} - \mathbf{u}\,\sin\frac{\theta}{2}\right) = \mathbf{p}\,\cos^2\frac{\theta}{2} + (\mathbf{u}\,\mathbf{p} - \mathbf{u}\,\mathbf{p})\sin\frac{\theta}{2}\,\cos\frac{\theta}{2} - \mathbf{u}\,\mathbf{p}\,\mathbf{u}\sin^2\frac{\theta}{2} \\
&= \vec{p}\,\cos^2\frac{\theta}{2} + 2\,\vec{u}\times\vec{p}\,\sin\frac{\theta}{2}\,\cos\frac{\theta}{2} + \vec{u}\,(\vec{p}\cdot\vec{u} - \vec{p}\times\vec{u})\sin^2\frac{\theta}{2} \\
&= \vec{p}\,\cos^2\frac{\theta}{2} + \vec{u}\times\vec{p}\,\sin\theta + [\vec{u}(\vec{p}\cdot\vec{u}) + \vec{u}\cdot(\vec{p}\times\vec{u}) - \vec{u}\times(\vec{p}\times\vec{u})]\sin^2\frac{\theta}{2} \\
&= \vec{p}\,\cos^2\frac{\theta}{2} + \vec{u}\times\vec{p}\,\sin\theta + [\vec{u}(\vec{p}\cdot\vec{u}) + 0 - (\vec{p}(\vec{u}\cdot\vec{u}) - \vec{u}(\vec{u}\cdot\vec{p}))]\sin^2\frac{\theta}{2} \\
&= \vec{p}\left(\cos^2\frac{\theta}{2} - \sin^2\frac{\theta}{2}\right) + \vec{u}\times\vec{p}\,\sin\theta + 2\,\vec{u}\,(\vec{u}\cdot\vec{p})\sin^2\frac{\theta}{2} = \vec{p}\,\cos\theta + (\vec{u}\cdot\vec{p})\,\vec{u}\,(1-\cos\theta) + \vec{u}\times\vec{p}\,\sin\theta. \quad \square
\end{aligned}$$

Note that, applying (6.59a) to (6.62), a 3×3 matrix formula for the rotated vector $\vec{p}' = \mathbf{q}\,\vec{p}\,\mathbf{q}^*$ is given by

$$\begin{pmatrix} p'_1 \\ p'_2 \\ p'_3 \end{pmatrix} = R_{\mathbf{q}} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \quad \text{with} \quad R_{\mathbf{q}} = \begin{pmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_1q_3+2q_0q_2 \\ 2q_1q_2+2q_0q_3 & q_0^2-q_1^2+q_2^2-q_3^2 & 2q_2q_3-2q_0q_1 \\ 2q_1q_3-2q_0q_2 & 2q_2q_3+2q_0q_1 & q_0^2-q_1^2-q_2^2+q_3^2 \end{pmatrix}. \tag{6.63}$$

Fact 6.6 *The product of two unit quaternions is a unit quaternion.*

Proof: Assume $\mathbf{p}$ and $\mathbf{q}$ are unit quaternions. In the first expression in (6.59a), the matrix derived from $\mathbf{p}$ is orthogonal, and $q_0^2+q_1^2+q_2^2+q_3^2 = 1$; in the second expression in (6.59a), the matrix derived from $\mathbf{q}$ is orthogonal, and $p_0^2+p_1^2+p_2^2+p_3^2 = 1$. Fact 6.6 thus follows directly from either expression. $\square$

Coupling Facts 6.5 and 6.6, it is seen that, if a rotation characterized by a unit quaternion $\mathbf{q}$ is followed by a rotation characterized by a unit quaternion $\mathbf{p}$, the total effect of the two rotations is equivalent to a single rotation characterized by the unit quaternion $\mathbf{r} = \mathbf{p}\,\mathbf{q}$.

6.3.2.3 Euler and Tait-Bryan rotation sequences

It is easy to see that, starting from a reference orientation, successively rotating a body about one of its body-fixed axes, then about a different body-fixed axis, then about the first body-fixed axis, called a **Euler rotation sequence**, any possible final orientation of the body can be achieved. Once the axes are affixed to the body, there are $3! = 6$ choices for which axes to rotate about following this approach. Starting from a reference configuration, one commonly-used convention is known as the **3-1-3 Euler rotation sequence**:

3 rotate the body by an angle α about the body-fixed z axis, then

1 rotate the body by an angle β about the body-fixed x axis, then

3 rotate the body by an angle γ about the body-fixed z axis.

Note that rotations are always performed using the **right-hand rule** (pointing your right thumb along the axis, the direction that your fingers curl corresponds to positive rotation). Note also that *order matters*: these rotations must be applied in succession, in this order, or a different final orientation results.

Similarly, it is easy to see that successively rotating a body about each of its body-fixed axes in turn, called a **Tait-Bryan rotation sequence**, any possible final orientation of the body can be achieved. There are again

$3! = 6$ choices for which axes to rotate about following this approach. Starting from a reference configuration, one commonly-used convention is known as the **3-2-1 Tait-Bryan rotation sequence**[21]:

3 rotate the body by an angle α about the body-fixed z axis, then

2 rotate the body by an angle β about the body-fixed y axis, then

1 rotate the body by an angle γ about the body-fixed x axis.

This convention is commonly used in the aerospace industry, where the Body frame axes are taken as vectors from the nominal center of mass out the nose, the right wingtip, and the bottom of the aircraft for x, y, and z, respectively, and the Reference frame axes are taken as north, east, and down (**NED**) for $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ axes, respectively. In this case, the 3-2-1 Tait-Bryan rotation sequence may be described as follows[22]:

3 **yaw** the aircraft by α about the z (down) axis (positive α yaws the nose to the right), then

2 **pitch** the aircraft by β about the y (out-the-right-wing) axis (positive β pitches the nose up), then

1 **roll** the aircraft by γ about the x (out-the-nose) axis (positive γ rolls the right wing down).

The 3-2-1 Tait-Bryan rotation sequence is also commonly used in the automobile industry, where two different conventions are used: in the **SAE** standards J670 (c. 2008, regarding automobile dynamics) and J1594 (c. 2010, regarding automobile aerodynamics), essentially same conventions as described above are used, whereas in the **ISO** standard 8855 (c. 2011) the body-fixed axes are taken as vectors out the front of the automobile, the left side, and the top for the x, y, and z axes, respectively, and the reference orientation is taken as east, north, and up (**ENU**) for the inertial $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ axes, respectively[23]. It is important to note that, in both the SAE and ISO automobile standards, the center of the coördinate system is taken as some reference point at the center of the automobile chassis, not necessarily the nominal center of mass; these rotation sequences are, of course, otherwise identical mathematically to the 3-2-1 Tait-Bryan rotation sequence used in the aerospace industry, but with the ISO convention having a slightly different physical interpretation:

3 **yaw** the automobile by α about the z (up) axis (positive α yaws the front to the left), then

2 **pitch** the automobile by β about the y (out-the-left-side) axis (positive β pitches the front down), then

1 **roll** the automobile by γ about the x (out-the-front) axis (positive γ rolls the right side down).

[21] The Euler and Tait-Bryan rotation sequences described here are called **intrinsic** rotation sequences, as subsequent rotations are applied around the (new) body axes after the previous rotations are complete. Alternatively, **extrinsic** rotation sequences may be applied, with each subsequent rotation applied around the inertial (unrotated) axes. Curiously, any intrinsic rotation sequence is equivalent to a corresponding extrinsic rotation sequence applied in the reverse order. Thus, for example, the (intrinsic) 3-2-1 Tait-Bryan rotation sequence described here is equivalent to the following (extrinsic) rotation sequence: [1] rotate the body by an angle γ about the inertial $\mathbf{i}$ axis, then [2] rotate the body by an angle β about the inertial $\mathbf{j}$ axis, then [3] rotate the body by an angle α about the inertial $\mathbf{k}$ axis.

[22] In this work, we denote the three successive rotations of any intrinsic rotation sequence as α, β, and γ, in order to emphasize the order in which the rotations are applied, where α and γ are defined modulo 2π radians (e.g., $-\pi < \alpha \leq \pi$, $-\pi < \gamma \leq \pi$, and β covers π radians (e.g., $-\pi/2 \leq \beta \leq \pi/2$). Note that, for the 3-2-1 Tait-Bryan rotation sequence commonly used in the aerodynamics literature, the notation ϕ, θ, and ψ for, respectively, **yaw**, **pitch**, and **roll** (a.k.a. **heading**, **elevation**, and **bank**), is somewhat more customary.

[23] Note that ENU is also a natural Reference frame for the 3-1-3 Euler rotation sequence discussed previously. When applied to a rotating top, the $\{\alpha, \beta, \gamma\}$ angles of the 3-1-3 Euler rotation sequence correspond precisely to **precession**, **nutation**, and **intrinsic rotation** (a.k.a., **spin**), as discussed further in Example 6.15.

There are many other possible choices for which axes to rotate about and which reference to compare with in Euler and Tait-Bryan rotation sequences; it is advised to stick with one of these common conventions.

To describe the orientation of a body as a rotation from its reference orientation using an Euler or Tait-Bryan rotation sequence, we may simply apply a product of three Givens rotations (see §??). To simplify, denote $\{\cos\alpha, \sin\alpha, \cos\beta, \sin\beta, \cos\gamma, \sin\gamma\}$ as $\{c1, s1, c2, s2, c3, s3\}$. Using the 3-1-3 Euler rotation sequence, first yaw about the z axis by α, then roll about the x axis by β, then yaw about the z axis by γ; that is, we define $R_{313}^{B\leftarrow R} = G(1,2;\gamma)\,G(2,3;\beta)\,G(1,2;\alpha)$:

$$R_{313}^{B\leftarrow R} = \begin{pmatrix} c3 & s3 & 0 \\ -s3 & c3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & c2 & s2 \\ 0 & -s2 & c2 \end{pmatrix} \begin{pmatrix} c1 & s1 & 0 \\ -s1 & c1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} c1\,c3 - c2\,s1\,s3 & c3\,s1 + c1\,c2\,s3 & s2\,s3 \\ -c1\,s3 - c2\,c3\,s1 & c1\,c2\,c3 - s1\,s3 & c3\,s2 \\ s1\,s2 & -c1\,s2 & c2 \end{pmatrix}$$

To rotate the body back to the reference orientation, perform the same rotations, using the opposite angles, and apply in the reverse order; that is,

$$R_{313}^{R\leftarrow B} = G(1,2;-\alpha)\,G(2,3;-\beta)\,G(1,2;-\gamma) = [R_{313}^{B\leftarrow R}]^T = \begin{pmatrix} c1\,c3 - c2\,s1\,s3 & -c1\,s3 - c2\,c3\,s1 & s1\,s2 \\ c3\,s1 + c1\,c2\,s3 & c1\,c2\,c3 - s1\,s3 & -c1\,s2 \\ s2\,s3 & c3\,s2 & c2 \end{pmatrix}$$

Similarly, using the 3-2-1 Tait-Bryan rotation sequence, first yaw about the z axis by α, then pitch about the y axis by β, then roll about the x axis by γ; that is, we define $R_{321}^{B\leftarrow R} = G(2,3;\gamma)\,G(3,1;\beta)\,G(1,2;\alpha)$:

$$R_{321}^{B\leftarrow R} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & c3 & s3 \\ 0 & -s3 & c3 \end{pmatrix} \begin{pmatrix} c2 & 0 & -s2 \\ 0 & 1 & 0 \\ s2 & 0 & c2 \end{pmatrix} \begin{pmatrix} c1 & s1 & 0 \\ -s1 & c1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} c1\,c2 & c2\,s1 & -s2 \\ c1\,s2\,s3 - c3\,s1 & c1\,c3 + s1\,s2\,s3 & c2\,s3 \\ s1\,s3 + c1\,c3\,s2 & c3\,s1\,s2 - c1\,s3 & c2\,c3 \end{pmatrix}$$

Again, to rotate the body back to the reference orientation, perform the same rotations, using the opposite angles, and apply in the reverse order; that is,

$$R_{321}^{R\leftarrow B} = G(1,2;-\alpha)\,G(3,1;-\beta)\,G(2,3;-\gamma) = [R_{321}^{B\leftarrow R}]^T = \begin{pmatrix} c1\,c2 & c1\,s2\,s3 - c3\,s1 & s1\,s3 + c1\,c3\,s2 \\ c2\,s1 & c1\,c3 + s1\,s2\,s3 & c3\,s1\,s2 - c1\,s3 \\ -s2 & c2\,s3 & c2\,c3 \end{pmatrix}$$

Thus, for example, the (extrinsic) quaternion representation of the (intrinsic) 3-1-3 Euler rotation sequence, applying the Hamilton product corresponding to each rotation in the reverse order of the intrinsic rotation as required by footnote 21 on page 6-31, is

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \cos\alpha/2 \\ 0 \\ 0 \\ \sin\alpha/2 \end{pmatrix} \begin{pmatrix} \cos\beta/2 \\ \sin\beta/2 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} \cos\gamma/2 \\ 0 \\ 0 \\ \sin\gamma/2 \end{pmatrix} = \begin{pmatrix} \cos\alpha/2\ \cos\beta/2\ \cos\gamma/2 - \sin\alpha/2\ \cos\beta/2\ \sin\gamma/2 \\ \cos\alpha/2\ \sin\beta/2\ \cos\gamma/2 + \sin\alpha/2\ \sin\beta/2\ \sin\gamma/2 \\ \sin\alpha/2\ \sin\beta/2\ \cos\gamma/2 - \cos\alpha/2\ \sin\beta/2\ \sin\gamma/2 \\ \cos\alpha/2\ \cos\beta/2\ \sin\gamma/2 + \sin\alpha/2\ \cos\beta/2\ \cos\gamma/2 \end{pmatrix}$$

whereas the (extrinsic) quaternion representation of the (intrinsic) 3-2-1 Tait-Bryan rotation sequence is

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} = \begin{pmatrix} \cos\alpha/2 \\ 0 \\ 0 \\ \sin\alpha/2 \end{pmatrix} \begin{pmatrix} \cos\beta/2 \\ 0 \\ \sin\beta/2 \\ 0 \end{pmatrix} \begin{pmatrix} \cos\gamma/2 \\ \sin\gamma/2 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \cos\alpha/2\ \cos\beta/2\ \cos\gamma/2 + \sin\alpha/2\ \sin\beta/2\ \sin\gamma/2 \\ \cos\alpha/2\ \cos\beta/2\ \sin\gamma/2 - \sin\alpha/2\ \sin\beta/2\ \cos\gamma/2 \\ \cos\alpha/2\ \sin\beta/2\ \cos\gamma/2 + \sin\alpha/2\ \cos\beta/2\ \sin\gamma/2 \\ \sin\alpha/2\ \cos\beta/2\ \cos\gamma/2 - \cos\alpha/2\ \sin\beta/2\ \sin\gamma/2 \end{pmatrix}$$

Conversely, comparing $R_{313}^{R\leftarrow B}$ above to $R_{\mathbf{q}}$ in (6.63), it is easy to verify that the following relations convert from a quaternion rotation sequence to the equivalent 3-1-3 Euler rotation sequence:

$$\text{if } r_{3,3} \approx 0 \begin{cases} \alpha = \text{atan2}(r_{3,3}, r_{1,1}), \\ \beta = \text{acos}\,(r_{3,3}), \\ \gamma = 0, \end{cases} \qquad \text{else} \quad \begin{cases} \alpha = \text{atan2}(r_{1,3}, -r_{2,3}), \\ \beta = \text{acos}\,(r_{3,3}), \\ \gamma = \text{atan2}(r_{3,1}, r_{3,2}), \end{cases}$$

and, comparing $R_{321}^{R\leftarrow B}$ above to $R_{\mathbf{q}}$ in (6.63), the following relations convert from a quaternion rotation sequence to the equivalent 3-2-1 Tait-Bryan rotation sequence:

$$\begin{aligned} \alpha &= \text{atan2}(2q_1q_2 - 2q_0q_3, q_0^2 + q_1^2 - q_2^2 - q_3^2), \\ \beta &= -\text{asin}\,(2q_1q_3 + 2q_0q_2), \\ \gamma &= \text{atan2}(2q_2q_3 - 2q_0q_1, q_0^2 - q_1^2 - q_2^2 + q_3^2). \end{aligned}$$

These relations, and those for the other 10 rotation sequences, are implemented in RR_rotation_sequence.

Euler and Tait-Bryan rotation sequences are **singular**, which means that the three angles $\{\alpha, \beta, \gamma\}$ must make a finite jump, in the vicinity of certain critical orientations, as the orientation goes through an infinitesimal change. [This singularity is akin to the (simpler, 2D) description of your location on the surface of the Earth, in terms of longitude and latitude, suddenly jumping (in longitude) by 180° when you take a single step over one of the poles.] The singularity of rotation sequences, and the nonsingular behavior of the quaternion representation of rotations, is illustrated well by comparing Examples 6.12 and 6.13 below.

The singularity of rotation sequences is often associated with **gimbal lock**, which is the loss of a degree of freedom of a three-gimbal mechanism in a gyroscope (used to measure vehicle orientation) which happens when the axes of two of the three gimbals become parallel. Note, however, that the singularity of rotation sequences is inherent to the mathematical description of the orientation itself, and is independent of the mechanical device actually used to measure the orientation of the vehicle.

Example 6.12 Starting with a body in the NED reference orientation, consider two rotations in succession: first roll the body (about its x axis) by $\pi/2$, then pitch the body (about its y axis) by $\pi/2$. We now describe the orientation of the body during each rotation using the (intrinsic) 3-2-1 Tait-Bryan rotation sequence, the (intrinsic) 3-1-3 Euler rotation sequence, and the (extrinsic) quaternion description of rotation.

In terms of a 3-2-1 Tait-Bryan rotation sequence, during the first rotation, γ changes continuously from 0 to $\pi/2$, while $\alpha = 0$ and $\beta = 0$ stay constant. During the second rotation, α (note: not β!) changes continuously from 0 to $\pi/2$, while $\beta = 0$ and $\gamma = \pi/2$ stay constant. After both rotations, $\{\alpha, \beta, \gamma\} = \{\pi/2, 0, \pi/2\}$.

In terms of a 3-1-3 Euler rotation sequence, during the first rotation, β changes continuously from 0 to $\pi/2$, while $\alpha = 0$ and $\gamma = 0$ stay constant. During the second rotation, α changes continuously from 0 to $\pi/2$, while $\beta = \pi/2$ and $\gamma = 0$ stay constant. After both rotations, $\{\alpha, \beta, \gamma\} = \{\pi/2, \pi/2, 0\}$.

In terms of quaternions, the first rotation is given by a rotation of $\theta_1 = \pi/2$ degrees about the $\mathbf{i}$ axis (i.e., $\vec{u}_1 = \mathbf{i}$). Thus, noting (6.60) and Fact 6.5, $\mathbf{q}_1 = (\sqrt{2}/2)(1 + \mathbf{i})$. The second rotation is given by a rotation of $\theta_2 = \pi/2$ degrees about the $\mathbf{k}$ axis (that is, $\vec{u}_2 = \mathbf{k}$). Thus, $\mathbf{q}_2 = (\sqrt{2}/2)(1 + \mathbf{k})$. The total rotation is given by

$$\mathbf{q} = \mathbf{q}_2\,\mathbf{q}_1 = (1 + \mathbf{k})(1 + \mathbf{i})/2 = (1 + \mathbf{i} + \mathbf{j} + \mathbf{k})/2 = \cos(\pi/3) + [(\mathbf{i} + \mathbf{j} + \mathbf{k})/\sqrt{3}]\sin(\pi/3);$$

that is, it is given by a rotation of $\theta = 2\pi/3$ radians around the unit vector $\vec{u} = (\mathbf{i} + \mathbf{j} + \mathbf{k})/\sqrt{3}$.

Example 6.13 We now repeat Example 6.12 for the following two rotations: first pitch the body (about its y axis) by $\pi/2$, then yaw the body (about its z axis) by $\pi/2$. Note that the final orientation after these two rotations happens to be the same as that after the two rotations considered in Example 6.12.

In terms of a 3-2-1 Tait-Bryan rotation sequence, during the first rotation, β changes continuously from 0 to $\pi/2$, while $\alpha = 0$ and $\gamma = 0$ stay constant. In contrast with Example 6.12, describing the second rotation in terms of Tait-Bryan angles is problematical, as the coördinate description of the configuration after the first rotation is singular. Before the second rotation begins, the Tait-Bryan angles must jump suddenly, from $\{\alpha, \beta, \gamma\} = \{0, \pi/2, 0\}$ to, e.g., $\{\alpha, \beta, \gamma\} = \{\pi/2, \pi/2, \pi/2\}$. During the second rotation, β then gradually reduces, from $\pi/2$ to 0. After both rotations are complete, $\{\alpha, \beta, \gamma\} = \{\pi/2, 0, \pi/2\}$.

In terms of a 3-1-3 Euler rotation sequence, before the first rotation begins, the Euler angles must jump suddenly, from $\{\alpha, \beta, \gamma\} = \{0, 0, 0\}$ to, e.g., $\{\alpha, \beta, \gamma\} = \{\pi/2, 0, -\pi/2\}$, as in this case the coördinate description of the initial configuration is singular. During the first rotation, β then gradually increases, from 0 to $\pi/2$. During the second rotation, γ changes continuously from $-\pi/2$ to 0, while $\alpha = \pi/2$ and $\beta = \pi/2$ stay constant. After both rotations are complete, $\{\alpha, \beta, \gamma\} = \{\pi/2, \pi/2, 0\}$.

In terms of quaternions, the first rotation is given by a rotation of $\theta_1 = \pi/2$ degrees about the **j** axis (that is, $\vec{u}_1 = \mathbf{j}$). Thus, noting (6.60) and Fact 6.5, $\mathbf{q}_1 = (\sqrt{2}/2)(1 + \mathbf{j})$. The second rotation is given by a rotation of $\theta_2 = \pi/2$ degrees about the **i** axis (that is, $\vec{u}_2 = \mathbf{i}$). Thus, $\mathbf{q}_2 = (\sqrt{2}/2)(1 + \mathbf{i})$. The total rotation is given by

$$\mathbf{q} = \mathbf{q}_2\,\mathbf{q}_1 = \frac{1}{2}(1+\mathbf{i})(1+\mathbf{j}) = \frac{1}{2}(1+\mathbf{i}+\mathbf{j}+\mathbf{k}) = \cos(\pi/3) + \frac{\mathbf{i}+\mathbf{j}+\mathbf{k}}{\sqrt{3}}\sin(\pi/3);$$

that is, it is given by a rotation of $\theta = 2\pi/3$ radians around the unit vector $\vec{u} = (\mathbf{i} + \mathbf{j} + \mathbf{k})/\sqrt{3}$.

As expected, the final configurations in Examples 6.12 and 6.13 are identical in terms of the final angles of the 3-2-1 Tait-Bryan rotation sequence and the 3-1-3 Euler rotation sequence, as well as the total rotation quaternion $\mathbf{q}$. These configurations are interrelated by the several equations derived earlier in this subsection.

Note that special treatment was required in Example 6.13 to move through the singularities of both the 3-2-1 Tait-Bryan rotation sequence as well as the 3-1-3 Euler rotation sequence, neither of which happened to be encountered in Example 6.12. In sharp contrast, the quaternion description of a rotation is always nonsingular, never requiring such special treatment. In problems in which general vehicle rotations must be well handled (for example, in a fighter aircraft), quaternion descriptions are thus preferred; in problems in which the expected motions of the vehicle is limited in ways that avoid such singularities (for example, in a commercial transport aircraft), rotation sequences are sometimes more intuitive and convenient.

6.3.3 Vectors in different frames of reference, and the rate of rotation $\vec{\omega}$

We will have occasion in the discussion that follows to describe vectors in different frames of reference, some of which are moving. Though straightforward, this process is somewhat subtle, and must be treated with care.

We begin with a nonrotating, nonaccelerating reference frame E, with the Cartesian unit vectors $\{\vec{e}^1, \vec{e}^2, \vec{e}^3\}$ (see §??) providing an orthogonal set of basis vectors satisfying the right-hand rule. In this reference frame, we define additional sets of orthogonal unit vectors satisfying the right-hand rule, $\{\vec{g}^1, \vec{g}^2, \vec{g}^3\}$ and $\{\vec{b}^1, \vec{b}^2, \vec{b}^3\}$ (each referred to as a **dextral set**), which may be assembled as the columns of corresponding matrices, G and B (each sometimes referred to as a **vectrix**), satisfying the following properties

$$\vec{g}_i \cdot \vec{g}_j = \delta_{ij}, \quad \vec{g}_1 \times \vec{g}_2 = \vec{g}_3, \quad \vec{g}_2 \times \vec{g}_3 = \vec{g}_1, \quad \vec{g}_3 \times \vec{g}_1 = \vec{g}_2; \quad G^T G = I, \quad |G| = 1; \tag{6.64a}$$

$$\vec{b}_i \cdot \vec{b}_j = \delta_{ij}, \quad \vec{b}_1 \times \vec{b}_2 = \vec{b}_3, \quad \vec{b}_2 \times \vec{b}_3 = \vec{b}_1, \quad \vec{b}_3 \times \vec{b}_1 = \vec{b}_2; \quad B^T B = I, \quad |B| = 1. \tag{6.64b}$$

The unit vectors $\{\vec{b}^1, \vec{b}^2, \vec{b}^3\}$ may be considered as *rotations* of $\{\vec{g}^1, \vec{g}^2, \vec{g}^3\}$ into new directions by the action of some **rotation matrix** $R^{B\leftarrow G}$; it follows that

$$B = R^{B\leftarrow G} G \text{ with } R^{B\leftarrow G} = B\,G^T, \quad \text{and} \quad G = R^{G\leftarrow B} B \text{ with } R^{G\leftarrow B} = G\,B^T = [R^{B\leftarrow G}]^T. \tag{6.65}$$

Consider now some vector $\vec{r}$ defined in the original E frame, which is now *represented* as a linear combination of the unit vectors in the G frame and in the B frame, that is, $\vec{r} = G\,\vec{r}^{\,G} = B\,\vec{r}^{\,B}$, and thus

$$\vec{r}^{\,B} = D^{\,B\leftarrow G}\vec{r}^{\,G} \;\text{ with }\; D^{\,B\leftarrow G} = B^T G, \quad \text{and} \quad \vec{r}^{\,G} = D^{\,G\leftarrow B}\vec{r}^{\,B} \;\text{ with }\; D^{\,G\leftarrow B} = G^T B = [D^{\,B\leftarrow G}]^T; \tag{6.66}$$

the vectors $\vec{r}^{\,G}$ and $\vec{r}^{\,B}$ are called the **representations** of the vector $\vec{r}$ in the G and B frames, respectively. The (orthogonal) **direction cosine matrix** $D^{\,B\leftarrow G}$ relates these two representations. The elements of the direction cosine matrix $D^{\,B\leftarrow G}$ derived in (6.66) are given by the *inner* products [see (??)] of the corresponding unit vectors in the B and G frames, these inner products are referred to as the **direction cosines** of the corresponding frames such that $d_{ij}^{B\leftarrow G} = \vec{b}^{\,i}\cdot\vec{g}^{\,j} = \cos\alpha_{ij}$, where α_{ij} is the angle between $\vec{b}^{\,i}$ and $\vec{g}^{\,j}$. In contrast, the rotation matrix $R^{\,B\leftarrow G}$ defined in (6.65) is given by the sum of the *outer* products [see (??)] of the corresponding unit vectors in the B and G frames. In certain special cases (e.g., if G is the identity matrix, and thus the G frame coincides with E frame), the direction cosine matrix and rotation matrices relating the G and B frames satisfy $D^{\,B\leftarrow G} = R^{\,G\leftarrow B}$ and $D^{\,G\leftarrow B} = R^{\,B\leftarrow G}$; however, these relations are *not* true in general[24].

We now develop a useful identity that will be leveraged in the discussion that follows.

Fact 6.7 *If $\vec{a} = B\,\vec{a}^{\,B}$, $\vec{c} = B\,\vec{c}^{\,B}$, and B is a vectrix with columns satisfying (6.64b), then, noting (B.18),*

$$\vec{a}\times\vec{c} = \big[\vec{a}\big]_\times\vec{c} = (B\,\vec{a}^{\,B})\times(B\,\vec{c}^{\,B}) = B\big[\vec{a}^{\,B}\big]_\times\vec{c}^{\,B} = B(\vec{a}^{\,B}\times\vec{c}^{\,B}). \tag{6.67}$$

Proof: Write $B\,\vec{a}^{\,B} = \sum_i \vec{b}^{\,i}a_i^B$ and $B\,\vec{c}^{\,B} = \sum_j \vec{b}^{\,j}c_j^B$. Then

$$(B\,\vec{a}^{\,B})\times(B\,\vec{c}^{\,B}) = \sum_{i,j} a_i^B c_j^B(\vec{b}^{\,i}\times\vec{b}^{\,j}) = \vec{b}^{\,1}(a_2^Bc_3^B - a_3^Bc_2^B) + \vec{b}^{\,2}(a_3^Bc_1^B - a_1^Bc_3^B) + \vec{b}^{\,3}(a_1^Bc_2^B - a_2^Bc_1^B) = B\big[\vec{a}^{\,B}\big]_\times\vec{c}^{\,B}.$$

The other relations in (6.67) follow trivially from the stated definitions. □

Now consider some vector $\vec{r}$ and its time derivative, $\dot{\vec{r}} = d\vec{r}/dt$. Since $\vec{r} = G\,\vec{r}^{\,G} = B\,\vec{r}^{\,B}$, we have

$$\dot{\vec{r}} = \dot{G}\,\vec{r}^{\,G} + G\,\dot{\vec{r}}^{\,G} = \dot{B}\,\vec{r}^{\,B} + B\,\dot{\vec{r}}^{\,B}. \tag{6.68}$$

Recalling from (6.66) that $B = G\,D^{\,G\leftarrow B}$ and differentiating, we also may write

$$\dot{B} = \dot{G}\,D^{\,G\leftarrow B} + G\,\dot{D}^{\,G\leftarrow B}.$$

We now suppose that *the G frame is fixed in time* (i.e., $\dot{G} = 0$), but *the B frame is attached in a convenient manner to the body*, and rotates with it; noting that $G = B\,(D^{\,G\leftarrow B})^T$, it follows that

$$\dot{B} = G\,\dot{D}^{\,G\leftarrow B} = B\,(D^{\,G\leftarrow B})^T\dot{D}^{\,G\leftarrow B}, \quad \text{and thus} \quad B^T\dot{B} = (D^{\,G\leftarrow B})^T\dot{D}^{\,G\leftarrow B}.$$

Recalling that B is orthogonal, it follows that $B^TB = I$; differentiating, it follows that

$$\dot{B}^TB + B^T\dot{B} = 0 \quad\Rightarrow\quad B^T\dot{B} = -(B^T\dot{B})^T;$$

that is, the expression $B^T\dot{B}$ itself is skew symmetric. Thus, noting (B.18), we may write

$$B^T\dot{B} = (D^{\,G\leftarrow B})^T\dot{D}^{\,G\leftarrow B} \triangleq \begin{pmatrix} 0 & -\omega_3^B & \omega_2^B \\ \omega_3^B & 0 & -\omega_1^B \\ -\omega_2^B & \omega_1^B & 0 \end{pmatrix} = \big[\vec{\omega}^{\,B}\big]_\times \quad\Rightarrow\quad \dot{B} = B\big[\vec{\omega}^{\,B}\big]_\times, \quad \dot{D}^{\,G\leftarrow B} = D^{\,G\leftarrow B}\big[\vec{\omega}^{\,B}\big]_\times.$$

[24] This point is somewhat muddled in many available texts and online resources, which sometimes use the terms "direction cosine matrix" and "rotation matrix" essentially synonymously. The reader is advised to be semantically precise, to avoid mistakes when $G \neq I$.

The vector $\vec{\omega}^B$ defined above requires further analysis to be properly interpreted. From the above together with Fact 6.7, defining $\vec{\omega} = B\vec{\omega}^B$, we have $\dot{B}\,\vec{r}^B = B\left[\vec{\omega}^B\right]_\times \vec{r}^B = \vec{\omega} \times \vec{r}$; thus, by (6.68) with $\dot{G} = 0$,

$$\boxed{\dot{\vec{r}} = G\,\dot{\vec{r}}^G = B\,\dot{\vec{r}}^B + B\vec{\omega}^B \times \vec{r}^B = B\,\dot{\vec{r}}^B + \vec{\omega} \times \vec{r}.} \tag{6.69}$$

If the body is not rotating (that is, if $\dot{B} = 0$), then $\vec{\omega} = \vec{\omega}^B = 0$, and $\dot{\vec{r}} = G\,\dot{\vec{r}}^G = B\,\dot{\vec{r}}^B$; however, if the body is rotating, then the $\omega \times \vec{r}$ term must be added as shown above to account for this rotation. [Alternatively, if the vector $\vec{r}^B$ is fixed to some point a on the (rotating) body, then $\dot{\vec{r}}^B = 0$, and $\dot{\vec{r}} = G\,\dot{\vec{r}}^G = \vec{\omega} \times \vec{r}$.] The vector $\vec{\omega}$ is called the **instantaneous rate of rotation** of the body; as with $\vec{r}$, it may be represented in three different reference frames: $\vec{\omega} = G\vec{\omega}^G = B\vec{\omega}^B$.

We now provide an alternative derivation of the instantaneous rate of rotation $\vec{\omega}$. We again define a vector $\vec{r}^B$ [in some convenient set of Body coördinates B] that is fixed to some point a on the (rotating) body, so that $\dot{\vec{r}}^B = 0$, and consider its corresponding (time-varying) coördinates in the original frame E at time t, which we denote $\vec{r}(t)$. Further, the Body frame B considered is taken as aligned with the original frame E at time t, so that $\vec{r}(t) = \vec{r}^B$. Recall from Fact 6.1 that any finite rotation is representable as a single vector in $\mathbb{R}^3$ in the direction of the rotation axis and of length given by the angle of rotation. Consider now an **infinitesimal rotation**, which occurs over the infinitesimal time δt, which may thus be expressed as the product of some unit vector along the **instantaneous axis of rotation** of the solid body, $\vec{u}$, times some **infinitesimal angle of rotation**, $\delta\phi$, around this axis via the right-hand rule. Via Fact 6.2, taking $\theta = \delta\phi$, we may write

$$\vec{r}(t+\delta t) = \vec{r}(t) + \delta\phi(\vec{u} \times \vec{r}) \quad \Rightarrow \quad \frac{\vec{r}(t+\delta t) - \vec{r}(t)}{\delta t} = \frac{\delta\vec{r}}{\delta t} = \frac{\vec{u}\,\delta\phi}{\delta t} \times \vec{r} = \vec{\omega} \times \vec{r},$$

thus identifying the **instantaneous rate of rotation** at time t as $\vec{\omega}(t) = \vec{u}\ d\phi/dt \triangleq d\vec{\phi}/dt$.

The 3-2-1 Tait-Bryan rotation $R_{321}^{G\leftarrow B}$ derived previously, taking $G = I$ and the infinitesimal rotations $\delta\vec{\phi} = (\delta\phi_1, \delta\phi_2, \delta\phi_3) = (\gamma, \beta, \alpha)$, provides an equivalent formulation of the scenario described in the previous paragraph. Noting the definition of $\left[\delta\vec{\phi}\right]_\times$ in (B.18), this rotation reduces to[25]

$$R_{\delta\vec{\phi}} = \begin{pmatrix} 1 & -\delta\phi_3 & \delta\phi_2 \\ \delta\phi_3 & 1 & -\delta\phi_1 \\ -\delta\phi_2 & \delta\phi_1 & 1 \end{pmatrix} = I + \left[\delta\vec{\phi}\right]_\times \tag{6.70}$$

It is thus seen that the infinitesimal rotations of a solid body about each of its axes are decoupled, and may be performed in any order. By (6.70), taking $\vec{\omega}(t) = d\vec{\phi}/dt$, we may thus write

$$\vec{r}(t+\delta t) = R_{\delta\vec{\phi}}\vec{r}(t) = \left(I + \left[\delta\vec{\phi}\right]_\times\right)\vec{r}(t) = \vec{r}(t) + \frac{d\vec{r}}{dt}\delta t \quad \Rightarrow \quad \frac{d\vec{r}}{dt} = \frac{\left[\delta\vec{\phi}\right]_\times}{\delta t}\vec{r} = \frac{\delta\vec{\phi}}{\delta t} \times \vec{r} = \vec{\omega} \times \vec{r}.$$

The vector $\vec{\omega}^B(t)$ describes the instantaneous rate of rotation of a solid body around its own body fitted axes at time t; the relations $\vec{\omega} = G\vec{\omega}^G = B\vec{\omega}^B$ may be used as necessary transform this vector to the E or G frame. More generally, to describe the evolution of the orientation of the body itself after the body rotates for a finite period of time, the rate of change of the Euler, Tait-Bryan, and quaternion descriptions of the solid body's orientation itself must be computed by integrating the effect of the instantaneous rate of rotation $\vec{\omega}(t)$ on these descriptions of the orientation over time, as discussed next.

[25] The quaternion representation of the 3-2-1 Tait-Bryan rotation sequence reveals an equivalent expression for an infinitesimal rotation about each of the axes. With $(\phi_1, \phi_2, \phi_3) = (\gamma, \beta, \alpha)$, this rotation may be represented as $\{q_0, q_1, q_2, q_3\} = \{1, \delta\phi_1/2, \delta\phi_2/2, \delta\phi_3/2\}$; applying these relations to (6.63), the same expression for $R_{\delta\vec{\phi}}$ as given in (6.70) results.

6.3.4 The rate of change of a solid body's orientation as a function of $\vec{\omega}$

The rate of change of the 3-2-1 Tait-Bryan rotation sequence

Recall the 3-2-1 Tait-Bryan rotation sequence and the transformation $R_{321}^{B\leftarrow R} = G(2,3;\gamma)\,G(3,1;\beta)\,G(1,2;\alpha)$ from the R frame to the B frame. We now associate with this rotation sequence two intermediate frames, I and II, such that

- $R_{321}^{I\leftarrow R} = G(1,2;\alpha)$ (i.e., the I frame is given by yawing the R frame by α),
- $R_{321}^{II\leftarrow I} = G(3,1;\beta)$ (i.e., the II frame is given by pitching the I frame by β), and
- $R_{321}^{B\leftarrow II} = G(2,3;\gamma)$ (i.e., the B frame is given by rolling the II frame by γ);

it follows that $R_{321}^{B\leftarrow R} = R_{321}^{B\leftarrow II}\ R_{321}^{II\leftarrow I}\ R_{321}^{I\leftarrow R}$. Note further that

- the yaw rate $\dot{\alpha}$ represents rotation of the body about the z-axis in both the R and I frames,
- the pitch rate $\dot{\beta}$ represents rotation of the body about the y-axis in both the I and II frames, and
- the roll rate $\dot{\gamma}$ represents rotation of the body about the x-axis in both the II and B frames.

Thus, to relate the rotations implied by the rate of change of the 3-2-1 Tait-Bryan angles ($\dot{\alpha}$, $\dot{\beta}$, and $\dot{\gamma}$) to the three instantaneous body rotation rates in the B frame (ω_1^B, ω_2^B, and ω_3^B), we may transform as follows:

$$\begin{pmatrix}\omega_1^B\\ \omega_2^B\\ \omega_3^B\end{pmatrix} = G(2,3;\gamma)\,G(3,1;\beta)\,G(1,2;\alpha)\begin{pmatrix}0\\0\\\dot{\alpha}\end{pmatrix} + G(2,3;\gamma)\,G(3,1;\beta)\begin{pmatrix}0\\\dot{\beta}\\0\end{pmatrix} + G(2,3;\gamma)\begin{pmatrix}\dot{\gamma}\\0\\0\end{pmatrix}$$

$$= G(2,3;\gamma)\,G(3,1;\beta)\begin{pmatrix}0\\0\\\dot{\alpha}\end{pmatrix} + G(2,3;\gamma)\begin{pmatrix}0\\\dot{\beta}\\0\end{pmatrix} + \begin{pmatrix}\dot{\gamma}\\0\\0\end{pmatrix} = \begin{pmatrix}1 & 0 & -\sin(\beta)\\ 0 & \cos(\gamma) & \sin(\gamma)\cos(\beta)\\ 0 & -\sin(\gamma) & \cos(\gamma)\cos(\beta)\end{pmatrix}\begin{pmatrix}\dot{\gamma}\\\dot{\beta}\\\dot{\alpha}\end{pmatrix}.$$

Thus, taking the inverse (easily confirmed by multiplying the matrix below by the last matrix above),

$$\frac{d}{dt}\begin{pmatrix}\gamma\\\beta\\\alpha\end{pmatrix} = \begin{pmatrix}1 & \sin(\gamma)\tan(\beta) & \cos(\gamma)\tan(\beta)\\ 0 & \cos(\gamma) & -\sin(\gamma)\\ 0 & \sin(\gamma)/\cos(\beta) & \cos(\gamma)/\cos(\beta)\end{pmatrix}\begin{pmatrix}\omega_1^B\\\omega_2^B\\\omega_3^B\end{pmatrix}. \tag{6.71}$$

Note that (6.71) is valid for all angles except the singular points $\beta = \pm\pi/2$ identified in Example 6.13.

The rate of change of the 3-1-3 Euler rotation sequence

Recall now the 3-1-3 Euler rotation sequence and transformation $R_{313}^{B\leftarrow R} = G(1,2;\gamma)\,G(2,3;\beta)\,G(1,2;\alpha)$. We now associate with this rotation sequence two intermediate frames, I and II, such that

- $R_{313}^{I\leftarrow R} = G(1,2;\alpha)$ (i.e., the I frame is given by yawing the R frame by α),
- $R_{313}^{II\leftarrow I} = G(2,3;\beta)$ (i.e., the II frame is given by rolling the I frame by β), and
- $R_{313}^{B\leftarrow II} = G(1,2;\gamma)$ (i.e., the B frame is given by yawing the II frame by γ);

it follows that $R_{313}^{B\leftarrow R} = R_{313}^{B\leftarrow II}\; R_{313}^{II\leftarrow I}\; R_{313}^{I\leftarrow R}$. Note further that

- the yaw rate $\dot{\alpha}$ represents rotation of the body about the z-axis in both the R and I frames,
- the roll rate $\dot{\beta}$ represents rotation of the body about the x-axis in both the I and II frames, and
- the yaw rate $\dot{\gamma}$ represents rotation of the body about the z-axis in both the II and B frames.

Thus, to relate the rotations implied by the rate of change of the 3-1-3 Euler angles ($\dot{\alpha}$, $\dot{\beta}$, and $\dot{\gamma}$) to the three instantaneous body rotation rates in the B frame (ω_1^B, ω_2^B, and ω_3^B), we may transform as follows:

$$\begin{pmatrix}\omega_1^B\\ \omega_2^B\\ \omega_3^B\end{pmatrix} = G(1,2;\gamma)\,G(2,3;\beta)\,G(1,2;\alpha)\begin{pmatrix}0\\0\\\dot{\alpha}\end{pmatrix} + G(1,2;\gamma)\,G(2,3;\beta)\begin{pmatrix}\dot{\beta}\\0\\0\end{pmatrix} + G(1,2;\gamma)\begin{pmatrix}0\\0\\\dot{\gamma}\end{pmatrix}$$
$$= G(1,2;\gamma)\,G(2,3;\beta)\begin{pmatrix}0\\0\\\dot{\alpha}\end{pmatrix} + G(1,2;\gamma)\begin{pmatrix}\dot{\beta}\\0\\0\end{pmatrix} + \begin{pmatrix}0\\0\\\dot{\gamma}\end{pmatrix} = \begin{pmatrix}\cos(\gamma) & \sin(\gamma)\sin(\beta) & 0\\ -\sin(\gamma) & \cos(\gamma)\sin(\beta) & 0\\ 0 & \cos(\beta) & 1\end{pmatrix}\begin{pmatrix}\dot{\beta}\\\dot{\alpha}\\\dot{\gamma}\end{pmatrix}.$$

To simplify the resulting expression, the last vector on the RHS above has been reordered. Taking the inverse,

$$\frac{d}{dt}\begin{pmatrix}\beta\\\alpha\\\gamma\end{pmatrix} = \begin{pmatrix}\cos(\gamma) & -\sin(\gamma) & 0\\ \sin(\gamma)/\sin(\beta) & \cos(\gamma)/\sin(\beta) & 0\\ -\sin(\gamma)\cot(\beta) & -\cos(\gamma)\cot(\beta) & 1\end{pmatrix}\begin{pmatrix}\omega_1^B\\\omega_2^B\\\omega_3^B\end{pmatrix}. \tag{6.72}$$

Note that (6.71) is valid for all angles except the singular point $\beta = 0$ identified in Example 6.13.

The rate of change of the quaternion description of orientation

Let the unit quaternion $\mathbf{q}$ represent the rotation of any vector $\vec{p}^{\,B}$ in the Body frame to the corresponding vector $\vec{p} = \mathbf{q}\,\vec{p}^{\,B}\,\mathbf{q}^*$ in the original E frame, noting Fact 6.5. We now consider a vector $\vec{p}^{\,B}$ fixed in the Body frame (that is, $d\vec{p}^{\,B}/dt = 0$), and relate $d\mathbf{q}/dt$ to the instantaneous rate of rotation of the body $\vec{\omega}$. Applying the product rule of differentiation,

$$\frac{d\vec{p}}{dt} = \frac{d\mathbf{q}}{dt}\,\vec{p}^{\,B}\,\mathbf{q}^* + \mathbf{q}\,\vec{p}^{\,B}\,\frac{d\mathbf{q}^*}{dt} = \frac{d\mathbf{q}}{dt}\,\vec{p}^{\,B}\,\mathbf{q}^* + \left[\frac{d\mathbf{q}}{dt}\left(\vec{p}^{\,B}\right)^*\mathbf{q}^*\right]^* = \frac{d\mathbf{q}}{dt}\,\vec{p}^{\,B}\,\mathbf{q}^* - \left(\frac{d\mathbf{q}}{dt}\,\vec{p}^{\,B}\,\mathbf{q}^*\right)^*. \tag{6.73}$$

Recalling from (6.69) that $d\vec{p}/dt = \vec{\omega}\times\vec{p}$, the quaternion formulation of this equation [noting (6.61d)] is

$$\frac{d\vec{p}}{dt} = [\vec{\omega}\,\vec{p} - (\vec{\omega}\,\vec{p})^*]/2. \tag{6.74}$$

Equating the RHS of (6.73) and (6.74), we have $(d\mathbf{q}/dt)\,\vec{p}^{\,B}\,\mathbf{q}^* = \vec{\omega}\,\vec{p}/2 = (\vec{\omega}\,\mathbf{q}/2)\,\vec{p}^{\,B}\,\mathbf{q}^*$ for arbitrary $\vec{p}^{\,B}$; thus, applying $\vec{\omega} = \mathbf{q}\,\vec{\omega}^B\,\mathbf{q}^*$, we have

$$\frac{d\mathbf{q}}{dt} = \vec{\omega}\,\mathbf{q}/2 = \mathbf{q}\,\vec{\omega}^B/2. \tag{6.75}$$

Unlike the expressions for the evolution of the 3-2-1 Tait-Bryan rotation sequence in (6.71) and the evolution of the 3-1-3 Euler rotation sequence in (6.72), the expression above for the evolution of $\mathbf{q}$ is nonsingular for all $\mathbf{q}$, as identified in Example 6.13.

6.4 Solid body dynamics

6.4.1 The (conserved) momentum, energy, and angular momentum of a free body

Recall the **total mass** $\mu = \sum_a m_a$, the **position of the center of mass** $\vec{R} = \sum_a m_a \vec{r}_a/\mu$, and the **inertial tensor** $I_{ik} \triangleq \sum_a m_a(r_j^2\delta_{ik} - r_i r_k)_a$ of a cloud of rigidly connected particles given in (6.53a); these definitions easily pass to the limit of a solid body (that is, to an infinite number of infinitesimal particles) by converting the sums to integrals, as given in (6.53b). We now develop the formulae for the momentum, energy, and angular momentum of a **free solid body** (i.e., a closed system) by passing the corresponding formula in §6.2 to the same continuüm limit, recalling that a solid body has six degrees of freedom: three to describe the location of its center of mass, and three to describe its orientation as a finite 3D rotation from a reference orientation, as discussed extensively above. To disambiguate the discussion that follows, we will make use of three reference frames (identified, perhaps pedantically, with superscripts): a *stationary* (that is, nonrotating, nonaccelerating) frame E, a *moving* frame A, aligned with E but centered at the center of mass of the body, and a *body* frame B, both centered at the center of mass of the body and rotating with the body itself.

In §6.2.2.1, the velocity of the center of mass of an N particle system (in the original E frame) was identified as $\vec{V} = \sum_a m_a \vec{v}_a/\mu = d\vec{R}/dt$, and the **total momentum** was defined as $\vec{P} = \sum_a \vec{p}_a = \sum_a m_a \vec{v}_a = \mu\vec{V}$. These definitions, and the property of **conservation of momentum** $d\vec{P}/dt = 0$ given in (6.49), extend immediately to solid bodies by defining

$$\vec{R} = \int_\Omega \rho\, \vec{r}^{\,E}\, dV/\mu, \quad \vec{V} = \int_\Omega \rho\, \vec{v}^{\,E}\, dV/\mu, \quad \text{and} \quad \vec{P} = \mu\vec{V}. \tag{6.76}$$

Once the solid body's position, velocity, and instantaneous rate of rotation are identified, we may write the position and velocity of any point a on the solid body, located a fixed position with respect to the center of mass in body coördinates (i.e., $\vec{r}_a^{\,B}/dt = 0$), as the sum the two components.

$$\vec{r}_a^{\,E} = \vec{R} + \vec{r}_a^{\,A} = \vec{R} + B\,\vec{r}_a^{\,B} \quad \text{and} \quad \vec{v}_a^{\,E} = \vec{V} + \vec{\omega}^{\,A} \times \vec{r}_a^{\,A} = \vec{V} + B\,\vec{\omega}^{\,B} \times \vec{r}_a^{\,B}. \tag{6.77}$$

In §6.2.2.2, the **kinetic energy** of a system of particles was identified in (6.41) as $T(\mathbf{v}) = \sum_a m_a \|\vec{v}_a\|^2/2$; passing to the continuüm limit, noting (6.77), (B.20), (B.19), and the definition of I_{ik} in §6.3.1, this definition may be extended to solid bodies by taking

$$\begin{aligned}
T &= \int_\Omega \frac{\rho\|\vec{v}^{\,E}\|^2}{2}\, dV = \int_\Omega \frac{\rho\|\vec{V} + \vec{\omega}^{\,A} \times \vec{r}^{\,A}\,\|^2}{2}\, dV = \frac{\mu\|\vec{V}\|^2}{2} + \int_\Omega \rho\Big(\vec{V} \cdot \vec{\omega}^{\,A} \times \vec{r}^{\,A} + \frac{\|\vec{\omega}^{\,A} \times \vec{r}^{\,A}\,\|^2}{2}\Big)\, dV \\
&= \frac{\mu\|\vec{V}\|^2}{2} + \cancelto{0}{\int_\Omega \rho\vec{r}^{\,A}\, dV} \cdot \vec{V} \times \vec{\omega}^{\,A} + \int_\Omega \rho \frac{\|\vec{\omega}^{\,A}\|^2\,\|\vec{r}^{\,A}\|^2 - (\vec{\omega}^{\,A} \cdot \vec{r}^{\,A})^2}{2}\, dV \\
&= \frac{\mu\|\vec{V}\|^2}{2} + \int_\Omega \rho \frac{\|\vec{\omega}^{\,B}\|^2\,\|\vec{r}^{\,B}\|^2 - (\vec{\omega}^{\,B} \cdot \vec{r}^{\,B})^2}{2}\, dV = \frac{\mu\|\vec{V}\|^2}{2} + \int_\Omega \rho \frac{\omega_i^B\,\omega_k^B\,\delta_{ik}\,r_j^B\,r_j^B - \omega_i^B\,r_i^B\,\omega_k^B\,r_k^B}{2}\, dV \\
&= \frac{\mu\|\vec{V}\|^2}{2} + \frac{I_{ik}\omega_i^B\omega_k^B}{2} \quad \text{where} \quad I_{ik} \triangleq \int_\Omega \rho(\delta_{ik}\,r_j^B\,r_j^B - r_i^B\,r_k^B)\, dV,
\end{aligned} \tag{6.78a}$$

where $\mu\|\vec{V}\|^2/2$ is the kinetic energy due to the motion of the center of mass of the solid body, and $I_{ik}\omega_i^B\omega_k^B/2$ is the kinetic energy due to the rotation of the solid body about its center of mass. Note that, if the B frame is taken in the principal axes (in which the inertial tensor I_{ik} is diagonal), then (6.78a) reduces to

$$T = \frac{\mu\|\vec{V}\|^2}{2} + \frac{I_1(\omega_1^B)^2 + I_2(\omega_2^B)^2 + I_3(\omega_3^B)^2}{2}. \tag{6.78b}$$

As in (6.41), the **Lagrangian** of a solid body is again given by $L = T - U$. Considering the **total energy** $E = T + U$, the property of the **conservation of energy** $dE/dt = 0$ given in (6.50) follows again as before.

In §6.2.2.3, the **total angular momentum** of a system of particles was defined as $\vec{M} = \sum_a \vec{r}_a \times \vec{p}_a$ where, following (6.49), the momentum of each particle is $\vec{p}_a = m_a \vec{v}_a$; passing to the continuüm limit, noting (B.21), the definition of I_{ik} in (6.78a), and that $\int_\Omega \rho \vec{r}^{\,A}\, dV = 0$, this definition may be extended to solid bodies by taking

$$\begin{aligned}
\vec{M} &= \int_\Omega \rho\, \vec{r}^{\,E} \times \vec{v}^{\,E}\, dV = \int_\Omega \rho\, \vec{r}^{\,E} \times (\vec{V} + \vec{\omega}^{\,A} \times \vec{r}^{\,A})\, dV \\
&= \int_\Omega \frac{\rho\, \vec{r}^{\,E}}{\mu}\, dV \times (\mu \vec{V}) + \int_\Omega \rho[\vec{\omega}^{\,A}((\vec{R} + \vec{r}^{\,A}) \cdot \vec{r}^{\,A}) - \vec{r}^{\,A}((R + \vec{r}^{\,A}) \cdot \vec{\omega}^{\,A})]\, dV \\
&= \vec{R} \times \vec{P} + B \int_\Omega \rho[\vec{\omega}^{\,B}(\vec{r}^{\,B} \cdot \vec{r}^{\,B}) - \vec{r}^{\,B}(\vec{r}^{\,B} \cdot \vec{\omega}^{\,B})]\, dV \\
&= \vec{R} \times \vec{P} + \vec{M}^{\,A} \quad \text{where} \quad \vec{M}^{\,A} = B\, \vec{M}^{\,B} \quad \text{and} \quad M_i^B = I_{ik} \omega_k^B,
\end{aligned} \tag{6.79}$$

where $\vec{R} \times \vec{P}$ is the angular momentum due to the motion of the center of mass of the body, $\vec{M}^{\,A}$ is the intrinsic angular momentum due to the rotation of the body about its center of mass in the nonrotating reference frame A, and $\vec{M}^{\,B}$ is the intrinsic angular momentum in the body frame B. Note that, if the B frame is taken in the principal axes (in which the inertial tensor I_{ik} is diagonal), then $\vec{M}^{\,B}$ reduces to

$$M_1^B = I_1 \omega_1^B, \quad M_2^B = I_2 \omega_2^B, \quad M_3^B = I_3 \omega_3^B. \tag{6.80}$$

The property of the **conservation of angular momentum** $d\vec{M}/dt = 0$ given in (6.51) follows as before; it follows that the squared magnitude of the angular momentum, $\|\vec{M}\|^2$, is also conserved.

6.4.2 Lagrange's equations of motion for a solid body in an external field

In this section, we develop the equations of motion for the instantaneous translation and rotation of a solid body in an inertial (nonrotating, nonaccelerating) reference frame E.

Returning to (6.42), noting $\vec{p}_a = m_a \vec{v}_a$, summing over each particle, and taking $\vec{P} = \sum_a \vec{p}_a = \mu \vec{V} = \mu\, d\vec{R}/dt$, it follows that the equation of motion for the translation of a solid body in the reference frame E is simply

$$\frac{d\vec{P}}{dt} = \vec{F} = \sum_a \vec{f}_a. \tag{6.81a}$$

Note that the forces accounted for in the sum above may be taken as the externally-applied forces on the system only, as the internally-generated forces cancel when computing the sum. Taking U as the potential energy of the solid body in an external field (i.e., in an open system) and considering an infinitesimal translation of the entire body through a distance $\delta\vec{R}$, noting from (6.42) that $\vec{f}_a = -\partial U/\partial \vec{r}_a$ and from (6.41) that $L = T(\mathbf{v}) - U(\mathbf{r})$, the corresponding change in the potential energy may be written

$$\delta U = \sum_a \frac{\partial U}{\partial \vec{r}_a} \cdot \delta \vec{r}_a = \Big(\sum_a \frac{\partial U}{\partial \vec{r}_a}\Big) \cdot \delta \vec{R} = -\Big(\sum_a \vec{f}_a\Big) \cdot \delta \vec{R} = -\vec{F} \cdot \delta \vec{R} \quad \Rightarrow \quad \vec{F} = -\frac{\partial U}{\partial \vec{R}} = \frac{\partial L}{\partial \vec{R}}. \tag{6.81b}$$

Noting from and (6.78) that $\partial L/\partial \vec{V} = \partial T/\partial \vec{V} = \mu \vec{V} = \vec{P}$, it is seen that (6.81a) may be interpreted as a direct consequence of Lagrange's equation for the coördinates of the center of mass, $d(\partial L/\partial \vec{V})/dt = \partial L/\partial \vec{R}$.

We now consider $\vec{r}_a \times$ (6.42). To simplify the derivation, we will restrict the reference frame E such that the center of mass is centered at the origin, at zero velocity, at the instant considered. Recall from §6.2.2.3 that $\vec{M}^E = \sum_a \vec{r}_a \times \vec{p}_a$. It follows that $d\vec{M}^E/dt = \sum_a (d\vec{r}_a/dt) \times \vec{p}_a + \sum_a \vec{r}_a \times (d\vec{p}_a/dt)$; since $d\vec{r}_a/dt$ and $\vec{p}_a$ point the same direction, the first term in this sum is zero. Recalling from (6.42) that $\vec{f}_a = d\vec{p}_a/dt$, it follows that the instantaneous equation of motion for the rotation of a solid body in the inertial frame E is simply

$$\frac{d\vec{M}^E}{dt} = \vec{K} = \sum_a \vec{r}_a \times \vec{f}_a. \tag{6.82a}$$

Note that the moments accounted for in the sum above may be taken as the externally-applied moments on the system only, as the internally-generated moments cancel when computing the sum. As in §6.2.2.3, consider again the rotation of a solid body by the vector $\delta\vec{\phi}$, where the magnitude of this vector is the (infinitesimal) angle of rotation $\delta\phi$, and the direction of this vector is the axis of rotation, using the right-hand rule. As derived there, we again have $\delta L = \delta\vec{\phi} \cdot \frac{d}{dt} \sum_a \vec{r}_a \times \vec{p}_a$; considering L for an open system, however, $\delta L \neq 0$ in general. Taking $L = T(\mathbf{v}) - U(\mathbf{r})$ and noting (6.82a), we instead have

$$\vec{K} = -\frac{\partial U}{\partial \vec{\phi}} = \frac{\partial L}{\partial \vec{\phi}}. \tag{6.82b}$$

Noting from (6.78b) that, in principle coordinates, $\partial L/\partial\vec{\omega}^E = \partial T/\partial\vec{\omega}^E = \vec{M}^E$ where $M_i^E = I_i\omega_i^E$, it is seen that (6.82a) may be interpreted as a direct consequence of Lagrange's equation for the rotation of the body about center of mass, $d(\partial L/\partial\vec{\omega}^E)/dt = \partial L/\partial\vec{\phi}$.

Together, (6.81)-(6.82) are referred to as **Lagrange's equations** for the instantaneous translation and rotation of a solid body with applied forces and moments, such as those arising from an external field, in inertial coordinates. Recalling the restriction on the inertial reference frame E that led to the simple form for the instantaneous equation of motion for the rotation of the solid body given in (6.82a), this equation can not immediately be integrated in time to develop an evolution equation for the long-time evolution of the orientation of the body. This shortcoming is addressed in §6.4.3, where we develop the equations of motion for a solid body in the body frame B

Example 6.14 Momentum, energy, and angular momentum conservation of a free solid body Consider a solid body moving freely in space. By the **conservation of momentum**, we have

$$\frac{d\vec{P}}{dt} = 0; \tag{6.83a}$$

that is, the center of mass of the solid body moves at a constant speed in a straight line. Fixing the center of the E frame at the center of mass of the solid body, we thus have $\vec{R} = \vec{V} = \vec{P} = 0$. Taking the B frame in the principal axes, the inertial tensor is diagonal (w.l.o.g., we take $I_1 \geq I_2 \geq I_3 \geq 0$). By the **conservation of energy**, noting (6.80), we have

$$\frac{dT}{dt} = 0 \quad \text{where} \quad 2T = \frac{(M_1^B)^2}{I_1} + \frac{(M_2^B)^2}{I_2} + \frac{(M_3^B)^2}{I_3}; \tag{6.83b}$$

that is, the total energy T is constant. Finally, by the **conservation of angular momentum**, we have

$$\frac{d\vec{M}}{dt} = 0 \quad \text{where} \quad \vec{M} = B\,\vec{M}^{\,B} \quad \text{and} \quad M_i^B = I_{ik}\omega_k^B; \tag{6.83c}$$

that is, the total angular momentum $\vec{M}$ is constant. It follows that $\|\vec{M}\|$ is also constant, and thus

$$\frac{dM^2}{dt} = 0 \quad \text{where} \quad M^2 = (M_1^B)^2 + (M_2^B)^2 + (M_3^B)^2. \tag{6.84}$$

The fact that both (6.83b) and (6.84) must be satisfied simultaneously limits the three components of $\vec{M}^{\,B}$ to move on the intersection of the *ellipsoid* defined by the energy conservation constraint (6.83b) and the *sphere* defined by the squared magnitude of the angular momentum conservation constraint (6.84). This intersection is illustrated in Figure 6.11 for three different solid bodies. For the asymmetric top (top row) and the elongated symmetric top (middle row), it is seen that, in the nearly maximal energy configuration possible for a given value of M^2 (left), the momentum vector in body coordinates wobbles slightly around the **minor principal axis** M_3^B; further, as the energy of rotation is dissipated (reduced), this wobble is magnified. For the asymmetric top (top row) and the flattened symmetric top (bottom row), it is seen that, in the nearly minimal energy configuration possible for a given value of M^2 (right), the momentum vector in body coordinates wobbles slightly around the **major principal axis** M_1^B; further, as the energy of rotation is dissipated, this wobble is diminished. For all three mass distributions considered, it is evident that small perturbations from spinning about the **intermediate principal axis** M_2^B leads to a large deviation of the $\vec{M}^{\,B}$ vector.

The mass distribution of the elongated symmetric top considered in the middle row of subfigures in Figure 6.11 is approximately that of America's first satellite, Explorer 1, illustrated in Figure 6.11. For the purpose of computing its principle moments of intertia, we we may idealize this satellite as a uniform cylinder with mass $m = 13.37$ kg, length $h = 2.05$ m, and radius $r = 0.0825$ m; the principal moments of its inertial tensor are thus $I_1 = I_2 = m(3\,r^2 + h^2)/12 = 4.705$ and $I_3 = m\,r^2/2 = 0.0455$. This satellite was spin stabilized about its minor principal axis, M_3^B, in the nearly maximal energy configuration possible for the prescribed value of M^2. As seen in Figure 6.11, Explorer 1 had four small whip antennae. As the momentum vector wobbled slightly, these antennae deformed, generating heat and thereby gradually dissipating the energy of rotation. As the energy of rotation dissipated towards the minimal energy configuration possible for the prescribed value of M^2, this wobble was magnified until eventially Explorer 1 was tumbling, and the $\vec{M}^{\,B}$ vector was rotating between the M_1^B and M_2^B directions (recall of course that the $\vec{M}^{\,A}$ vector remains constant). Since such gradual dissipation of the energy of rotation is essentially inevitable (liquid fuel sloshing in tanks is another common source of energy dissipation), all subsequent satellites have been constructed as either asymmetric tops or flattened symmetric tops, and are spin stabilized about their *major* principal axis M_1^B.

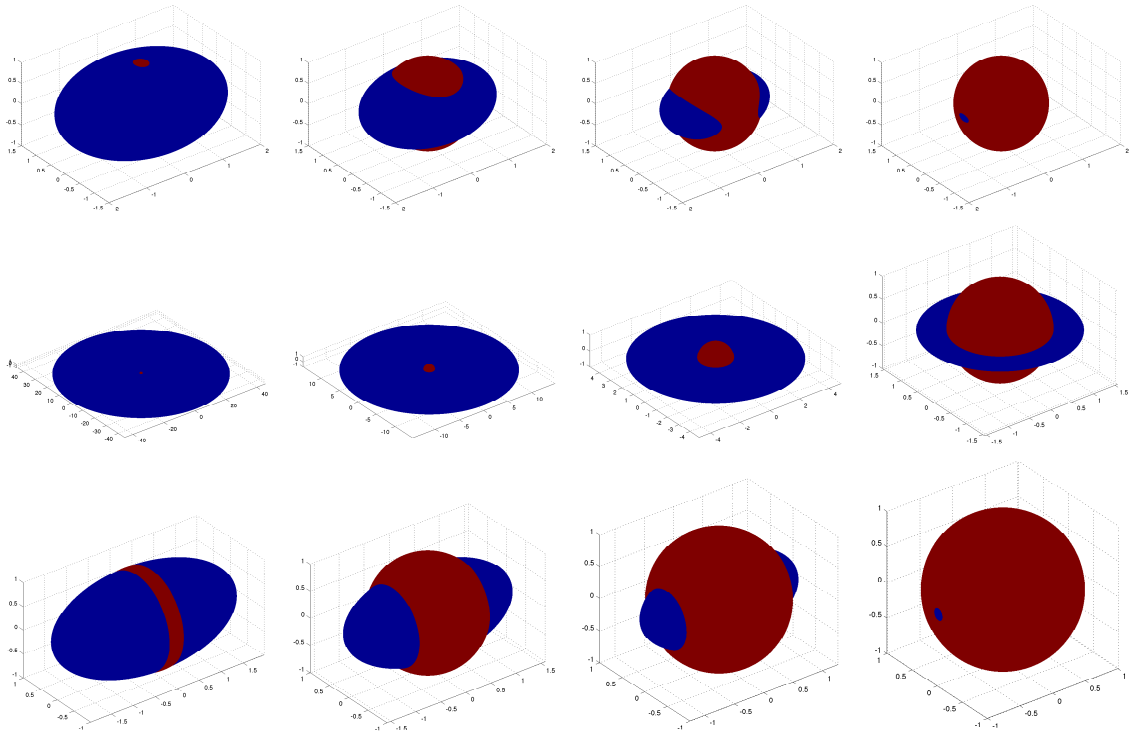


Figure 6.11: The intersection (a.k.a. **polhode**), in the space of $\{M_1^B, M_2^B, M_3^B\}$ (the **minor principal axis** M_3^B is up in each figure), of the ellipsoid defined by (blue) the energy conservation constraint (6.83b) and (red) the sphere defined by the squared magnitude of the angular momentum conservation constraint (6.84) for three different solid bodies: (top) an asymmetric top with $I_1 = 4$, $I_2 = 3$, and $I_3 = 2$, (middle) an elongated symmetric top (the Explorer 1 satellite illustrated in Figure 6.11) with $I_1 = I_2 = 4.705$ and $I_3 = 0.0455$, and (bottom) a flattened symmetric top with $I_1 = 4$, $I_2 = I_3 = 2$, in (left) the nearly maximal energy configuration possible for a given value of M^2, (center) intermediate energy configurations, and (right) the nearly minimal energy configuration possible for a given value of M^2.

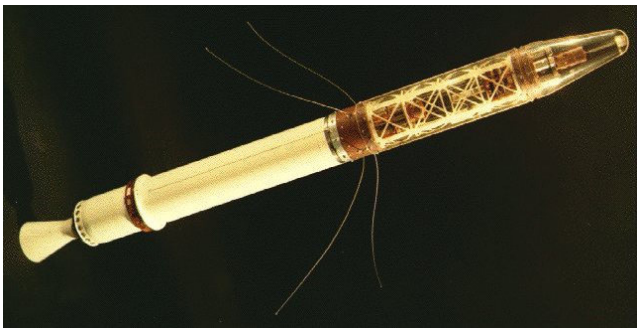


Figure 6.12: The Explorer 1 satellite, an elongated symmetric top with $I_1 = I_2 = 4.705$ and $I_3 = 0.0455$. Spin stabilization of such a body about its axis of symmetry (i.e., it's *minor* principal axis M_3^B) is problematic; as the energy of rotation is dissipated (by heat generation in the whip antennae, etc) while the magnitude of the angular momentum is conserved, the body will begin to tumble. Constructing satellites as asymmetric or flattened symmetric tops, and spin stabilizing about the *major* principal axis M_1^B, is thus preferred.

6.4.3 Euler's equations of motion for a solid body in an external field

(This section still under construction.)

In the frame of a rotating rigid body, Euler's equations for the motion of body are

$$\mu\frac{d\mathbf{V}}{dt} + \boldsymbol{\Omega}\times(\mu\,\mathbf{V}) = \mathbf{F} \qquad (6.85) \qquad\qquad I\frac{d\boldsymbol{\Omega}}{dt} + \boldsymbol{\Omega}\times(I\,\boldsymbol{\Omega}) = \mathbf{K} \qquad (6.86)$$

where I is the inertial tensor (computed in some convenient body-fixed coördinates), and $\boldsymbol{\Omega}$ and $\mathbf{K}$ are, respectively, the rate of rotation and torque applied around these coördinate directions.

In the special case that the coördinate directions are aligned with the principal coördinate directions of the body, Euler's equations (6.85)-(6.86) reduce to:

$$\mu\Big(\frac{dV_1}{dt} + \Omega_2 V_3 - \Omega_3 V_2\Big) = F_1, \quad (6.87a) \qquad\qquad I_1\frac{d\Omega_1}{dt} + (I_3 - I_2)\Omega_2\Omega_3 = K_1, \quad (6.88a)$$

$$\mu\Big(\frac{dV_2}{dt} + \Omega_3 V_1 - \Omega_1 V_3\Big) = F_2, \quad (6.87b) \qquad\qquad I_2\frac{d\Omega_2}{dt} + (I_1 - I_3)\Omega_3\Omega_1 = K_2, \quad (6.88b)$$

$$\mu\Big(\frac{dV_3}{dt} + \Omega_1 V_2 - \Omega_2 V_1\Big) = F_3, \quad (6.87c) \qquad\qquad I_3\frac{d\Omega_3}{dt} + (I_2 - I_1)\Omega_1\Omega_2 = K_3. \quad (6.88c)$$

where I_1, I_2, and I_3 are the principal moments of inertia of the body.

Transforming the instantaneous rotation rates of the body $\{\Omega_1, \Omega_2, \Omega_3\}$, which is measured directly by the rate gyros in the Body frame, into the rate of change of the Euler angles or the Tait-Bryan angles is a bit involved.

Evolution equation for the 3-2-1 Tait-Bryan rotation sequence

Evolution equation for the 3-1-3 Euler rotation sequence

Evolution equation for a quaternion representation

Taking the time derivative of (??), applying the product rule of differentiation, and substituting in (??) and (??) results in a nonlinear equation of the form $\ddot{\mathbf{q}} = \mathbf{f}(\mathbf{q}, \dot{\mathbf{q}}, t)$:

$$\begin{aligned}\ddot{\mathbf{q}} &= [\dot{\mathbf{q}}\vec{\Omega} + \mathbf{q}\dot{\vec{\Omega}}]/2 = [\dot{\mathbf{q}}\vec{\Omega} + \mathbf{q}I^{-1}(\mathbf{K} - \vec{\Omega}\times(I\,\vec{\Omega})]/2 \\ &= \dot{\mathbf{q}}\,\mathbf{q}^*\,\dot{\mathbf{q}} + \mathbf{q}\,I^{-1}(\mathbf{K} - 4\mathbf{q}^*\dot{\mathbf{q}}\times(I\,\mathbf{q}^*\dot{\mathbf{q}})/2 \triangleq \mathbf{g}(\mathbf{q}, \dot{q}, t).\end{aligned}$$

6.4.4 Frictional losses

Example 6.15 A spinning top *??*

Chapter 7

Numerical Methods

7.1 Interpolation

In **interpolation** problems, we aspire to draw an "appropriately smooth" curve which passes exactly through a set of available datapoints in one or more dimensions, as illustrated in Figure 7.1. This problem description is subject to a significant degree of interpretation; only a few such interpretations will be discussed here.

Interpolation is a foundational idea in numerics that is useful when, e.g., developing differentiation and integration strategies, estimating the value of a function between known values, producing computer-generated imagery (**CGI**), etc.

Note specifically that the process of interpolation passes a curve *exactly* through each datapoint. This is sometimes what is desired. However, if the data is from an experiment and has any appreciable uncertainty associated with it, then it is preferred to take many measurements and use a least-squares technique to fit a low-order curve in the general vicinity of several datapoints, as discussed in the data fitting framework described in §2 of *NR*. This technique minimizes a weighted sum of the square distance from each datapoint to this curve without forcing the curve to pass through each datapoint individually, and generally produces a much smoother curve (and a more physically-meaningful result) when the available data is noisy.

7.1.1 Linear spline interpolation

Linear spline interpolation amounts to nothing more than the game of **Connect the Dots**, using straight line segments between each pair of points. Implementation (see RR_LinearSpline) is straightforward, and provides a reference solution against which improved interpolation schemes may be compared.

7.1.2 Lagrange interpolation: n'th-order polynomials fitting $n+1$ datapoints

Suppose we have a set of $n+1$ datapoints $\{x_i, y_i\}$. The process of Lagrange interpolation fits an n'th degree polynomial (that is, a polynomial with $n+1$ degrees of freedom) exactly through this data. There are two ways of accomplishing this: solve a system of $n+1$ simultaneous equations for the $n+1$ coefficients of this polynomial, or construct the polynomial directly in factored form.

Solving $n+1$ simultaneous equations for the $n+1$ coefficients

Consider the polynomial

$$P(x) = a_o + a_1\, x + a_2\, x^2 + \ldots + a_n\, x^n.$$

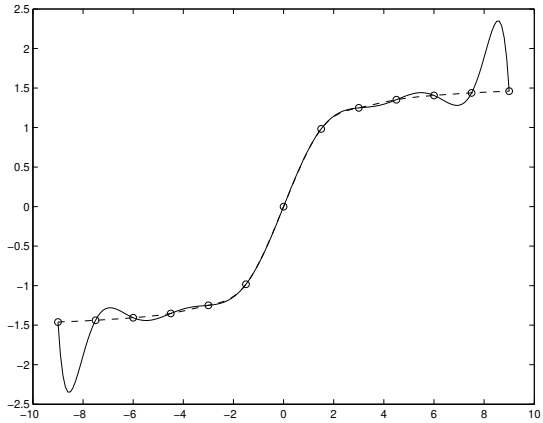


Figure 7.1: The interpolation problem: fit a curve of the specified form to intersect $n+1$ points (○); the solutions illustrated are (——) the Lagrange interpolant and (- - - -) the cubic spline interpolant with parabolic run-out. Lagrange interpolation often gives a spurious result when the number of datapoints is large.

At each point x_i, the polynomial has the value y_i; that is,

$$y_i = P(x_i) = a_o + a_1\, x_i + a_2\, x_i^2 + \ldots + a_n\, x_i^n \qquad \text{for} \qquad i = 0, 1, 2, \ldots, n.$$

In matrix form, we may write this system as

$$\underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix}}_{V} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}}_{\mathbf{a}} = \underbrace{\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}}_{\mathbf{y}}. \tag{7.1}$$

This system is of the form $V\mathbf{a} = \mathbf{y}$, where V is commonly referred to as **Vandermonde's matrix**, and may be solved for the vector $\mathbf{a}$ containing the coefficients a_i of the desired polynomial. Vandermonde's matrix is often poorly conditioned, and thus this technique of finding an interpolating polynomial is unreliable.

Constructing the polynomial directly

Consider the n'th degree polynomial given by the factored expression

$$L_\kappa(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{\kappa-1})(x-x_{\kappa+1})\cdots(x-x_n)}{(x_\kappa-x_0)(x_\kappa-x_1)\cdots(x_\kappa-x_{\kappa-1})(x_\kappa-x_{\kappa+1})\cdots(x_\kappa-x_n)} = \prod_{\substack{i=0 \\ i\neq\kappa}}^{n} \frac{x-x_i}{x_\kappa-x_i}. \tag{7.2a}$$

Note that, by construction,

$$L_\kappa(x_i) = \delta_{i\kappa} = \begin{cases} 1 & i = \kappa, \\ 0 & i \neq \kappa. \end{cases}$$

Scaling this result, the polynomial $y_\kappa\, L_\kappa(x)$ (no summation implied) passes through zero at every datapoint $x = x_i$ except at $x = x_\kappa$, where it has the value y_κ. Finally, a linear combination of $n+1$ of these polynomials

$$P(x) = \sum_{\kappa=0}^{n} y_\kappa L_\kappa(x) \tag{7.2b}$$

provides an n'th degree polynomial which exactly passes through *all* of the datapoints, by construction. To verify, note that $P(x_\iota) = \sum_{\kappa=0}^{n} y_\kappa \delta_{\iota\kappa} = y_\iota$ as required. Implementation of this constructive technique to determine the interpolating polynomial is given in RR_Lagrange).

Unfortunately, if the number of datapoints is large, high-order polynomials sometimes meander significantly between the datapoints even if the data appears to be fairly regular, as shown in Figure 7.1. Thus, Lagrange interpolation should be thought of as dangerous for anything more than a few datapoints, and should be avoided in favor of other techniques, such as the cubic spline interpolation technique discussed below.

7.1.3 Piecewise cubic interpolation

Instead of forcing a high-order polynomial through the entire dataset, we may instead construct a continuous, smooth, piecewise cubic function through the data. We will first construct this function to be smooth in the sense of having continuous first and second derivatives at each datapoint. These conditions, together with the appropriate conditions at each end, uniquely determine a piecewise cubic function through the data which is usually reasonably smooth; we will call this function the **cubic spline interpolant**.

Defining the interpolant in this manner is akin to deforming a single **spline**, or a thin piece of wood or metal, to pass over all of the datapoints plotted on a large block of wood and marked with thin nails. The elasticity equation governing the deformation f of such a spline is

$$f'''' = G, \tag{7.3a}$$

where G is a force localized near each nail which is sufficient to pass the spline through the data. As G is nonzero only in the immediate vicinity of each nail, such a spline takes an approximately piecewise cubic shape between the datapoints. Thus, *between the datapoints*, $f(x)$ is cubic:

$$f''''(x) = 0, \quad f'''(x) = C_1, \quad f''(x) = C_1\,x + C_2, \quad f'(x) = \frac{C_1}{2}x^2 + C_2\,x + C_3, \quad f(x) = \frac{C_1}{6}x^3 + \frac{C_2}{2}x^2 + C_3\,x + C_4. \tag{7.3b}$$

Constructing the cubic spline interpolant

Let $f_i(x)$ denote the cubic in the interval $x_i \le x \le x_{i+1}$ and let $f(x)$ denote the collection of all the cubics for the entire range $x_0 \le x \le x_n$. As noted above, f_i'' varies linearly with x between each datapoint. *At* each datapoint, we would like to piece these cubics together as smoothly as possible, thereby mimicking the physical situation in which the force G localized on the spline near each nail is as smooth as possible; in fact, we have enough flexibilty to impose C^2 continuity, that is:

(a) continuity of the function f, *i.e.*, $f_{i-1}(x_i) = f_i(x_i) = f(x_i) = y_i$,
(b) continuity of the first derivative f', *i.e.*, $f'_{i-1}(x_i) = f'_i(x_i) = f'(x_i)$, and
(c) continuity of the second derivative f'', *i.e.*, $f''_{i-1}(x_i) = f''_i(x_i) = f''(x_i)$.

We now describe a procedure to determine an f which satisfies conditions (a) and (c) by *construction*, in a manner analogous to the construction of the Lagrange interpolant in §7.1.2, and which satisfies condition (b) by setting up and solving the appropriate system of equations for the value of f'' at each datapoint x_i.

To begin the constructive procedure for determining f, note that on each interval $x_i \le x \le x_{i+1}$ for $i = 0, 1, \ldots, n-1$, we may write a linear equation for $f_i''(x)$ as a function of its value at the endpoints, $f''(x_i)$ and $f''(x_{i+1})$, which are (as yet) undetermined. The following form (which is linear in x) fits the bill by construction:

$$f_i''(x) = f''(x_i)\frac{x - x_{i+1}}{x_i - x_{i+1}} + f''(x_{i+1})\frac{x - x_i}{x_{i+1} - x_i}. \tag{7.4}$$

Note that this first degree polynomial is in fact just a Lagrange interpolation of the two datapoints $\{x_i, f''(x_i)\}$ and $\{x_{i+1}, f''(x_{i+1})\}$ [see (7.2), for $n = 1$]. By construction, condition (c) is satisfied. Integrating this equation twice and defining $\Delta_i = x_{i+1} - x_i$, it follows that

$$f_i'(x) = -\frac{f''(x_i)}{2}\frac{(x_{i+1}-x)^2}{\Delta_i} + \frac{f''(x_{i+1})}{2}\frac{(x-x_i)^2}{\Delta_i} + C_1,$$
$$f_i(x) = \frac{f''(x_i)}{6}\frac{(x_{i+1}-x)^3}{\Delta_i} + \frac{f''(x_{i+1})}{6}\frac{(x-x_i)^3}{\Delta_i} + C_1 x + C_2.$$

The undetermined constants of integration are obtained by matching the end conditions

$$f_i(x_i) = y_i \qquad \text{and} \qquad f_i(x_{i+1}) = y_{i+1}.$$

A convenient way of constructing the linear and constant terms in the expression for $f_i(x)$ in such a way that the desired end conditions are met is by writing $f_i(x)$ in the form

$$f_i(x) = \frac{f''(x_i)}{6}\left(\frac{(x_{i+1}-x)^3}{\Delta_i} - \Delta_i(x_{i+1}-x)\right) + \frac{f''(x_{i+1})}{6}\left(\frac{(x-x_i)^3}{\Delta_i} - \Delta_i(x-x_i)\right) + y_i\frac{(x_{i+1}-x)}{\Delta_i} + y_{i+1}\frac{(x-x_i)}{\Delta_i}, \qquad \text{where} \qquad x_i \le x \le x_{i+1}. \tag{7.5}$$

By construction, condition (a) is satisfied. Finally, an expression for $f_i'(x)$ may now be found by differentiating this expression for $f_i(x)$, which gives

$$f_i'(x) = \frac{f''(x_i)}{6}\left(-3\frac{(x_{i+1}-x)^2}{\Delta_i} + \Delta_i\right) + \frac{f''(x_{i+1})}{6}\left(3\frac{(x-x_i)^2}{\Delta_i} - \Delta_i\right) + \frac{y_{i+1}}{\Delta_i} - \frac{y_i}{\Delta_i}.$$

The second derivative of f at each node, $f''(x_i)$, is still undetermined. A system of equations from which the $f''(x_i)$ may be found is obtained by imposing condition (b), which is achieved by setting

$$f_i'(x_i) = f_{i-1}'(x_i) \qquad \text{for} \qquad i = 1, 2, \ldots, n-1.$$

Substituting appropriately from the above expression for $f_i'(x)$, noting that $\Delta_i = x_{i+1} - x_i$, leads to

$$\frac{\Delta_{i-1}}{6} f''(x_{i-1}) + \frac{\Delta_{i-1}+\Delta_i}{3} f''(x_i) + \frac{\Delta_i}{6} f''(x_{i+1}) = \frac{y_{i+1}-y_i}{\Delta_i} - \frac{y_i - y_{i-1}}{\Delta_{i-1}} \tag{7.6}$$

for $i = 1, 2, \ldots, n-1$. This is a diagonally-dominant tridiagonal system of $n-1$ equations for the $n+1$ unknowns $f''(x_0)$, $f''(x_1)$, …, $f''(x_n)$. We find the two remaining equations by prescribing conditions on the interpolating function at each end. We will consider three types of end conditions:

- parabolic run-out: $f''(x_0) = f''(x_1)$ and $f''(x_n) = f''(x_{n-1})$;
- free run-out (also known as natural splines): $f''(x_0) = 0$ and $f''(x_n) = 0$; or
- periodic end conditions: $f''(x_0) = f''(x_{n-1})$ and $f''(x_1) = f''(x_n)$.

Equation (7.6) may be taken together with the appropriate choice of end conditions (depending upon the problem at hand) to give $n+1$ equations for the $n+1$ unknowns $f''(x_i)$. This set of equations is then solved for the $f''(x_i)$, which thereby ensures that condition (b) is satisfied. Once this system is solved for the $f''(x_i)$, the cubic spline interpolant follows immediately from (7.5).

Note that, when (7.6) is taken together with parabolic or free run-out at the ends, a tridiagonal system results which can be solved efficiently with the Thomas algorithm. When (7.6) is taken together periodic end

conditions, a tridiagonal circulant system $A\mathbf{x} = \mathbf{b}$ results with $a_{1,1} = 0$. A pair of codes which sets up and solves these systems with any of the above three end conditions is given in RR_CubicSplineSetup and RR_CubicSpline.

Applying periodic end conditions to develop a spline for a system that is not well approximated as periodic can lead to significant non-physical meanderings of the interpolant near the ends of the domain; thus, periodic end conditions should be reserved for systems which are actually periodic. On the other hand, parabolic run-out extends a parabolic curve between x_0 and x_1, and free run-out tapers the curvature of the interpolant down to zero near the endpoints; both of these choices usually generate reasonably smooth interpolants.

Tension splines

For certain interpolation problems, cubic splines aren't adequately smooth. In such problems, it is helpful to use **tension splines**, which are cubic splines with the mechanical equivalent of a bit of tension added to straighten out the curvature between the datapoints. As the tension gets large in this approach, the interpolant approaches a piecewise linear function. Tensioned splines obey the differential equation [cf. (7.3a)]:

$$f'''' - \sigma^2 f'' = G$$

where σ is the tension of the spline. This leads to the following relationships between the datapoints [cf. (7.3b)]:

$$[f'' - \sigma^2 f]'' = 0, \qquad [f'' - \sigma^2 f]' = C_1, \qquad [f'' - \sigma^2 f] = C_1\, x + C_2.$$

Solving the ODE on the right leads to an equation of the form [cf. (**??**)]

$$f = -\sigma^{-2}(C_1 x + C_2) + C_3 e^{-\sigma x} + C_4 e^{\sigma x}.$$

Proceeding with a constructive process to satisfy condition (a) analogous to that used previously, we assemble the linear and constant terms of $f'' - \sigma^2 f$ such that [cf. (7.4)]

$$\left[f_i''(x) - \sigma^2 f_i(x)\right] = \left[f_i''(x_i) - \sigma^2 y_i\right] \frac{x - x_{i+1}}{x_i - x_{i+1}} + \left[f_i''(x_{i+1}) - \sigma^2 y_{i+1}\right] \frac{x - x_i}{x_{i+1} - x_i}.$$

Similarly, we assemble the exponential terms in the solution of this ODE for f in a constructive manner such that condition (c) is satisfied. Rewriting the exponentials as $\sinh$ functions, the desired solution may be written [cf. (7.5)]

$$\begin{aligned} f_i(x) = -\sigma^{-2}\Big\{ &\left[f''(x_i) - \sigma^2 y_i\right] \frac{x_{i+1} - x}{\Delta_i} + \left[f''(x_{i+1}) - \sigma^2 y_{i+1}\right] \frac{x - x_i}{\Delta_i} - f''(x_i) \frac{\sinh \sigma(x_{i+1} - x)}{\sinh \sigma\Delta_i} \\ &- f''(x_{i+1}) \frac{\sinh \sigma(x - x_i)}{\sinh \sigma\Delta_i} \Big\} \qquad \text{where} \qquad x_i \le x \le x_{i+1}. \end{aligned} \tag{7.7}$$

Differentiating once and appling condition (b) leads to the tridiagonal system [cf. (7.6)]

$$\begin{aligned} &\left(\frac{1}{\Delta_{i-1}} - \frac{\sigma}{\sinh \sigma\Delta_{i-1}}\right) \frac{f''(x_{i-1})}{\sigma^2} - \left(\frac{1}{\Delta_{i-1}} - \frac{\sigma \cosh \sigma\Delta_{i-1}}{\sinh \sigma\Delta_{i-1}} + \frac{1}{\Delta_i} - \frac{\sigma \cosh \sigma\Delta_i}{\sinh \sigma\Delta_i}\right) \frac{f''(x_i)}{\sigma^2} \\ &\qquad + \left(\frac{1}{\Delta_i} - \frac{\sigma}{\sinh \sigma\Delta_i}\right) \frac{f''(x_{i+1})}{\sigma^2} = \frac{y_{i+1} - y_i}{\Delta_i} - \frac{y_i - y_{i-1}}{\Delta_{i-1}}. \end{aligned} \tag{7.8}$$

The tridiagonal system (7.8) can be set up and solved exactly as was done with (7.6), even though the coefficients have a slightly more complicated form. The tensioned-spline interpolant is then given by (7.7).

B-splines

We may easily express the cubic spline (or tension spline) interpolant in a form similar to our construction of the Lagrange interpolant, that is,

$$f(x) = \sum_{\kappa=0}^{n} y_\kappa b_\kappa(x),$$

where the **basis functions** $b_\kappa(x)$ are spline interpolations of Kronecker delta functions such that $b_\kappa(x_i) = \delta_{i\kappa}$, as discussed in §7.1.2 for the functions $L_\kappa(x)$. The basis functions so constructed are found to have **localized support** (in other words, $b_\kappa(x) \to 0$ for large $|x - x_\kappa|$).

By relaxing some of the continuity constraints, we may confine each of the basis functions to have **compact support** (i.e., we can set $b_\kappa(x) = 0$ exactly for $|x - x_\kappa| > R$ for some R). With such functions, it is easier both to compute the interpolations themselves and to project the interpolated function onto a different grid of points.

Cubic Hermite interpolation

Cubic spline interpolants $\tilde{f}(x)$ are C^2 continuous, and pass through the given function values at the datapoints, $f(x_i)$. This is achieved by selecting appropriately the the first and second derivatives of the interpolant at the datapoints, $\tilde{f}'(x_i)$ and $\tilde{f}''(x_i)$.

On the other hand, if the values of both the function and its derivative, $f(x_i)$ and $f'(x_i)$, are specified at the datapoints, then a C^1 continuous **Cubic Hermite interpolant** $\tilde{f}(x)$ may be fit to this data in a simple fashion using basis functions with compact support. This approach is in fact a variant of the B-spline approach; defining $\tilde{x} = (x - x_i)/(x_{i+1} - x_i)$, it produces an interpolant $\tilde{f}(x)$ on each interval $x \in [x_i, x_{i+1}]$ such that

$$\tilde{f}(x) = h_{00}(\tilde{x})\, f_i + h_{01}(\tilde{x})\, (x_{i+1} - x_i)\, f'_i + h_{10}(\tilde{x})\, f_{i+1} + h_{11}(\tilde{x})\, (x_{i+1} - x_i)\, f'_{i+1}, \tag{7.9}$$

using the following four simple basis functions on each interval $x \in [x_i, x_{i+1}]$:

$$h_{00}(\tilde{x}) = 2\,\tilde{x}^3 - 3\,\tilde{x}^2 + 1, \quad \text{with } h_{00}(0) = 1 \text{ and } h'_{00}(0) = h_{00}(1) = h'_{00}(1) = 0, \tag{7.10a}$$
$$h_{01}(\tilde{x}) = \tilde{x}^3 - 2\,\tilde{x}^2 + \tilde{x}, \quad \text{with } h'_{01}(0) = 1 \text{ and } h_{01}(0) = h_{01}(1) = h'_{01}(1) = 0, \tag{7.10b}$$
$$h_{10}(\tilde{x}) = -2\,\tilde{x}^3 + 3\,\tilde{x}^2, \quad \text{with } h_{10}(1) = 1 \text{ and } h_{10}(0) = h'_{10}(0) = h'_{10}(1) = 0, \tag{7.10c}$$
$$h_{11}(\tilde{x}) = \tilde{x}^3 - \tilde{x}^2, \quad \text{with } h'_{11}(1) = 1 \text{ and } h_{11}(0) = h'_{11}(0) = h_{11}(1) = 0. \tag{7.10d}$$

7.1.4 Multivariate interpolation of structured data

The interpolation strategies described above are well suited for 1D problems, and can be extended fairly easily to higher dimensions on structured n-dimensional grids. Below we describe two such extensions.

Multilinear interpolation

The idea of 1D linear spline interpolation (see §7.1.1) extends immediately to the **multilinear interpolation** of data defined on an n-dimensional Cartesian grid (that is, function values $f_{i_1,i_2,\ldots,i_n} = f(x_{1_{i_1}}, x_{2_{i_2}}, \ldots, x_{n_{i_n}})$ where $i_1 = 1, \ldots, N_1,\ i_2 = 1, \ldots, N_2$, etc.) as follows:

- Determine the grid cell that new interpolating point $\mathbf{x}$ lies in: that is, find the i_1 through i_n such that $x_{k_{i_k}} \leq x_k \leq x_{k_{(i_k+1)}}$ for $1 \leq k \leq n$.
- Determine the fraction of the distance that the new point $\mathbf{x}$ is across this cell in each direction: that is, compute $\eta_{k,0} = (x_{k_{(i_k+1)}} - x_k)/(x_{k_{(i_k+1)}} - x_{k_{i_k}})$ and $\eta_{k,1} = 1 - \eta_{k,0}$ for $1 \leq k \leq n$.

- Linearly interpolate in each direction independently by setting the interpolant $\tilde{f}(x)$ such that

$$\tilde{f}(x) = \sum_{d_1=0}^{1} \sum_{d_2=0}^{1} \cdots \sum_{d_n=0}^{1} f_{i_1+d_1, i_2+d_2, \ldots, i_n+d_n} \, \eta_{1,d_1} \eta_{2,d_2} \cdots \eta_{n,d_n}.$$

Implementation for $n = 2$ is given in RR_BilinearSpline; see Figure 7.2a for typical results.

Multicubic interpolation

The idea of cubic spline interpolation (see §7.1.3) may be extended in a couple of different ways to data defined on an n-dimensional Cartesian grid.

An accurate and simple approach is to do cubic spline interpolation in each dimension, one at a time:

- First, interpolate onto the specified value of x_1 for each value of x_2 through x_n on the grid (that is, for $i_2 = 1, \ldots, N_2$, $i_3 = 1, \ldots, N_3$, $i_4 = 1, \ldots, N_4$, etc.).
- Then, working only with those function values interpolated onto the specified value of x_1, interpolate onto the specified value of x_2 for each value of x_3 through x_n on the grid (that is, for $i_3 = 1, \ldots, N_3$, $i_4 = 1, \ldots, N_4$, etc.).
- Continue in an analogous fashion through the remaining dimensions, one at a time.

Recall that, in the one-dimensional case described in §7.1.3, the computationally expensive part of setting up the cubic spline interpolant could be computed once during the initialization step, then used for interpolating onto any specified point x. Unfortunately, in the multidimensional approach described above, this is no longer the case, as the interpolations performed in the x_j direction, for $j = 2, \ldots, n$, depend on the data that results from the interpolations performed in the x_1 to x_{j-1} directions. This approach is thus too expensive to be practically useful when interpolating onto a large number of gridpoints.

An inexpensive alternative[1] for extending cubic spline interpolation to n-dimensional grids follows:

(i) First, during an initialization step, approximate all first and cross derivatives of f at each gridpoint where the function f is initially specified. Note that these numerical approximations may be computed by successive cubic spline interpolations along the gridlines, evaluated at the gridpoints.

(ii) Then, as in the multilinear interpolation approach described in §7.1.4, determine (for each new interpolation point) which grid cell that the new interpolation point $\mathbf{x}$ lies in, and the fraction of the distance that the new point $\mathbf{x}$ is across this cell in each coordinate direction.

(iii) Finally, construct a function which is cubic in each coordinate variable and matches the first and cross derivative information computed in step i at each of the corners of the cell identified in step ii.

This idea is best made concrete by example. In the case of $n = 2$ (**bicubic interpolation**), we first use cubic spline interpolation along each of the gridlines to compute $\{f_x, f_y, f_{xy}\}$ at each of the gridpoints where the function values f are initially specified. Then, as in bilinear interpolation, we determine which grid cell that the new interpolating point $\mathbf{x}$ lies in, and the fraction of the distance that the new point $\mathbf{x}$ is across this cell in each direction (denoted here x and y). Finally, the interpolant on the cell is defined by

$$f(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j, \tag{7.11}$$

[1] Recall (from the introduction to §7.1) that the interpolation problem itself is an approximate problem subject to a significant degree of interpretation; it may thus be argued that approximate solution to a problem of this class is good enough, and one should not code up an unduly expensive scheme in order to solve an approximate problem of this class "exactly".

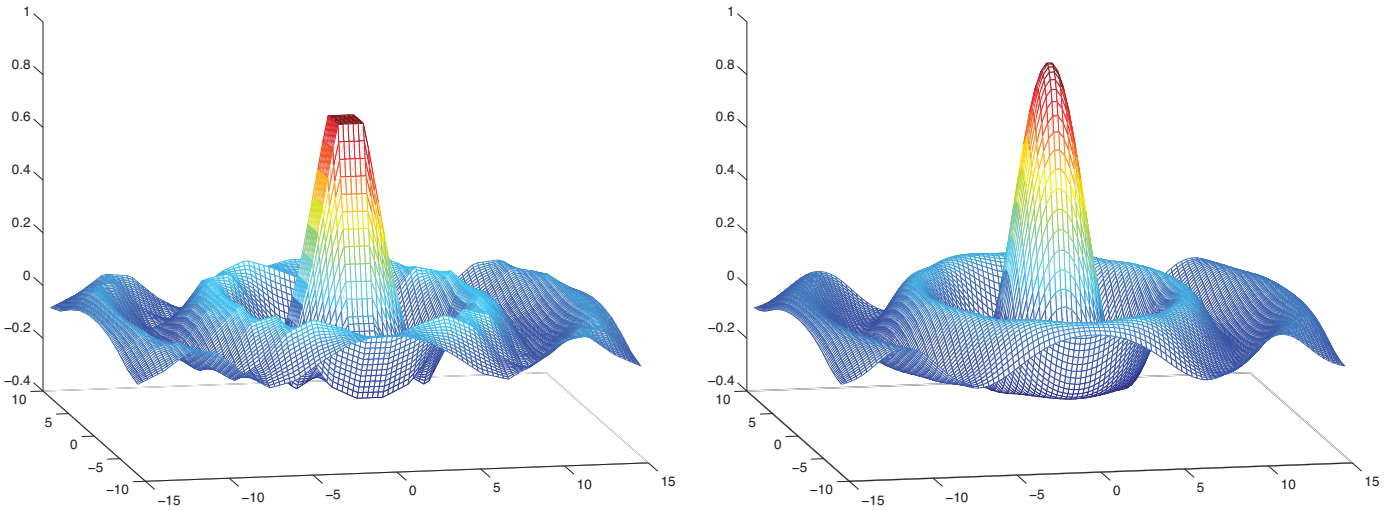


Figure 7.2: (left) Bilinear interpolation and (right) bicubic interpolation of a 10×20 grid of data from the function $\text{sinc}(r) \triangleq \sin(r)/r$ where $r = \sqrt{x^2 + y^2}$; the bilinear case reveals some noticeable artifacts of the interpolation, whereas the bicubic interpolant is visually almost indistinguishable from the original function.

where the a_{ij} for $i = 0, \ldots, 3$ and $j = 0, \ldots, 3$ are selected to match the values of $\{f, f_x, f_y, f_{xy}\}$ at each of the 4 corners of the cell (w.l.o.g, taken here to be the unit square, with corners denoted $\{\mathbf{x}_{00}, \mathbf{x}_{01}, \mathbf{x}_{10}, \mathbf{x}_{11}\}$) that contains the new interpolation point; this results in a linear 16×16 problem of the form $A\mathbf{x} = \mathbf{b}$ where

$$\begin{pmatrix}
1 & & & & & & & & & & & & & & & \\
1 & 1 & 1 & 1 & & & & & & & & & & & & \\
1 & & & & 1 & & & & 1 & & & & 1 & & & \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
 & 1 & & & & & & & & & & & & & & \\
 & 1 & 2 & 3 & & & & & & & & & & & & \\
 & 1 & & & & 1 & & & & 1 & & & & 1 & & \\
 & 1 & 2 & 3 & & 1 & 2 & 3 & & 1 & 2 & 3 & & 1 & 2 & 3 \\
 & & & & 1 & & & & & & & & & & & \\
 & & & & 1 & 1 & 1 & 1 & & & & & & & & \\
 & & & & 1 & & & & 2 & & & & 3 & & & \\
 & & & & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 \\
 & & & & & 1 & & & & & & & & & & \\
 & & & & & 1 & 2 & 3 & & & & & & & & \\
 & & & & & 1 & & & & 2 & & & & 3 & & \\
 & & & & & 1 & 2 & 3 & & 2 & 4 & 6 & & 3 & 6 & 9
\end{pmatrix}
\begin{pmatrix} a_{00} \\ a_{10} \\ a_{20} \\ a_{30} \\ a_{01} \\ a_{11} \\ a_{21} \\ a_{31} \\ a_{02} \\ a_{12} \\ a_{22} \\ a_{32} \\ a_{03} \\ a_{13} \\ a_{23} \\ a_{33} \end{pmatrix} =
\begin{pmatrix} f|_{\mathbf{x}_{00}} \\ f|_{\mathbf{x}_{10}} \\ f|_{\mathbf{x}_{01}} \\ f|_{\mathbf{x}_{11}} \\ f_x|_{\mathbf{x}_{00}} \\ f_x|_{\mathbf{x}_{10}} \\ f_x|_{\mathbf{x}_{01}} \\ f_x|_{\mathbf{x}_{11}} \\ f_y|_{\mathbf{x}_{00}} \\ f_y|_{\mathbf{x}_{10}} \\ f_y|_{\mathbf{x}_{01}} \\ f_y|_{\mathbf{x}_{11}} \\ f_{xy}|_{\mathbf{x}_{00}} \\ f_{xy}|_{\mathbf{x}_{10}} \\ f_{xy}|_{\mathbf{x}_{01}} \\ f_{xy}|_{\mathbf{x}_{11}} \end{pmatrix}.$$

Note that A^{-1} is simple, with all integer entries, and is thus entered directly in the implementation of bicubic interpolation in given in RR_BicubicSplineSetup and RR_BicubicSpline; see Figure 7.2b for typical results.

Similarly, in the case of $n = 3$ (**tricubic interpolation**), we use cubic spline interpolation along each gridline to compute $\{f_x, f_y, f_z, f_{xy}, f_{yz}, f_{xz}, f_{xyz}\}$ at each of the gridpoints where the values of f are initially prescribed. We then determine which grid cell that new interpolating point $\mathbf{x}$ lies in, and the fraction of the distance that the point $\mathbf{x}$ is across this cell in each direction. Finally, the interpolant on the cell is defined by

$$f(x, y, z) = \sum_{i=0}^{3} \sum_{j=0}^{3} \sum_{k=0}^{3} a_{ijk}\, x^i\, y^j\, z^j, \tag{7.12}$$

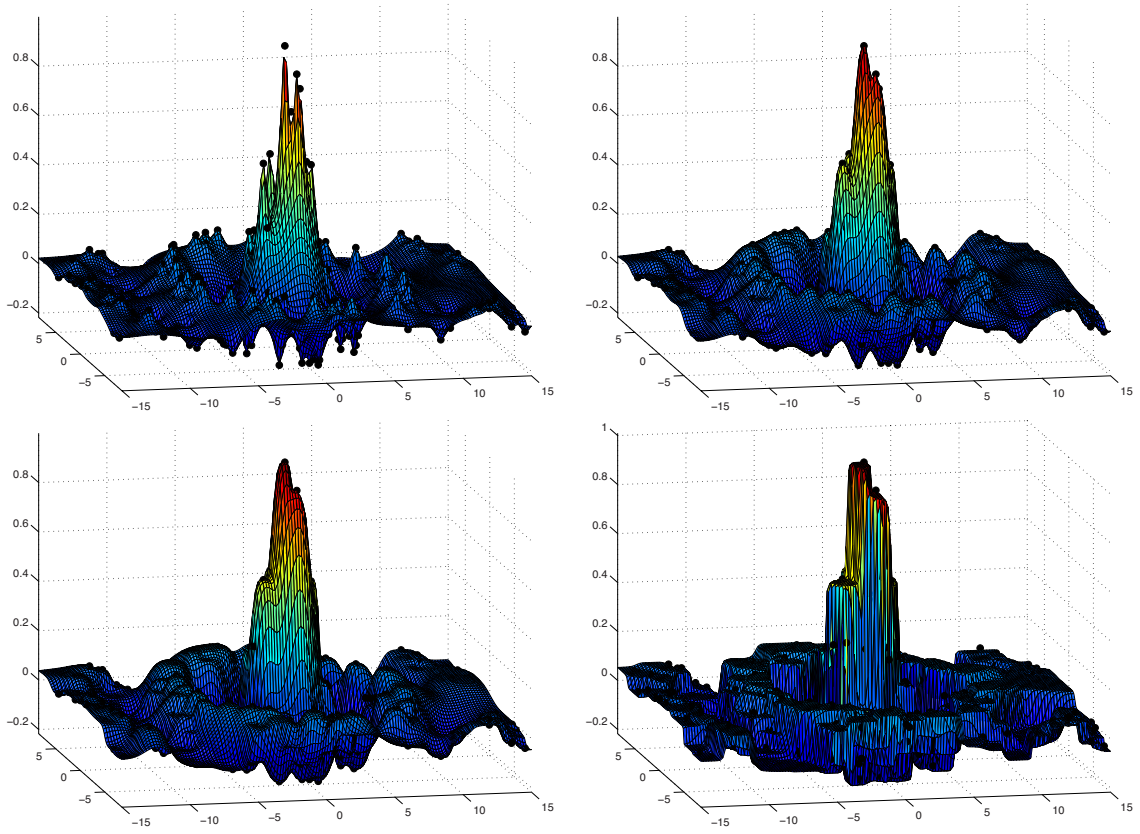


Figure 7.3: Interpolation via inverse distance of 200 points with (a) $p = 2$, (b) $p = 3$, (c) $p = 4$, (d) $p = 20$.

where the a_{ijk} are selected to match the values of $\{f, f_x, f_y, f_z, f_{xy}, f_{yz}, f_{xz}, f_{xyz}\}$ at each of the 8 corners of the cell (w.l.o.g, taken to be a unit cube) that contains the new interpolation point; this results in 64 linear equations for 64 unknowns which may easily be solved.

7.1.5 Multivariate interpolation of unstructured data

The extension of interpolation strategies to unstructured data (that is, for data not lying on a regular grid) requires a bit more effort than the case of structured data considered above; we will thus consider three different approaches to this problem.

7.1.5.1 Interpolation via inverse distance

The simplest approach for approximating the function value f at location $\mathbf{x}$ based on N known function values f_i at various locations $\mathbf{c}_i$, for $i = 1, \ldots, N$, is the **inverse distance** interpolation formulae given by

$$f(\mathbf{x}) = \frac{1}{C} \sum_{\substack{i=1 \\ d_i \leq R}}^{N} f_i / d_i^p \quad \text{where} \quad C = \sum_{\substack{i=1 \\ d_i \leq R}}^{N} 1/d_i^p \quad \text{and} \quad d_i = \|\mathbf{x} - \mathbf{c}^i\|_2,$$

where $1 \leq p < \infty$ is some power and the sum includes all known function values within some prespecified distance R of the point in question, $\mathbf{x}$. Implementation is given in RR_InvDistanceInterp, and typical results are illustrated in Figure 7.3. Note that the minima and maxima of this interpolating function coincide with datapoints representing the largest and smallest function values in the dataset. For small p (e.g., $p = 2$), the interpolant looks like a tent propped up, and pushed down, at the various datapoints; for increasing values of p (e.g., $p = 3$, $p = 4$), the interpolant gains stronger "shoulders" near each datapoint; for $p \to \infty$, the interpolant takes the known function value at the nearest datapoint, and thus leads to a **piecewise constant** function over the **Voronoi cell** associated with each datapoint. For finite p in the limit that $R \to \infty$, the interpolation function is continuous; for reduced values of R, the interpolating function, though sometimes approximating the original function a bit more accurately, is discontinuous.

7.1.5.2 Polyharmonic spline interpolation

In many interpolation problems, such as that illustrated in Figure 7.3, the simple inverse distance formula for interpolation, discussed in §7.1.5.1, fails to give a sufficiently accurate result for any value of p. An effective alternative approach is given by the **polyharmonic spline**, a special case of which (in 2D and with $k = 2$), known as a **thin plate spline**, corresponds to the mechanical modeling of a 2D spline that is bent in order to make it touch the specified unstructured data points. This interpolation formula takes the form[2]

$$f(\mathbf{x}) = \sum_{i=1}^{N} w_i \phi(r) + \mathbf{v}^T \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} \qquad \text{where} \quad r = \|\mathbf{x} - \mathbf{c}^i\|_2, \tag{7.13a}$$

$$\phi(r) = \begin{cases} r^k & \text{for } k \text{ odd} \\ r^k \ln(r) & \text{for } k \text{ even,} \end{cases} \tag{7.13b}$$

and where the weights w_i and v_i are selected such that: (a) $f(\mathbf{c}_i) = f_i$ in the above equation for the N available data points $\{\mathbf{c}_i, f_i\}$, (b) the sum of the weights, $\sum_i w_i$, is zero, and (c) in each of the n coordinate directions, $j = 1, \ldots, n$, the weighted sum of the center locations, $\sum_i w_i c_{ji}$, is also zero. These three sets of conditions on the weights may be enforced by solving the $(N + 1 + n) \times (N + 1 + n)$ linear system

$$\begin{bmatrix} A & V^T \\ V & 0 \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{y} \\ 0 \end{bmatrix} \qquad \text{where} \quad A_{ij} = \phi(\|\mathbf{c}_j - \mathbf{c}^i\|_2), \quad V = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \mathbf{c}^1 & \mathbf{c}^2 & \cdots & c^N \end{bmatrix}. \tag{7.14}$$

Implementation is given in RR_PolyharmonicSplineSetup and RR_PolyharmonicSpline. Typical results are illustrated in Figure 7.4; note that increasing values of k are usually found to give a smoother interpolant on the interior of the portion of the domain covered by the data, but higher irregularities near the edges of this portion of the domain. Also, smaller values of k (e.g., 2 or 3) are often found to be more accurate when a relatively small number of function evaluations are available, with larger values of k (e.g., 6 to 8) being more accurate when more function evaluations are available.

It is worth noting that the **polyharmonic spline** basis functions $\phi(r)$ given in (7.13b), used in the interpolation formula given in (7.13a) and plotted in Figure 7.5, are special cases of what are commonly referred to as **radial basis functions** (**RBF**s), as they depend on the Euclidian distance $r = \|\mathbf{x} - \mathbf{c}^i\|_2$ of the new point $\mathbf{x}$ from the centers $\mathbf{c}_i$ only. RBFs come in two essential types: those which decay with radius, such as the **Gaussian RBF** $\phi(r) = e^{-(\epsilon r)^2}$ as well as the **inverse distance RBF** $\phi(r) = 1/r^p$ used in the interpolation strategy presented in §7.1.5.1, and those which eventually grow with radius, such as the **polyharmonic spline RBF** defined in (7.13b) and used in the interpolation strategy presented in §7.1.5.2; the former are essentially "local"

[2] In cases with k even, the equivalent formula $\phi(r) = r^{k-1} \ln(r^r)$ is better behaved numerically for $r < 1$. Various other possible formulae for $\phi(r)$, such as $\phi(r) = r^2$, are found to be ill-behaved in this setting, and are not considered further here.

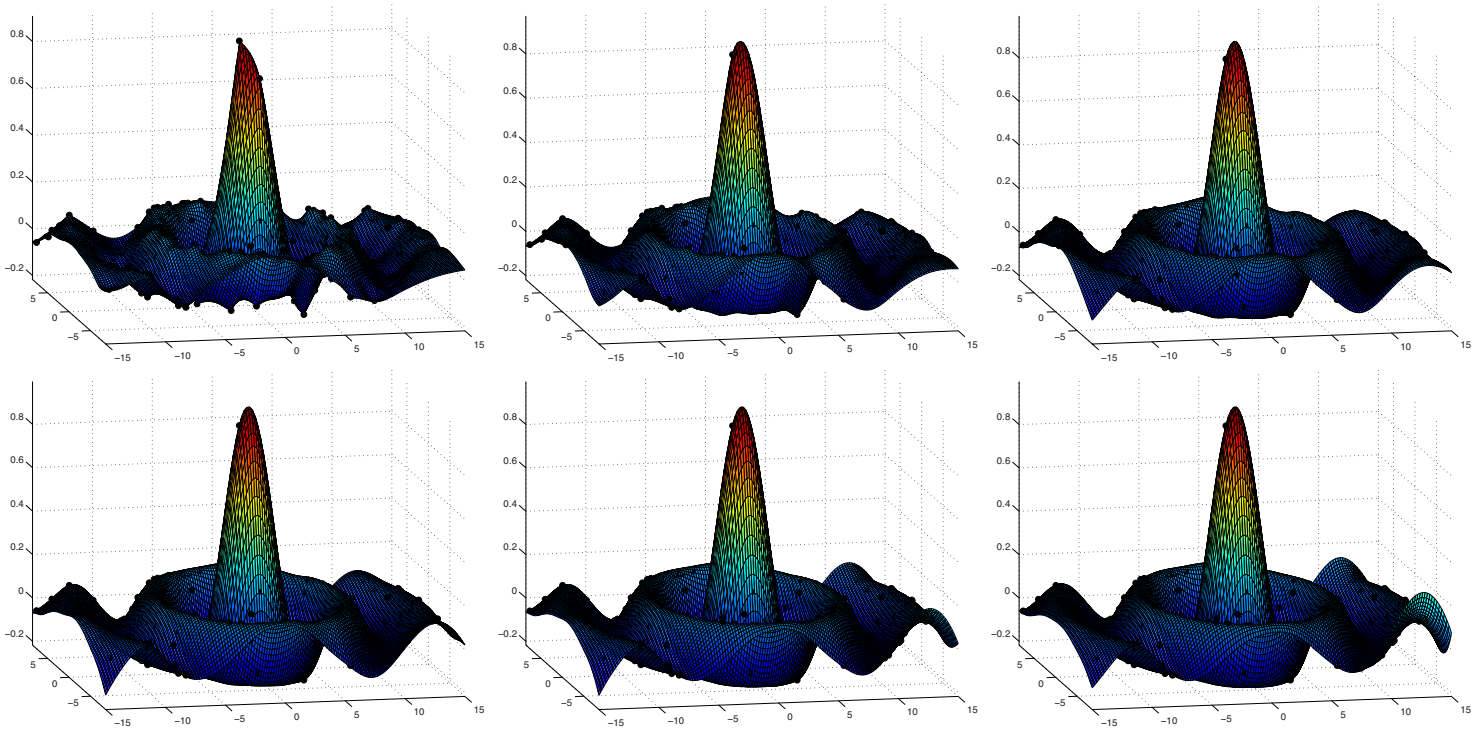


Figure 7.4: Polyharmonic spline interpolation of 200 points with (a) $k = 1$ through (f) $k = 6$. Over the domain illustrated, the maximum error is $\{0.176, 0.123, 0.119, 0.131, 0.186, 0.259\}$ and the rms error is $\{0.0376, 0.0220, 0.0157, 0.0130, 0.0142, 0.0171\}$, respectively, in the six cases considered.

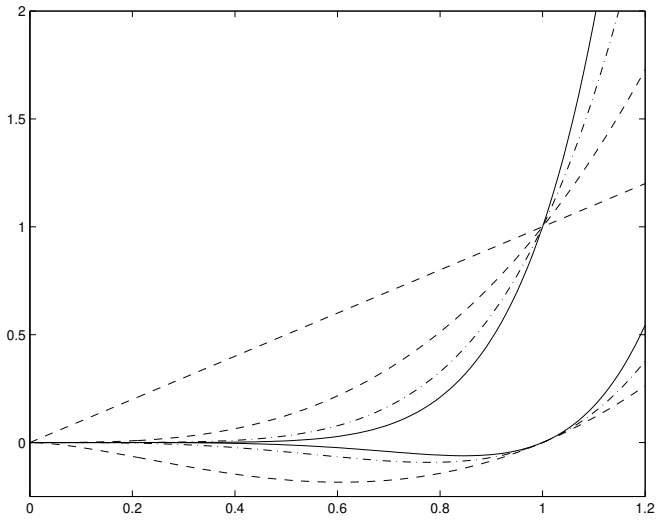


Figure 7.5: **Polyharmonic spline radial basis functions** [see (7.13b)] for (dashed) $k = 1$, $k = 2$, $k = 3$; (dot-dashed) $k = 4$, $k = 5$; (solid) $k = 6$, $k = 7$. Though they eventually grow with radius, these non-local RBFs are found to be effective in the polyharmonic spline interpolation strategy presented in §7.1.5.2.

in nature, whereas the latter are "global" in effect, and thus their weights must be determined via a solve over the entire set of datapoints [see (7.14)].

Chapter 8

Signals & Systems

Contents

In §8.1, we briefly introduce the three essential classes of transforms at the heart of signal analysis: Fourier, Laplace, and Z. The **Fourier transform**, which comes in four forms appropriate for either **continuous-time** (CT) signals or **discrete-time** (DT) signals {that is, defined only at regularly-spaced intervals over the time domain of interest}, and for signals defined on either **infinite domains** {that is[1], $t \in (-\infty, \infty)$} or **bounded domains** {that is, $t \in [0, T)$}, is built on *sinusoidal* basis functions, written as $e^{i\omega t} = \cos(\omega t) + i\sin(\omega t)$, as studied in depth in §5 of *NR*. The Laplace and Z transforms extend this tetralogy of Fourier transforms to **semi-infinite domains** {that is, $t \in [0, \infty)$}, for CT and DT signals respectively.

Before considering further the presentation of the Laplace and Z transforms in this chapter, and their extensive utility in control theory in §10, it is helpful to make this discussion more concrete by considering the ODEs modeling a handful of simple physical systems (a.k.a. "plants"), as reviewed briefly[2] in §6.1.

[1] As in §5 of *NR*, the physical coördinate over which such transforms may be applied may be interpreted as time or space, and is denoted without loss of generality in the present chapter as t; see also footnote 19 in §5.5 of *NR*.

[2] Note also related discussions of (a) the realization of various ODEs of interest as (CT) *electric circuits*, as considered in §9, particularly as low-pass, high-pass, or notch "filters" or as analog "controllers", and (b) the realization of various difference equations of interest on *microcontrollers*, as (DT) finite impulse response (FIR) or infinite impulse response (IIR) filters, as considered in §1.5.3.2.

The **Laplace transform**, appropriate for the analysis of **continuous-time** signals on **semi-infinite** domains $t \in [0, \infty)$, as well as for the analysis of **differential equations** governing their evolution from given initial conditions at $t = 0$, is built on *exponential* basis functions, e^{st}, and is considered in depth in §8.2.

The Z **transform**, appropriate for the analysis of **discrete-time** signals on **semi-infinite** domains $t_k \in \{0, h, 2h, \ldots\}$, as well as for the analysis of **difference equations** governing their evolution from given initial conditions around $t_0 = 0$, is built on *polynomial* basis functions, z^{k-1}, and is considered in depth in §8.3.

The use of Fourier transforms to analyze systems (specifically, using **Bode plots**) is examined in §8.4.

8.1 Introduction to transforms: Fourier, Laplace, and Z

Recall first the tetralogy of Fourier representations[3], which are defined for both *continuous* and *discrete* functions, on both *bounded* and *infinite* domains:

1. The forward and inverse **infinite Fourier series transform** [see §5.2 of *NR*], defined for *continuous signals* $u(t)$ on *bounded domains* $t \in [0, T)$ with $\omega_n = 2\pi n/T$ for $n \in \{\ldots, -2, -1, 0, 1, 2, \ldots\}$, are defined by

$$\hat{u}_n = \frac{1}{T}\int_0^T u(t) e^{-\mathrm{i}\omega_n t}\, dt \quad \Leftrightarrow \quad u(t) = \sum_{n=-\infty}^{\infty} \hat{u}_n e^{\mathrm{i}\omega_n t}. \tag{8.1a}$$

2. The forward and inverse **infinite Fourier integral transform** [see §5.3 of *NR*], defined for *continuous signals* $u(t)$ on *infinite domains* $t \in (-\infty, \infty)$ with $\omega \in (-\infty, \infty)$, are defined[4] by

$$\hat{u}(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} u(t) e^{-\mathrm{i}\omega t}\, dt \quad \Leftrightarrow \quad u(t) = \int_{-\infty}^{\infty} \hat{u}(\omega) e^{\mathrm{i}\omega t} d\omega. \tag{8.1b}$$

3. The forward and inverse **finite Fourier series transform** [see §5.4 of *NR*], defined for *discrete signals* $u_k = u(t_k)$ on *bounded domains* $t_k = kh$ for $k = \{0, \ldots, N-1\}$ and $h = T/N$ with[5] $\omega_n = 2\pi n/T$ for $n \in \{-N/2, \ldots, N/2\}$, are defined by

$$\hat{u}_n = \frac{1}{N}\sum_{k=0}^{N-1} u_k e^{-\mathrm{i}\omega_n t_k} \quad \Leftrightarrow \quad u_k = \sum_{n=-N/2}^{N/2} \hat{u}_n e^{\mathrm{i}\omega_n t_k}. \tag{8.1c}$$

4. The forward and inverse **finite Fourier integral transform** [Exercise 5.2 of *NR*], defined for *discrete signals* $u_k = u(t_k)$ on *infinite domains* $t_k = kh$ for $k = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ with $\omega \in (-\pi/h, \pi/h)$, are defined by

$$\hat{u}(\omega) = \frac{h}{2\pi}\sum_{k=-\infty}^{\infty} u_k e^{-\mathrm{i}\omega t_k} \quad \Leftrightarrow \quad u_j = \int_{-\pi/h}^{\pi/h} \hat{u}(\omega) e^{\mathrm{i}\omega t_j}\, d\omega. \tag{8.1d}$$

Similarly, the forward and inverse **Laplace transform** [developed in §8.2], defined for *continuous signals* $u(t)$ on *semi-infinite domains*, $t \in [0, \infty)$, are defined by

$$U(s) = \int_0^{\infty} u(t) e^{-st} dt \quad \Leftrightarrow \quad u(t) = \frac{1}{2\pi\mathrm{i}}\int_{\gamma-\mathrm{i}\infty}^{\gamma+\mathrm{i}\infty} U(s) e^{st} ds, \tag{8.2}$$

and the forward and inverse Z **transform** [developed in §8.3], defined for *discrete signals* $u_k = u(t_k)$ on *semi-infinite domains*, $t_k = kh$ for $k = \{0, 1, 2, \ldots\}$, are defined by

$$U(z) = \sum_{k=0}^{\infty} u_k z^{-k} \quad \Leftrightarrow \quad u_k = \frac{1}{2\pi\mathrm{i}}\oint_{\Gamma} U(z) z^{k-1}\, dz. \tag{8.3}$$

[3] Time permitting, it is instructive at this point to also examine various other aspects of Fourier representations; see §5 of *NR*.

[4] Different authors place the factor of $1/(2\pi)$ in the Fourier integral (8.1b) in different ways (see Footnote 10 in §5.4 of *NR*).

[5] Note in particular the discussion in §5.5 of *NR* of the peculiar component of this signal at the **Nyquist frequency** $\omega_{N/2} = \pi N/T$.

8.1.1 The relation of the Laplace and Z transforms to the Fourier transform

At the outset, note that the Laplace transform at right in (8.2) is simply a representation, or "expansion", of a continuous function $u(t)$ on $t \in [0, \infty)$ as a linear combination of a set of exponential basis functions $e^{s\,t}$ with the coefficient function $U(s)$ as weights. Similarly, the Z transform at right in (8.3) is simply a representation of a discrete function u_k on $k = 0, 1, 2, \ldots$ as a linear combination of a set of polynomial basis functions z^{k-1} with the coefficient function $U(z)$ as weights. The Laplace and Z transforms are thus remarkably similar to the corresponding Fourier transforms (8.1b) and (8.1d), respectively, which similarly represent continuous and discrete functions on infinite domains as a linear combination of a set of complex exponential basis functions with the Fourier coefficients as weights. Indeed, noting the definition of the Laplace transform in (8.2) and the infinite Fourier integral expansion in (8.1b), it follows (taking $s = \mathrm{i}\,\omega$) that

$$\left.\begin{aligned} U(s) &= \int_0^\infty u(t) e^{-s\,t} dt \\ \hat{u}(\omega) &= \frac{1}{2\pi} \int_{-\infty}^\infty u(t) e^{-\mathrm{i}\,\omega\,t}\, dt \end{aligned}\right\} \quad \Rightarrow \quad \hat{u}(\omega) = \frac{1}{2\pi} U(\mathrm{i}\,\omega) \quad \text{if} \quad u(t) = 0 \;\text{ for }\; t < 0. \tag{8.4}$$

Similarly, noting the definition of the Z transform in (8.3) and the finite Fourier integral expansion in (8.1d), it follows (taking $z = e^{sh}$ with $s = \mathrm{i}\,\omega$) that

$$\left.\begin{aligned} U(z) &= \sum_{k=0}^\infty u_k z^{-k} \\ \hat{u}(\omega) &= \frac{h}{2\pi} \sum_{k=-\infty}^\infty u_k e^{-\mathrm{i}\,\omega\,t_k} \end{aligned}\right\} \quad \Rightarrow \quad \hat{u}(\omega) = \frac{h}{2\pi} U(e^{\mathrm{i}\omega h}) \quad \text{if} \quad u_k = 0 \;\text{ for }\; k < 0. \tag{8.5}$$

8.1.2 The remarkable utility of such transforms

The utility of the Fourier transform in the identification and analysis of the various sinusoidal components of a signal at different temporal "wavenumbers" or spatial "frequencies" should already be well familiar to the reader. Indeed, any aspiring young audiophile is already familiar with the need to route the "low-frequency sinusoidal components" of an audio signal to a woofer, to route the "high-frequency sinusoidal components" of an audio signal to a tweeter, and to dampen the "highest-frequency sinusoidal components" of an audio signal associated with noise, which can come from a variety of sources; the Fourier transform simply makes this decomposition of a signal into sinusoidal components at different frequencies mathematically precise.

The Laplace and Z transforms are similarly natural for the analysis of the *evolution* of **continuous-time** (**CT**) systems and **discrete-time** (**DT**) systems from initial conditions, governed by differential equations and difference equations respectively. As such transform methods are centrally based on an *abstraction* (the temporal frequency ω or spatial wavenumber k in the case of the Fourier transforms, the exponential scaling s in the case of the Laplace transform, and the base of the polynomial expansion, z, in the case of the Z transform), they require a bit of analysis, as provided in §8.2 and §8.3, before their utility is fully apparent.

It should be noted at the outset that all of these transforms are **linear**: that is, if X and Y are the (Fourier, Laplace, or Z) transforms of x and y, then $W = \alpha X + \beta Y$ is the corresponding transform of $w = \alpha x + \beta y$. Further, all of these transforms are **invertible**: that is, knowledge of the untransformed variable x over the appropriate region of the physical domain is sufficient to reconstruct the transformed variable X over the abstracted domain, and knowledge of the transformed variable X over the appropriate region of the abstracted domain is sufficient to reconstruct the untransformed variable x over the physical domain. These two points are essential to the practical utility of analysis, filtering, and control techniques based on such transforms.

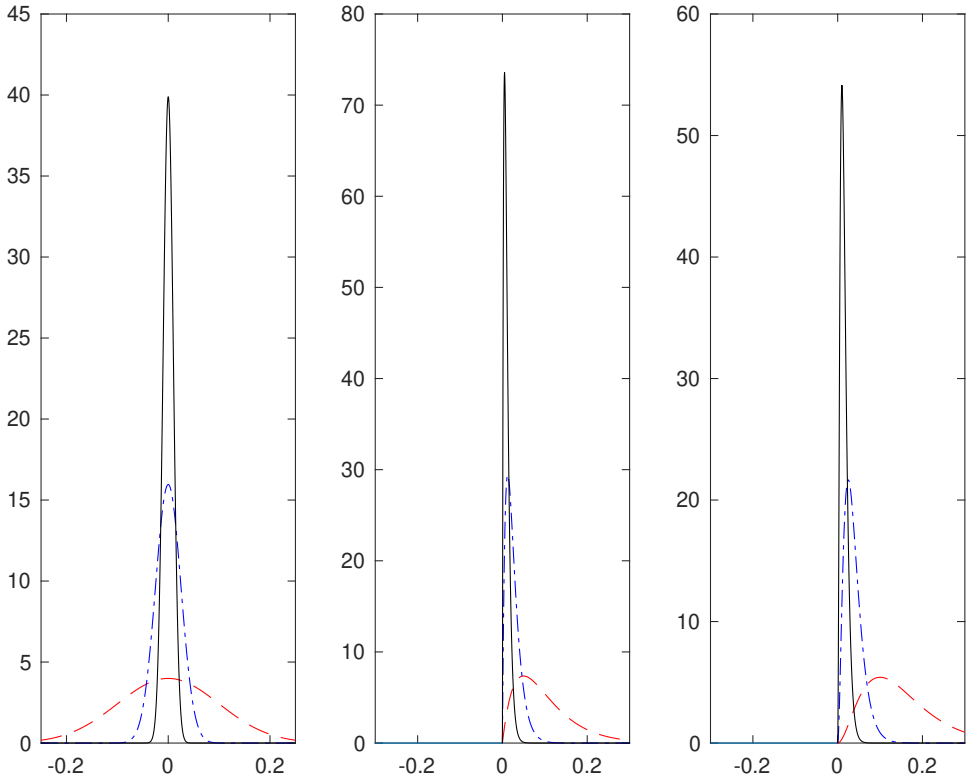


Figure 8.1: Three families of curves of unit area that are nonzero only in the immediate vicinity of the origin: (left) the two-sided function $\delta^{\sigma}(t) = e^{-t^2/(2\sigma^2)}/(\sigma\sqrt{2\pi})$ taking (dashed) $\sigma = 0.1$, (dot-dashed) $\sigma = 0.025$, and (solid) $\sigma = 0.01$, and (center, right) the one-sided function $\delta^{\lambda,m}(t) = \lambda^m\, t^{m-1}\, e^{-\lambda\, x/(m-1)!}$ for $t > 0$, with $m = 2$ and $m = 3$, respectively, taking (dashed) $\lambda = 20$, (dot-dashed) $\lambda = 80$, and (solid) $\lambda = 200$. As $\sigma \to 0$ in the definition of $\delta^{\sigma}(t)$, and as $\lambda \to \infty$ in the definition of $\delta^{\lambda,m}(t)$ for $m \geq 2$, these curves become increasingly accurate finite approximations of the Dirac Delta, as discussed at length in §5.3.3 and §5.3.4 of *NR*. Note that, for $m \geq 2$, $\delta^{\lambda,m}(t)$ and all its derivatives up to order $m-2$ are continuous at the origin.

8.2 Laplace transform methods

The (one-sided) **Laplace transform** $F(s)$ of a **continuous-time** (**CT**) signal $f(t)$ is, in general, defined as

$$F(s) = \lim_{\epsilon \to 0} \int_{-\epsilon}^{\infty} f(t) e^{-s\,t} dt \triangleq \int_{0^-}^{\infty} f(t) e^{-s\,t} dt. \tag{8.6}$$

In this text, we restrict all CT unit impulses to be constructed as the large λ limit of the *one*-sided function of unit area $\delta^{\lambda,m}(t)$, as defined and plotted in Figure 8.1b and c, rather than the small σ limit of the *two*-sided function of unit area $\delta^{\sigma}(t)$, as defined and plotted in Figure 8.1a. That is, we define the Dirac delta in this work as the limit of some smooth function of unit area that *begins* at the time of the impulse and is zero before it, rather than being *centered* at the time of the impulse, in the limit that the duration of the impulse is infinitesimally short. In this restricted[6] setting, the Laplace transform may be defined (see LePage 1961) more simply as

$$F(s) = \int_{0}^{\infty} f(t) e^{-s\,t} dt. \tag{8.7a}$$

[6]This "restriction" is said to be **technical**; that is, it narrows the precise mathematical setting in which the transform definition may be used, but in application does not limit the practical problems to which the transform may, when used correctly, be applied.

Given a function $f(t)$ for $t \geq 0$ restricted as stated above, we define $F(s)$ via (8.7a); the **inverse Laplace transform** is then

$$f(t) = \frac{1}{2\pi \mathrm{i}} \int_{\gamma-\mathrm{i}\infty}^{\gamma+\mathrm{i}\infty} F(s) e^{s\,t} ds, \tag{8.7b}$$

where the real number γ is chosen such that the vertical line of the contour in the complex plane s as given in the above integral is to the right of all of the singularities[7] of $F(s)$.

Verification that (8.7b) in fact represents the inverse of the relationship expressed in (8.7a) is straightforward, by substituting (8.7a) into the RHS of (8.7b), substituting $s = \gamma + \mathrm{i}k$, applying Fubini's theorem (see Footnote 2 in §5.2.1 of *NR*), and noting that, for sufficiently large γ, $f(t)$ is indeed recovered:

$$\begin{aligned}
\frac{1}{2\pi \mathrm{i}} \int_{\gamma-\mathrm{i}\infty}^{\gamma+\mathrm{i}\infty} \Big[\int_0^\infty f(t')\, e^{-st'} dt' \Big] e^{s\,t}\, ds &= \lim_{K\to\infty} \frac{1}{2\pi \mathrm{i}} \int_{-K}^{K} \Big[\int_0^\infty f(t')\, e^{\gamma(t-t')} e^{\mathrm{i}k(t-t')} dt' \Big] \mathrm{i}\, dk \\
&= \lim_{K\to\infty} \int_0^\infty f(t') e^{\gamma(t-t')} \Big[\int_{-K}^{K} \frac{1}{2\pi} e^{\mathrm{i}k(t-t')} dk \Big] dt' = \int_0^\infty [f(t')\, e^{\gamma(t-t')}]\, \delta(t-t')\, dt' = f(t),
\end{aligned}$$

where the definition of the Dirac delta given in §5.2.2 of *NR* has been applied in the second line. The reason that the $e^{\gamma(t-t')}$ factor, for sufficiently large positive γ, is required by this formula is to ensure that the term $g(t') = [f(t')\, e^{\gamma(t-t')}]$ decays to zero exponentially as $t' \to \infty$, which allows us to swap the order of the integrals using Fubini's theorem and obtain the result that $\int_0^\infty g(t')\, \delta(t-t')\, dt' = g(t)$.

As discussed further below, the forward and inverse transforms expressed by (8.7) are immensely useful when solving differential equations (in CT). By (8.7a), knowing $f(t)$ for $t \geq 0$, one can define $F(s)$ for any s. Conversely, by (8.7b), knowing $F(s)$ on an appropriate contour, one can determine $f(t)$ for $t \geq 0$. Before demonstrating further *why* such a transformation is useful, we first mention that, in practice, you do not actually need to compute the somewhat involved integrals given in (8.7) in order to use the Laplace transform effectively. Rather, leveraging the linearity of this transform and the process of partial fraction expansion, it is sufficient to reference a table listing some Laplace transform pairs in a few special cases, as shown in Table 8.1. Note also the following:

Fact 8.1 *The* **Laplace transform is linear***; that is,* **superposition holds***, and thus if the Laplace transforms of $x(t)$ and $y(t)$ are $X(s)$ and $Y(s)$, then the Laplace transform of $w(t) = \alpha x(t) + \beta y(t)$ is $W(s) = \alpha X(s) + \beta Y(s)$.*

Fact 8.2 *If the Laplace transform of $f(t)$ is $F(s)$, then the Laplace transform of the* **exponentially scaled** *function $g(t) = e^{-at} f(t)$ is $G(s) = \int_0^\infty f(t) e^{-(s+a)t} dt = F(s+a)$, and the Laplace transform of the* **delay** *function $g(t) = f(t-d)$ is $G(s) = \int_0^\infty f(t-d) e^{-s\,t} dt = \int_{-d^-}^\infty f(t) e^{-s(t+d)} dt = e^{-ds} F(s)$.*

Note in Table 8.1 that the Laplace transform of the **delay function**, $f(t) = \delta^{\lambda,m}(t-d)$ for $d > 0$ and $m \geq 2$ in the limit of large λ, is $F(s) = e^{-ds}$. This is not a rational function[8] of s, which is inconvenient. The following **Padé approximation** of $F(s) = e^{-ds}$, valid for small values of $|ds|$, is thus convenient to use in its place

$$e^{-ds} \approx F_{m,n}(s) \triangleq \frac{\sum_{k=0}^m a_k (-ds)^k}{\sum_{k=0}^n b_k (ds)^k}, \quad a_k = \frac{(m+n-k)!\, m!}{(m+n)!\, k!\, (m-k)!}, \quad b_k = \frac{(m+n-k)!\, n!}{(m+n)!\, k!\, (n-k)!}. \tag{8.8a}$$

[7] That is, the contour of integration in (8.7b) is chosen to the right of all points $\overline{s}$ for which $|F(s)| \to \infty$ as $s \to \overline{s}$ in (8.7a).

[8] A **rational function** of s is a polynomial in s divided by another polynomial in s. Both polynomials are often normalized such that the leading coefficient of the polynomial in the denominator is **monic** (that is, has a leading coefficient of one). In addition, the polynomial in the numerator of a rational function is sometimes normalized to be monic, with its leading coefficient factored out (this coefficient is called the **gain**, and is often denoted K). Further, the numerator and denominator are often factored; the roots of the polynomial in the denominator are called **poles**, and the roots of the polynomial in the numerator are called **zeros**.

$f(t)$ (for $t \geq 0$)	$F(s)$
e^{at}	$1/(s-a)$
$t\,e^{at}$	$1/(s-a)^2$
$t^2\,e^{at}$	$2/(s-a)^3$
$t^p\,e^{at}$ (for integer $p \geq 0$)	$p!/(s-a)^{p+1}$
1 [i.e., $f(t) = H_0(t)$]	$1/s$
t	$1/s^2$
t^2	$2/s^3$
t^p (for integer $p \geq 0$)	$p!/s^{p+1}$
$\delta^{\lambda,m}(t)$ (for $m \geq 2$)	$\xrightarrow[\lambda\to\infty]{} 1$
$\delta^{\lambda,m}(t-d)$ (for $d \geq 0$)	$\xrightarrow[\lambda\to\infty]{} e^{-ds}$
$d[\delta^{\lambda,m}(t)]/dt \triangleq \delta'(t)$	$\xrightarrow[\lambda\to\infty]{} s$
$d^2[\delta^{\lambda,m}(t)]/dt^2 \triangleq \delta''(t)$	$\xrightarrow[\lambda\to\infty]{} s^2$
$\cos(\omega_d t)$	$s/(s^2+\omega_d^2)$
$\sin(\omega_d t)$	$\omega_d/(s^2+\omega_d^2)$
$e^{-\sigma t}\cos(\omega_d t)$	$\dfrac{s+\sigma}{(s+\sigma)^2+\omega_d^2}$
$e^{-\sigma t}\sin(\omega_d t)$	$\dfrac{\omega_d}{(s+\sigma)^2+\omega_d^2}$
$\cosh(at)$	$s/(s^2-a^2)$
$\sinh(at)$	$a/(s^2-a^2)$

f_k (for $k = 0, 1, \ldots$)	$F(z)$
r^k	$z/(z-r)$
$k\,r^k$	$r\,z/(z-r)^2$
$k^2 r^k$	$r\,z(z+r)/(z-r)^3$
$k^3 r^k$	$\dfrac{rz(z^2+4\,r\,z+r^2)}{(z-r)^4}$
$k^p r^k$ (for integer $p > 0$)	$Li_{-p}(r/z)$
1 [e.g., $f_k = H_{0k}$]	$z/(z-1)$
k	$z/(z-1)^2$
k^2	$z(z+1)/(z-1)^3$
k^p (for integer $p > 0$)	$Li_{-p}(1/z)$
δ_{0k}	1
δ_{dk} (for integer $d > 0$)	$1/z^d$
$r^k\cos(\theta k)$	$\dfrac{z[z - r\,\cos(\theta)]}{z^2 - 2\,r\,z\,\cos(\theta) + r^2}$
$r^k\sin(\theta k)$	$\dfrac{z\,r\,\sin(\theta)}{z^2 - 2\,r\,z\,\cos(\theta) + r^2}$
$r^k\,H_{1k}$	$r/(z-r)$
$(k-1)\,r^k\,H_{2k}$	$r^2/(z-r)^2$
$(k-2)(k-1)\,r^k\,H_{3k}$	$2\,r^3/(z-r)^3$
$(k-3)(k-2)(k-1)\,r^k\,H_{4k}$	$6\,r^4/(z-r)^4$
$(k-p)\cdots(k-1)\,r^k\,H_{p+1,k}$	$p!\,r^{p+1}/(z-r)^{p+1}$

Table 8.1: Tables of (left) some Laplace transform pairs, as considered in §8.2, and (right) some Z transform pairs, as considered in §8.3. The values of the CT fns $f(t)$ for $t < 0$, and the values of the DT fns f_k for $k < 0$, do not affect the above calculations. Note that:

- The **CT unit impulse** (aka the **Dirac delta**) in this work is defined via the large λ limit of the *one*-sided function $\delta^{\lambda,m}(t)$ for some integer $m \geq 2$ (see Figures 8.1b and c), and is thus taken to *begin* at $t = 0$.
- The **CT unit step** (aka the **CT Heaviside step fn**) is defined as $H_0(t) = 0$ for $t \leq 0$ and $H_0(t) = 1$ for $t > 0$.
- The **DT unit impulse** is defined via the **Kronecker delta** such that $\delta_{dk} = 1$ for $k = d$ and $\delta_{dk} = 0$ for $k \neq d$.
- The **DT unit step** (aka the **DT Heaviside step fn**) is defined as $H_{dk} = 0$ for $k < d$ and $H_{dk} = 1$ for $k \geq d$.
- Also, the **polylogarithm** $Li_{-p}(x)$ is defined such that $Li_{-p}(x) = (x\,d/dx)^p[x/(1-x)]$.

Such rational approximations of e^{-ds} have m RHP zeros and n LHP poles. Examples include

$$F_{2,2}(s) = \frac{s^2 - 6s/d + 12/d^2}{s^2 + 6s/d + 12/d^2} = \frac{(s+p_+)(s+p_-)}{(s-p_+)(s-p_-)} \quad \text{where} \quad p_\pm = (-3 \pm \sqrt{3}\,\mathrm{i})/d, \tag{8.8b}$$

$$F_{3,4} = \frac{-4s^3/d + 60s^2/d^2 - 360s/d^3 + 840/d^4}{s^4 + 16s^3/d + 120s^2/d^2 + 480s/d^3 + 840/d^4}, \quad F_{4,4} = \frac{s^4 - 20s^3/d + 180s^2/d^2 - 840s/d^3 + 1680/d^4}{s^4 + 20s^3/d + 180s^2/d^2 + 840s/d^3 + 1680/d^4};$$

RR_pade.m generates other approximations, and plots the poles & zeros, and the step & ramp response, of each. One often takes $m = n$, but note that taking m slightly less than n generates a smoother step response.

8.2.1 The Laplace Transform of derivatives and integrals of functions

Assume $f(t)$ is smooth and bounded and define $f^{(1)}(t) = df(t)/dt = f'(t)$. Then, by integration by parts [i.e., $\int_a^b u\,dv = (uv)_a^b - \int_a^b v\,du$, taking $u = e^{-st}$ and $dv = f^{(1)}(t)\,dt$], the Laplace transform of $f^{(1)}(t)$ is given by

$$\begin{aligned} F^{(1)}(s) &= \int_0^\infty f^{(1)}(t)\,e^{-s\,t}dt = \lim_{b\to\infty}\int_0^b f^{(1)}(t)e^{-s\,t}dt \\ &= \lim_{b\to\infty}\left[e^{-sb}f(a) - f(0) + s\int_0^b e^{-s\,t}f(t)dt\right] = sF(s) - f(0) \end{aligned} \tag{8.9a}$$

for $\Re(s) > 0$. Similarly, if $f^{(2)}(t) = d^2f(t)/dt^2 = f''(t)$ and $f^{(n)}(t) = d^nf(t)/dt^n$, then

$$F^{(2)}(s) = \int_0^\infty f^{(2)}(t)\,e^{-s\,t}dt = \ldots = s^2F(s) - sf(0) - f^{(1)}(0), \tag{8.9b}$$

$$F^{(n)}(s) = \int_0^\infty f^{(n)}(t)\,e^{-s\,t}dt = \ldots = s^nF(s) - s^{n-1}f(0) - s^{n-2}f^{(1)}(0) - \ldots - f^{(n-1)}(0). \tag{8.9c}$$

Thus, if $f^{(1)}(t) = df(t)/dt$, then $F^{(1)}(s) = sF(s) - f(0)$. Conversely, by integration, it therefore follows that, if $f(t) = \int_0^t f^{(1)}(t')dt'$, and thus $f(0) = 0$, then $F(s) = \frac{1}{s}F^{(1)}(s)$. We thus arrive at the most useful interpretation of the s variable:

Fact 8.3 *Multiplication of the Laplace transform of a CT signal by s corresponds to differentiation of this signal in the time domain, multiplication times s^2 corresponds to twice differentiation, etc. Conversely, multiplication by $1/s$ corresponds to integration of this signal (from $t = 0$), multiplication by $1/s^2$ corresponds to double integration, etc.*

Note that, with $f^{(1)}(t) = df(t)/dt$,

$$\lim_{s\to0}\left[F^{(1)}(s)\right] = \lim_{s\to0}\left[\int_0^\infty f^{(1)}(t)e^{-s\,t}dt\right] = \int_0^\infty \lim_{s\to0}\left[f^{(1)}(t)e^{-s\,t}\right]dt = \int_0^\infty f^{(1)}(t)dt = f(\infty) - f(0).$$

Combining this result with that achieved by taking the limit of (8.9a) as $s \to 0$, it follows that

Fact 8.4 (The CT final value theorem) $\lim_{s\to0} sF(s) = \lim_{t\to\infty} f(t)$.

If we now consider the limit as $s \to \infty$ instead of $s \to 0$, we have to be a bit more careful. In the case in which $f(t)$ is a scalar $c = \lim_{\epsilon\to0} f(\epsilon) - f(0)$ times a (left-continuous) unit step plus other terms which are continuous near the origin, we define $f^{(1)}(t)$ (kept under the integral sign; see Fact 5.6 of *NR*) as the scalar c times the Dirac delta[9] plus other terms which are bounded near the origin. From the sifting property of the Dirac delta [see (5.21c) of *NR*], it follows by taking the limit of (8.9a) as $s \to \infty$ that $c = \lim_{s\to\infty} sF(s) - f(0)$, and thus

Fact 8.5 (The CT initial value theorem) $\lim_{s\to\infty} sF(s) = \lim_{t\to0^+} f(t)$.

[9] In this case, the Dirac delta $\delta(t)$ may be defined via the effect of a one-sided impulse function of unit area, specifically $\delta^{\lambda,m}(t)$ for $m \geq 2$ in the limit of large λ (see Figure 8.1b-c), when kept under the integral sign and integrated against some test function, as discussed in detail in §5.3.4 of *NR*.

8.2.2 Using the Laplace Transform to solve unforced linear differential equations

Consider the unforced linear constant-coefficient second-order differential equation given by

$$f''(t) + a_1 f'(t) + a_0 f(t) = 0 \quad \text{with} \quad f(0),\ f'(0) \text{ given.} \tag{8.10}$$

Taking the Laplace transform of this equation and applying the above relations gives

$$\int_0^\infty \{f''(t) + a_1 f'(t) + a_0 f(t) = 0\} e^{-st} dt \quad \Rightarrow \quad [s^2 F(s) - s f(0) - f'(0)] + a_1[sF(s) - f(0)] + a_0[F(s)] = 0$$

$$\Rightarrow \quad F(s) = \frac{c_1 s + c_0}{s^2 + a_1 s + a_0} \quad \text{where } \ c_1 = f(0),\ \ c_0 = f'(0) + a_1 f(0).$$

Defining the roots of the denominator $p_\pm = \big(-a_1 \pm \sqrt{a_1^2 - 4a_0}\big)/2$, known as the **poles** of this second-order equation, and performing a **partial fraction expansion**[10], it follows that

$$F(s) = \frac{c_1 s + c_0}{(s - p_+)(s - p_-)} = \frac{d_+}{s - p_+} + \frac{d_-}{s - p_-} \quad \Rightarrow \quad \left\{ \begin{matrix} d_+ + d_- = c_1 \\ -d_+ p_- - d_- p_+ = c_0 \end{matrix} \right\} \quad \Rightarrow \quad \left\{ \begin{matrix} d_+ = \dfrac{c_1 p_+ + c_0}{p_+ - p_-}, \\ d_- = \dfrac{c_1 p_- + c_0}{p_- - p_+}. \end{matrix} \right.$$

Thus, by Table 8.1a and the linearity of the Laplace transform (Fact 8.1, from which the superposition principle follows immediately), we deduce that

$$f(t) = d_+ e^{p_+ t} + d_- e^{p_- t}, \tag{8.11}$$

thus solving the original differential equation (8.10). It is seen that, if the real parts of the poles $p_\pm$ are negative, the magnitude of the solution decays with time, whereas if the real parts of $p_\pm$ are positive, the magnitude of the solution grows with time. Also note that, if the coefficients $\{a_0, a_1\}$ and initial conditions $\{f(0), f'(0)\}$ defining the system in (8.10) are real, then the roots $p_\pm$ are either real or a complex conjugate pair, and thus the solution $f(t)$ given by (8.11) is real even though the roots $p_\pm$ might be complex.

Higher-order unforced constant-coefficient CT linear differential equations of the form

$$f^{(n)}(t) + a_{n-1} f^{(n-1)}(t) + \ldots + a_1 f'(t) + a_0 f(t) = 0,$$

may be solved analogously, again leveraging partial fraction expansions (see RR_partial_fraction_expansion) to split up $F(s)$ into simple terms whose inverse Laplace transforms may be found in Table 8.1a. In such cases, as in the second-order case discussed above, the speed of oscillation and the rate of decay or growth of the various components of the solution are characterized solely by the poles [that is, in this case, the roots of $s^n + a_{n-1} s^{n-1} + \ldots + a_1 s + a_0 = 0$], whereas *how much* of each of these components this solution actually contains, in addition to their relative phase, is a function of the initial conditions on $f(t)$ and its derivatives.

[10] The pedestrian way of computing the coefficients of a partial fraction expansion is to multiply the RHS factors [in this case, $d_+/(s - p_+)$ and $d_-/(s - p_-)$] by simple rational expressions of s, equivalent to unity, in order to form a **common denominator** with the LHS, then setting like powers of s in the numerator on the LHS and RHS as equal and solving the resulting set of linear equations, as indicated here in curly brackets. A much more direct way to compute the same coefficients is the Heaviside Cover-up Method in which, for rational expressions $F(s)$ with no repeated roots in the denominator, you simply multiply $F(s)$ by one of the factors in its denominator [in this case, say, $(s - p_+)$], cancel this term everywhere it now appears in both the numerator and the denominator, and then evaluate what is left as s approaches this root [in this case, as $s \to p_+$]. In the present case, this gives:

$$\lim_{s \to p_+} \left[(s - p_+) \left\{ \frac{c_1 s + c_0}{(s - p_+)(s - p_-)} = \frac{d_+}{s - p_+} + \frac{d_-}{s - p_-} \right\} \right] \quad \Rightarrow \quad \lim_{s \to p_+} \left[\frac{\cancel{(s - p_+)}(c_1 s + c_0)}{\cancel{(s - p_+)}(s - p_-)} = \frac{d_+ \cancel{(s - p_+)}}{\cancel{(s - p_+)}} + \frac{d_-(s - p_+)}{s - p_-} \right].$$

Since the last term on the RHS goes to zero in the $s \to p_+$ limit, this gives $d_+ = (c_1 p_+ + c_0)/(p_+ - p_-)$, consistent with the answer found using the method of simultaneous equations; d_- may be computed similarly, and the method extends immediately to higher-order rational functions with distinct poles. Generalization to rational function with repeated poles is given in §B of *NR*.

8.2.3 Continuous-time (CT) transfer functions

Now consider the forced, CT, **linear time invariant** (**LTI**; that is, constant-coefficient), **single input, single output** (**SISO**) second-order ODE for $y(t)$ (the **output**) given by

$$y''(t) + a_1 y'(t) + a_0 y(t) = b_0 u(t), \tag{8.12}$$

where $u(t)$ (the **input**) is specified, assuming $y(t)$ and $y'(t)$ are zero at $t = 0$. Taking the Laplace transform now gives

$$\int_0^\infty \{y''(t) + a_1 y'(t) + a_0 y(t) = b_0 u(t)\} e^{-s\,t} dt \quad \Rightarrow \quad [s^2 + a_1 s + a_0] Y(s) = b_0 U(s)$$
$$\Rightarrow \quad G(s) \triangleq \frac{Y(s)}{U(s)} = \frac{b_0}{s^2 + a_1 s + a_0} = \frac{b_0}{(s - p_+)(s - p_-)}, \tag{8.13}$$

where, again, the poles $p_\pm = \big(-a_1 \pm \sqrt{a_1^2 - 4a_0}\big)/2$. The quantity $G(s)$ given above is known as the **transfer function** of the linear system (8.12).

Higher-order forced SISO constant-coefficient CT linear systems of the form

$$y^{(n)}(t) + a_{n-1} y^{(n-1)}(t) + \ldots + a_1 y'(t) + a_0 y(t) = b_m u^{(m)}(t) + b_{m-1} u^{(m-1)}(t) + \ldots + b_1 u'(t) + b_0 u(t), \tag{8.14}$$

with $b_m \neq 0$ [and, normally, $m \leq n$; see §8.2.3.1], may be manipulated in an analogous manner, leading to a transfer function of the rational form

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \ldots + b_1 s + b_0}{s^n + a_{n-1} s^{n-1} + \ldots + a_1 s + a_0} = K \frac{(s - z_1)(s - z_2) \cdots (s - z_m)}{(s - p_1)(s - p_2) \cdots (s - p_n)}. \tag{8.15}$$

The m roots of the numerator, z_i, are referred to as the **zeros** of the system, the n roots of the denomenator, p_i, are referred to as the **poles** of the system, and the coefficient K is referred to as the **gain** of the system.

Note that a differential equation governing a CT system with mechanical and/or electrical components, taken on its own, simply relates, at a single instant in time, linear combinations of two or more variables and their derivatives. Such a differential equation does *not* itself indicate one variable as a "cause" and another as an "effect" in a cause-effect relationship; the indication of that is left to the engineer. The statement of a transfer function, however, inherently defines a **cause-effect** or **input-output** relationship; in the examples discussed above, by putting $U(s)$ in the denominator and $Y(s)$ in the numerator, $u(t)$ is identified as the "input" (the prescribed quantity put into the system), and $y(t)$ is identified as the "output" (a resulting quantity generated by action of the system). This distinction between input and output, imposed by the engineer, is significant.

Further, as a differential equation only relates variables and their derivatives in a particular CT system at a single instant in time, there is no notion of "causality" associated with a differential equation[11]. However, differential equations that are actually implementable as physical systems or electric circuits generally have higher-order derivatives on the output than they do on the input, as discussed further in §8.2.3.1.

Given a CT linear system's transfer function $G(s)$, its response to simple inputs is easy to compute. Noting Table 8.1a:

- if $u(t)$ is a **unit impulse** [$u(t) = \delta^{\lambda,m}(t)$ for large λ and integer $m \geq 1$], then $U(s) \approx 1$;
- if $u(t)$ is a **unit step** [$u(t) = H_0(t)$; then $U(s) = 1/s$;
- if $u(t)$ is a **unit ramp** [$u(t) = t$ for $t > 0$], then $U(s) = 1/s^2$, etc.

[11] This is in contrast to difference equations relating DT signals at different timesteps, as discussed in §8.3.3; see in particular §8.3.3.2.

In such cases, $Y(s) = G(s)U(s)$ is easy to compute, and thus $y(t)$ may be found by partial fraction expansion (again, see RR_partial_fraction_expansion for automation of this process) and subsequent inverse Laplace transform. As in the unforced case discussed in §8.2.2, the speed of oscillation and the rate of decay or growth of the various components of the system's response to a simple input is characterized solely by the poles of the system, whereas *how much* of each of these components this response actually contains, in addition to their relative phase, is a function of its zeros and gain.

It is important to keep clear the distinction between the Laplace transform (a.k.a. transfer function) of a *system*, such as $G(s)$ above, and the Laplace transform of a *signal*, such as $Y(s)$ above. To make clear the connection between them, note in the special case that the input to the system happens to be a unit impulse $u(t) = \delta^{\lambda,m}(t)$ for large λ and integer $m \geq 2$, it follows that $U(s) \approx 1$ and thus $Y(s) \approx G(s)$. In other words,

Fact 8.6 *The transfer function of a CT linear system is the Laplace transform of its impulse response.*

It follows from the relation $Y(s) = G(s)U(s)$, expanding $Y(s)$, $G(s)$, and $U(s)$ with the Laplace transform (8.7a), noting that the impulse response $g(t) = 0$ for $t < 0$ (that is, that the system is causal, as discussed above), that

$$\int_0^\infty \Big[y(t)\Big] e^{-st} dt = \int_0^\infty g(t) e^{-st} dt \int_0^\infty u(t') e^{-st'} dt' = \int_0^\infty u(t') \left(\int_{-(t')}^\infty g(t) e^{-st} dt \right) e^{-st'} dt'$$

$$= \int_0^\infty u(t') \left(\int_0^\infty g(t-t') e^{-s(t-t')} dt \right) e^{-st'} dt' = \int_0^\infty \left[\int_0^t u(t') g(t-t')\, dt' \right] e^{-st} dt,$$

from which we deduce that, for $t \geq 0$,

$$y(t) = \int_0^t u(t') g(t-t') dt'; \tag{8.16}$$

note in particular that $y(t) \approx g(t)$ when $u(t) = \delta^{\lambda,m}(t)$ for large λ. Thus, as similarly noted for the Fourier transform in Fact 5.4 of *NR*,

Fact 8.7 *The product $Y(s) = G(s)U(s)$ in Laplace transform space corresponds to a convolution integral [of the input $u(t)$ with the impulse response $g(t)$] in the untransformed space.*

Products are generally much easier to work with than convolution integrals, thus highlighting the utility of the Laplace transform when solving constant-coefficient CT linear systems.

8.2.3.1 Proper, strictly proper, and improper CT systems

We now revisit the differential equation in (8.14) and its corresponding transfer function in (8.15), where the degree of the polynomial in the numerator is m, and the degree of the polynomial in the denominator is n. Define the **relative degree** of such a transfer function as $n_r = n - m$. In CT, such systems are said to be **improper** if $n_r < 0$. In §10 we will further distinguish the CT systems of interest as "plants" $G(s)$ and "controllers" $D(s)$. Most real plants $G(s)$ are **strictly proper**, with $n_r > 0$ [or at least **proper**, with $n_r \geq 0$], as most plants have some sort of **inertia**, **capacitance**, or **storage** which attenuates [or at least bounds] their response at high frequencies, as characterized precisely by their Bode plots (see §8.4). Further, to avoid amplifying high-frequency measurement noise which might be present as the measured signal is fed back to the actuator via control feedback, it is strongly advised to use a strictly proper [or, at least, proper], controller $D(s)$. Thus, we will focus our attention in this study almost exclusively on the case with $n_r \geq 0$. Note also that a transfer function with $n_r = 0$, which is proper but not strictly proper, is occasionally said to be **semi-proper**.

Example 8.1 The step response of second-order CT linear systems

We now focus further on the forced second-order case (8.12) when forced by a (CT) unit step $u(t) = H_0(t)$:

$$G(s) = \frac{b_0}{s^2 + a_1\, s + a_0} = \frac{b_0}{s^2 + 2\,\zeta\,\omega_n\, s + \omega_n^2} = \frac{b_0}{(s - p_+)(s - p_-)} \quad \text{and} \quad U(s) = \frac{1}{s}.$$

note that ω_n is called the **undamped natural frequency** or **speed** of the system, and ζ is called the **damping ratio**. If the poles $p_\pm = \left(-a_1 \pm \sqrt{a_1^2 - 4a_0}\right)/2$ are complex with negative real part, the solution may be written in terms of sines and cosines modulated by a decaying exponential, as implied by (8.11). To illustrate this more clearly, assume first that $a_0 > 0$ and $0 \le a_1 < 2\sqrt{a_0}$; it follows that

$$a_0 = \omega_n^2 \quad \text{and} \quad a_1 = 2\zeta\omega_n,$$

noting that $\omega_n > 0$ and $0 \le \zeta < 1$, it is seen (see Figure 8.2) that $p_\pm = -\sigma \pm \mathrm{i}\,\omega_d = -R\, e^{\pm \mathrm{i}\,\theta}$ where

$$\theta = \operatorname{asin}\zeta, \quad \sigma = \zeta\omega_n = a_1/2 \quad \text{and} \quad \omega_d = \omega_n\sqrt{1-\zeta^2} = \sqrt{a_0 - a_1^2/4} \quad \Rightarrow \quad \sigma^2 + \omega_d^2 = \omega_n^2,$$

in which case, via partial fraction expansion, we may write

$$Y(s) = G(s)U(s) = \frac{b_0}{(s-p_+)(s-p_-)} \cdot \frac{1}{s} = \frac{d_+}{s-p_+} + \frac{d_-}{s-p_-} + \frac{d_0}{s}, \quad \begin{cases} d_+ = -\mathrm{i}\, b_0/(2\,\omega_d\, p_+), \\ d_- = \mathrm{i}\, b_0/(2\,\omega_d\, p_-) = \overline{d_+}, \\ d_0 = b_0/\omega_n^2. \end{cases}$$

Thus, by Table 8.1a, noting that $y(t) = 0$ for $t \le 0$, the closed-form solution of $y(t)$ for $t > 0$ is given by

$$y(t) = d_+ e^{p_+ t} + d_- e^{p_- t} + d_0 = e^{-\sigma t}\big[d_c\cos(\omega_d t) + d_s \sin(\omega_d t)\big] + d_0, \quad \begin{cases} d_c = d_+ + d_- = -b_0/\omega_n^2, \\ d_s = \mathrm{i}(d_+ - d_-) = d_c\,\zeta/\sqrt{1-\zeta^2}, \end{cases}$$

as plotted in Figure 8.3 using the corresponding code in RR.ch08. Since the $G(s)$ in this example is real, the complex poles $\{p_+, p_-\}$ are a conjugate pair; as $u(t)$ in this example is also real, the coefficients $\{d_+, d_-\}$ are also a conjugate pair, and thus $\{d_c, d_s, d_0\}$, and $y(t)$ itself, are real. Note that the speed of oscillation ω_d and the rate of decay σ are simple functions of the location of the poles of $G(s)$.

As indicated in Figure 8.3, three useful characterizations of the step response are the **rise time** t_r, defined as the time it takes the response to increase from 0.1 to 0.9 of the steady state value of the step response, the **settling time** t_s, defined as the total time it takes the response to settle to within ± 5 percent of the steady state value of the step response, and the **overshoot** M_p, defined as the maximum percentage by which the output of the system exceeds its steady-state value when the system responds to a step input. By performing least-squares fits of the rise time, settling time, and overshoot of several such step responses of second-order systems as a function of ω_n, σ, and ζ, the following approximate relations are readily determined:

$$t_r \approx 1.8/\omega_n, \quad t_s \approx 4.6/\sigma, \quad M_p \approx e^{-\pi\zeta/\sqrt{1-\zeta^2}}.$$

If the maximum values of t_r, t_s, and/or M_p are specified, the following **approximate design guides** for the admissible pole locations of a second-order system follow:

$$\omega_n \gtrsim 1.8/t_r, \quad \sigma \gtrsim 4.6/t_s, \quad \begin{cases} \zeta \gtrsim 0.5 & \text{for} \quad M_p \le 15\% \\ \zeta \gtrsim 0.7 & \text{for} \quad M_p \le 5\%. \end{cases} \tag{8.17}$$

Typical approximate design guides of this sort are illustrated graphically in Figure 8.4. The response of many higher-order systems is dominated by the response due to a pair of **dominant second-order poles** [i.e., the slowest (smallest ω_n) poles of the system that are not approximately cancelled by nearby zeros]. Thus, these approximate design guides are often handy even if the system is not second order.

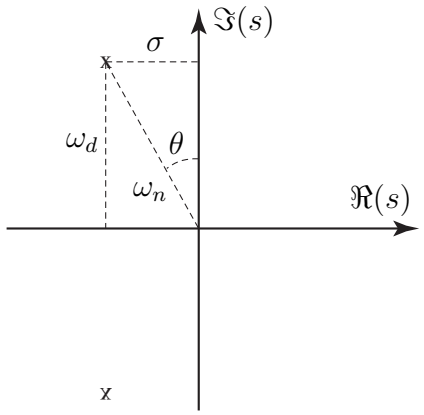


Figure 8.2: The poles $p_\pm$ of the system $y''(t) + 2\,\zeta\,\omega_n\,y'(t) + \omega_n^2\,y(t) = b_0\,u(t)$ in the complex plane s in terms of ω_n, $\theta = \operatorname{asin}\zeta$, $\sigma = \zeta\omega_n$, and $\omega_d = \omega_n\sqrt{1-\zeta^2}$. The response $y(t)$ to a step input $u(t)$ is plotted in Figure 8.3; note that ω_d sets the speed of oscillation and σ sets the exponential rate of decay.

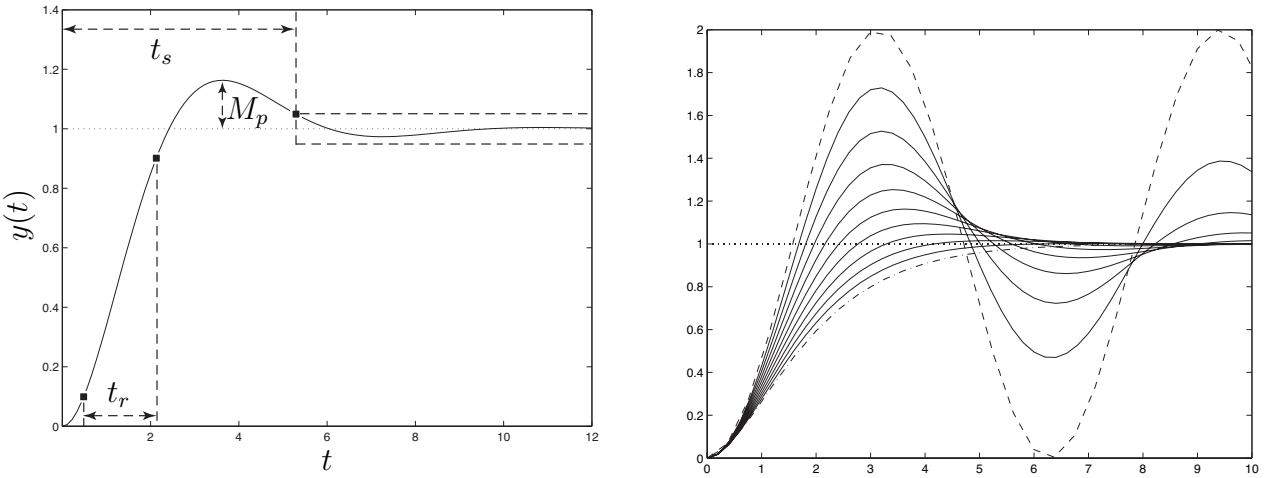


Figure 8.3: The unit step response of the system $y''(t) + 2\,\zeta\,\omega_n\,y'(t) + \omega_n^2\,y(t) = b_0\,u(t)$, for $\omega_n = 1$, $b = 1$ and (left) taking $\zeta = 0.5$, with the rise time, settling time, and overshoot indicated, and (right) taking $\zeta = 0$ (dashed), $\zeta = 0.1$, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, and 0.9 (solid), and $\zeta = 1$ (dot-dashed).

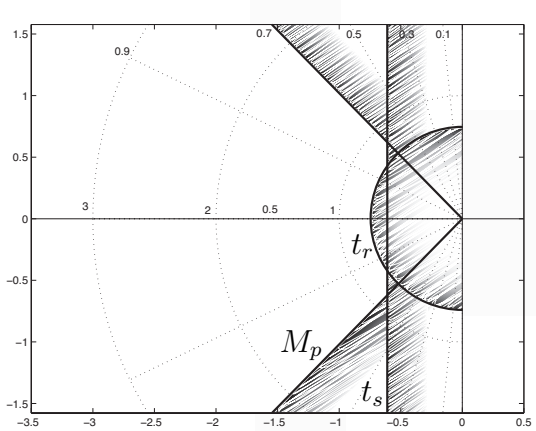


Figure 8.4: Approximate constraints, or **design guides**, on the admissible pole locations of a CT second-order system (or a higher-order system whose response is dominated by a pair of second-order poles) in the complex plane s in order to not exceed specified constraints on the rise time, settling time, and overshoot of the system's step response (see Figure 8.3), as specified in (8.17).

8.3 Z transform methods

The Z **transform** (a.k.a. the **unilateral** Z **transform**), $F(z)$, of a **discrete-time** (**DT**) signal f_k for $k = 0, 1, 2, \ldots$ is defined by

$$F(z) = \sum_{k=0}^{\infty} f_k z^{-k}. \tag{8.18a}$$

Given f_k for $k = 0, 1, 2, \ldots$, we define $F(z)$ via (8.18a). The **inverse** Z **transform** is given by

$$f_k = \frac{1}{2\pi \mathrm{i}} \oint_\Gamma F(z) z^{k-1}\, dz, \tag{8.18b}$$

where the complex contour Γ is a circle around the origin in complex plane z that is chosen to be of sufficiently small radius that it does not contain any singularities[12] of $F(z)$ in the complex plane z.

Verification that (8.18b) represents the inverse of the relationship expressed in (8.18a) is straightforward: substitute (8.18a) into the RHS of (8.18b) and note that $\int_{-\pi}^{\pi} e^{\mathrm{i}\, n\, \theta}\, d\theta = 0$ for integer $n \neq 0$; it is seen that, for a contour Γ given by $z = Re^{\mathrm{i}\theta}$ for $\theta = (-\pi, \pi)$ with sufficiently small R (thus, $dz = \mathrm{i}Re^{\mathrm{i}\theta}\, d\theta$), f_k is indeed recovered:

$$\begin{aligned}
\frac{1}{2\pi \mathrm{i}} \oint_\Gamma \Big[\sum_{k'=0}^{\infty} f_{k'} z^{-k'} \Big] z^{k-1}\, dz &= \frac{1}{2\pi \mathrm{i}} \int_{-\pi}^{\pi} \sum_{k'=0}^{\infty} f_{k'} R^{-k'} e^{-\mathrm{i}\theta k'} R^{k-1} e^{\mathrm{i}\theta(k-1)} R \mathrm{i} e^{\mathrm{i}\theta}\, d\theta \\
&= \sum_{k'=0}^{\infty} f_{k'} R^{k-k'} \Big[\frac{1}{2\pi} \int_{-\pi}^{\pi} e^{\mathrm{i}\theta(k-k')}\, d\theta \Big] = \sum_{k'=0}^{\infty} f_{k'} R^{k-k'} \delta_{k,k'} = f_k,
\end{aligned}$$

The reason that the $R^{k-k'}$ factor, for sufficiently small positive R, is required by this formula is to ensure that the magnitude of the integrand decays to zero exponentially as $k' \to \infty$, which allows us to swap the order of the integral and the sum using Fubini's theorem.

As shown below, the forward and inverse transforms expressed by (8.18) are immensely useful when solving difference equations (in DT). By (8.18a), knowing f_k for $k = 0, 1, 2, \ldots$, one can define $F(z)$ on an appropriate contour. Conversely, by (8.18b), knowing $F(z)$ on an appropriate contour, one can determine f_k for $k = 0, 1, 2, \ldots$ As in §8.2, before demonstrating further *why* such a transformation is useful, we first mention that, in practice, you don't actually need to compute the somewhat involved integrals given in (8.18) in order to use the Z transform effectively. Rather, it is sufficient to reference a table listing some Z transform pairs in a few special cases, as shown in Table 8.1b. Note also the following:

Fact 8.8 *The Z* **transform is linear***; that is,* **superposition holds***, and thus if the Z transforms of the sequences x_k and y_k are $X(z)$ and $Y(z)$, then the Z transform of the sequence $w_k = \alpha x_k + \beta y_k$ is $W(z) = \alpha X(z) + \beta Y(z)$.*

Fact 8.9 *If the Z transform of the sequence f_k is $F(z)$, then the Z transform of the* **scaled** *sequence $g_k = b^k f_k$ is $G(z) = \sum_{k=0}^{\infty} f_k (z/b)^{-k} = F(z/b)$, and the Z transform of the* **delayed** *sequence $g_k = f_{k-d}$ is $G(z) = \sum_{k=0}^{\infty} f_{k-d} z^{-k} = F(z)/z^d$.*

[12] That is, the circular contour of integration in the (8.18b) is chosen to be of sufficiently small radius that it does not contain any points $\overline{z}$ for which $|F(z)| \to \infty$ as $z \to \overline{z}$ in (8.18a).

8.3.1 The Z Transform of translated sequences

Define $f_k^{[1]} = f_{k+1}$ for $k = 0, 1, 2, \ldots$. Then the Z transform of $f_k^{[1]}$ is given by

$$F^{[1]}(z) = \sum_{k=0}^{\infty} f_k^{[1]} z^{-k} = z \sum_{k=0}^{\infty} f_{k+1} z^{-(k+1)} = z \sum_{k=1}^{\infty} f_k z^{-k} = zF(z) - zf_0. \tag{8.19}$$

Similarly, if $f_k^{[2]} = f_{k+2}$ and $f_k^{[n]} = f_{k+n}$, then

$$F^{[2]}(z) = \sum_{k=0}^{\infty} f_k^{[2]} z^{-k} = z^2 F(z) - z^2 f_0 - z f_1, \qquad F^{[n]}(z) = \sum_{k=0}^{\infty} f_k^{[n]} z^{-k} = z^n F(z) - z^n f_0 - z^{n-1} f_1 - \ldots - z f_{n-1}.$$

Thus, if $f_k^{[1]} = f_{k+1}$, then $F^{[1]}(z) = zF(z) - zf_0$. Conversely, it follows that, if $f_{k+1} = f_k^{[1]}$ for $k = 0, 1, 2, \ldots$ with $f_0 = 0$, then $F(z) = \frac{1}{z} F^{[1]}(z)$. We thus arrive at the most useful interpretation of the z variable:

Fact 8.10 *Multiplication of the Z transform of a DT signal by z corresponds to an advance of this signal by one timestep, multiplication times z^2 corresponds to an advance by two timesteps, etc. Conversely, multiplication by $1/z$ corresponds to a delay by one timestep, multiplication by $1/z^2$ corresponds to a delay by two timesteps, etc.*

Defining a new sequence $g_k = f_{k+1} - f_k$ for all k and taking the Z transform of g_k, applying (8.19), gives

$$G(z) = \sum_{k=0}^{\infty} g_k z^{-k} \quad \Rightarrow \quad [zF(z) - zf_0] - F(z) = \lim_{a \to \infty} \sum_{k=0}^{a-1} (f_{k+1} - f_k) z^{-k}.$$

Taking the limit of this expression as $z \to 1$, noting that the limit on the RHS approaches $f_\infty - f_0$ if the limit indicated in the above equation is bounded, thus gives

Fact 8.11 (The DT final value theorem) *If $\lim_{k \to \infty} f_k$ is bounded, then $\lim_{z \to 1} (z-1) F(z) = \lim_{k \to \infty} f_k$.*

On the other hand, it follows directly from the $z \to \infty$ limit of (8.18a) that

Fact 8.12 (The DT initial value theorem) $\lim_{z \to \infty} F(z) = f_0$.

8.3.2 Using the Z Transform to solve unforced linear difference equations

Now consider the unforced linear constant-coefficient second-order difference equation given by

$$f_{k+2} + a_1 f_{k+1} + a_0 f_k = 0 \quad \text{with} \quad f_0,\ f_1 \text{ given.} \tag{8.20}$$

Taking the Z transform of this equation and applying the above relations gives

$$\sum_{k=0}^{\infty} \{f_{k+2} + a_1 f_{k+1} + a_0 f_k = 0\} z^{-k} \quad \Rightarrow \quad [z^2 F(z) - z^2 f_0 - z f_1] + a_1 [zF(z) - z f_0] + a_0 [F(z)] = 0$$

$$\Rightarrow \quad F(z) = \frac{c_2 z^2 + c_1 z}{z^2 + a_1 z + a_0} \quad \text{where } \ c_2 = f_0, \ \ c_1 = f_1 + a_1 f_0.$$

Defining $p_\pm = (-a_1 \pm \sqrt{a_1^2 - 4a_0})/2$ and performing a partial fraction expansion, it follows that

$$F(z) = \frac{c_2 z^2 + c_1 z}{(z-p_+)(z-p_-)} = \frac{d_+ z}{z-p_+} + \frac{d_- z}{z-p_-} \quad \Rightarrow \quad \left\{ \begin{array}{r} d_+ + d_- = c_2 \\ -d_+ p_- - d_- p_+ = c_1 \end{array} \right\} \quad \Rightarrow \quad \left\{ \begin{array}{l} d_+ = \dfrac{c_2 p_+ + c_1}{p_+ - p_-} \\ d_- = \dfrac{c_2 p_- + c_1}{p_- - p_+}. \end{array} \right.$$

Thus, by Table 8.1b and the linearity of the Z transform (Fact 8.8), we deduce that

$$f_k = d_+ p_+^k + d_- p_-^k, \tag{8.21}$$

thus solving the difference equation (8.20) [i.e., f_k for any k can be calculated directly, without marching (8.20)]. It is seen that, if the magnitudess of both p_+ and p_- are less than one, the magnitude of the solution decays with time, whereas if the magnitudes of either is greater than one, the magnitude of the solution grows with time. Note also that taking $f_0 = 0$ and $f_1 = 1$ and $a_0 = a_1 = -1$ in (8.20) generates to Fibonacci's sequence.

Higher-order linear difference equations may be solved in an identical manner, leveraging partial fraction expansions to split up $F(z)$ into simple terms whose Z transforms may be found in Table 8.1b.

8.3.3 Discrete-time (DT) transfer functions

Now consider the forced linear constant-coefficient second-order difference equation, a.k.a. DT SISO LTI system, for u_k (the output[13]) given by

$$u_{k+2} + a_1\, u_{k+1} + a_0\, u_k = b_0\, e_k, \tag{8.22}$$

where e_k (the input) is specified, assuming u_k and e_k are zero for $k < 0$. Taking the Z transform now gives

$$\sum_0^\infty \{u_{k+2} + a_1\, u_{k+1} + a_0\, u_k = b_0\, e_k\} z^{-k} \quad \Rightarrow \quad [z^2 + a_1\, z + a_0] U(z) = b_0\, E(z)$$

$$\Rightarrow \quad D(z) \triangleq \frac{U(z)}{E(z)} = \frac{b_0}{z^2 + a_1\, z + a_0} = \frac{b_0}{(z-p_+)(z-p_-)}, \tag{8.23}$$

where, again, the poles $p_\pm = \big(-a_1 \pm \sqrt{a_1^2 - 4\, a_0}\big)/2$. The quantity $D(z)$ is known as the transfer function of the linear system (8.22). Higher-order forced SISO constant-coefficient DT linear systems of the form

$$u_{k+n} + a_{n-1}\, u_{k+n-1} + \ldots + a_1\, u_{k+1} + a_0\, u_k = b_m\, e_{k+m} + b_{m-1}\, e_{k+m-1} + \ldots + b_1\, e_{k+1} + b_0\, e_k \tag{8.24a}$$

with $b_m \neq 0$ [and, normally, $n \geq m$; see §8.3.3.2], may be manipulated in an analogous manner, leading to a transfer function of the form

$$D(z) = \frac{U(z)}{E(z)} = \frac{b_m\, z^m + b_{m-1}\, z^{m-1} + \ldots + b_1\, z + b_0}{z^n + a_{n-1}\, z^{n-1} + \ldots + a_1\, z + a_0} = K \frac{(z-z_1)(z-z_2)\cdots(z-z_m)}{(z-p_1)(z-p_2)\cdots(z-p_n)}. \tag{8.24b}$$

By comparison, the parallels with the CT case in §8.2.3 are clear. Note that, in implementation [see §1.5.3.2], it is often more convenient to write (8.24a) [in the case that $n \geq m$] as

$$u_k = -\overline{a}_1\, u_{k-1} - \ldots - \overline{a}_{n-1}\, u_{k-(n-1)} - \overline{a}_n\, u_{k-n} + \overline{b}_0 e_k + \overline{b}_1 e_{k-1} + \ldots + \overline{b}_{n-1} e_{k-(n-1)} + \overline{b}_n e_{k-n}, \tag{8.24c}$$

[13] For the sake of later convenience (in §10), we have changed the letters associated with the inputs and outputs in §8.3.3, where we consider a **DT controller** $D(z) = U(z)/E(z)$, as compared with §8.2.3, where we considered a **CT plant** $G(s) = Y(s)/U(s)$.

where, for convenience, we have renumbered the coefficients $\overline{a}_k = a_{n-k}$ and $\overline{b}_k = b_{n-k}$; this form is often referred to as a **finite impulse response** (**FIR**) filter if $\overline{a}_k = 0$ for $k > 0$, and an **infinite impulse response** (**IIR**) filter if not. Note that the FIR case is distinguished by an impulse response that vanishes after finite number of steps, whereas [due to the feedback built in to the difference equation (8.24c)] the IIR case is not.

Again, a difference equation governing a DT system simply relates linear combinations of two or more variables describing the system and their **tap delays**; such an equation does *not* itself indicate one variable as a "cause" and another as an "effect". However, the definition of a transfer function implies an input-output relationship; in the examples discussed above, e_k is the input, and u_k is the output. Almost all systems encountered are causal, meaning the variable identified as the output only responds to the current and past inputs, but not to future inputs. In the DT setting, this happens when $n \geq m$; for further discussion, see §8.3.3.2.

In the $n > m$ ("strictly causal") case, the MCU has a full timestep h to calculate the RHS of (8.24c) before changing the output of u_{k-1} to u_k. In the $n = m$ ("semi-causal") case, the MCU can first calculate all of the explicit terms (i.e., those corresponding to previous timesteps) on the RHS of (8.24c). Then, when the measurement e_k comes in, the term $\overline{b}_0\, e_k$ can be calculated with high priority (see §2.1.4) and added to the result to generate u_k, which can then be updated on the output pin shortly after the measurement e_k is received.

Once a (causal) DT linear system's transfer function is known, its response to simple inputs is easy to compute. Noting Table 8.1b, if e_k is a unit impulse (that is, $e_k = \delta_{0,k}$), then $E(z) = 1$, and if e_k is a unit step [that is, $e_k = 1$ for $k \geq 0$], then $E(z) = z/(z-1)$. In both cases, $U(z)$ is easy to compute from (8.23), and thus u_k may be found by partial fraction expansion and subsequent inverse Z transform.

As in the CT case, it is important to keep clear the distinction between the Z transform (a.k.a. transfer function) of a *system*, such as $D(z)$ above, and the Z transform of a *signal*, such as $E(z)$ above. To make clear the connection between them, note in the special case that the input to the system happens to be a unit impulse $e_k = \delta_{0,k}$, it follows that $E(z) = 1$ and thus $U(z) = D(z)$. In other words,

Fact 8.13 *The transfer function of a DT linear system is the Z transform of its impulse response.*

It follows from the relation $U(z) = D(z)\, E(z)$, expanding $U(z)$, $D(z)$, and $E(z)$ with the Z transform formula (8.18a), noting that the impulse response $d_k = 0$ for $k < 0$ (that is, that the DT system is causal), and following an analogous derivation as that leading to (8.16), that

$$\sum_{k=0}^{\infty} \Big[u_k\Big] z^{-k} = \sum_{j=0}^{\infty} e_j z^{-j} \sum_{k=0}^{\infty} d_k z^{-k} = \sum_{j=0}^{\infty} e_j \left(\sum_{k=-j}^{\infty} d_k z^{-k} \right) z^{-j}$$
$$= \sum_{j=0}^{\infty} e_j \left(\sum_{k=0}^{\infty} d_{k-j} z^{-(k-j)} \right) z^{-j} = \sum_{k=0}^{\infty} \left[\sum_{j=0}^{k} e_j d_{k-j} \right] z^{-k},$$

from which we deduce that, for $k \geq 0$,

$$u_k = \sum_{j=0}^{k} e_j\, d_{k-j}; \tag{8.25}$$

note in particular that $u_k = d_k$ when $e_j = \delta_{j,0}$. Thus, as similarly noted in the CT case,

Fact 8.14 *The product $U(z) = D(z)\, E(z)$ in Z transform space corresponds to a convolution sum [of the input e_k with the impulse response d_k] in the untransformed space.*

Products are generally much easier to work with than convolution sums, thus highlighting the utility of the Z transform when solving constant-coefficient DT linear systems.

8.3.3.1 The transfer function of a DAC – $G(s)$ – ADC cascade

By Fact 8.13, we may determine the transfer function of a DT system, $G(z)$, simply by computing the response of the system to an impulse input, $u_k = \delta_{0,k}$, then taking the Z transform of this response. Applying this experiment to a cascade of components given by (i) a **digital-to-analog converter** (**DAC**) implementing a **zero-order-hold**[14] (**ZOH**), (ii) a CT system $G(s)$, and (iii) an **analog-to-digital converter** (**ADC**) , noting in this case that $u(t)$ [that is, the input to $G(s)$] is simply a unit step (with Laplace transform $1/s$ in CT) followed by a one-timestep-delayed negative unit step, it follows that

$$G(z) = \mathcal{Z}\Big\{\frac{1-e^{-sh}}{s}\,G(s)\Big\} = (1-z^{-1})\mathcal{Z}\Big\{\frac{G(s)}{s}\Big\} = \frac{z-1}{z}\,\mathcal{Z}\Big\{\frac{G(s)}{s}\Big\}, \tag{8.26}$$

where z^{-1} corresponds to as one-timestep delay, with Laplace transform e^{-sh}, and the shorthand $\mathcal{Z}\{G(s)/s\}$ means the Z transform of the discretization of the CT signal whose Laplace transform is $G(s)/s$. We will make use of this convenient (and exact!) conversion, implemented in RR_C2D_zoh, in §10.4.2.

8.3.3.2 Causal, strictly causal, and noncausal DT systems

Define the **relative degree** of the DT transfer function in (8.24b) as $n_r = n - m$, where n is the degree of the polynomial in the denominator, and m is the degree of the polynomial in the numerator. DT systems with $n_r < 0$ are **noncausal** (i.e., the output depends, in part, on future values of the input). In §10 we will further distinguish the systems of interest as "plants" and "controllers"s. All real DT plants $G(z)$, or DT analogs of CT proper (see §8.2.3.1) plants [formed, e.g., via the technique given in (8.26) of §10.4.2], are **causal**, with $n_r \geq 0$. Further, any controller $D(z)$ must only be based on available measurements, and thus must also be causal, with $n_r \geq 0$. A DT transfer function with $n_r = 0$ in (8.24a) [that is, with $\bar{b}_0 \neq 0$ in (8.24c)] is said to be **semi-causal**. If there is significant computation time necessary to compute the control (in digital electronics) before it can be applied to the system, it is sometimes convenient to restrict the controller to be **strictly causal**, with $n_r > 0$.

If the output depends only on the current and *future* inputs, the transfer function is said to be **anti-causal**, and if the output depends strictly on future inputs, it is said to be **strictly anti-causal**. We will focus our attention exclusively on the causal case, with $n_r \geq 0$.

Example 8.2 The step response of second-order DT linear systems

We now focus further on the forced second-order case (8.22), written as $Y(z) = G(z)\,U(z)$, with $b_0 = 1+a_1+a_0$, when forced by a unit step $u_k = 1$ for $k \geq 0$; that is,

$$G(z) = \frac{1+a_1+a_0}{z^2+a_1 z+a_0} = \frac{1+a_1+a_0}{(z-p_+)(z-p_-)} \quad \text{and} \quad U(z) = \frac{z}{z-1}$$

where $1+a_1+a_0 = (1-p_+)(1-p_-)$. Assuming the poles are complex, $p_\pm = \big(-a_1 \pm \sqrt{a_1^2-4a_0}\big)/2 = r\,e^{\pm \mathrm{i}\theta}$ with

$$r = \sqrt{a_0} \quad \text{and} \quad \theta = \cos^{-1}[-a_1/(2r)],$$

and have magnitude less than one (i.e., $a_1^2/4 < a_0 < 1$), the solution of this system may again be written in terms of sines and cosines modulated by a decaying exponential: writing the partial fraction expansion

$$Y(z) = G(z)U(z) = \frac{1+a_1+a_0}{(z-p_+)(z-p_-)}\cdot\frac{z}{z-1} = \frac{d_+ p_+}{z-p_+} + \frac{d_- p_-}{z-p_-} + \frac{d_0}{z-1} \qquad \begin{cases} d_+ = \frac{1+a_1+a_0}{(p_+-p_-)(p_+-1)}, \\ d_- = \frac{-(1+a_1+a_0)}{(p_+-p_-)(p_--1)} = \overline{d_+}, \\ d_0 = 1. \end{cases}$$

[14] That is, holding the value of the analog signal as constant between timesteps

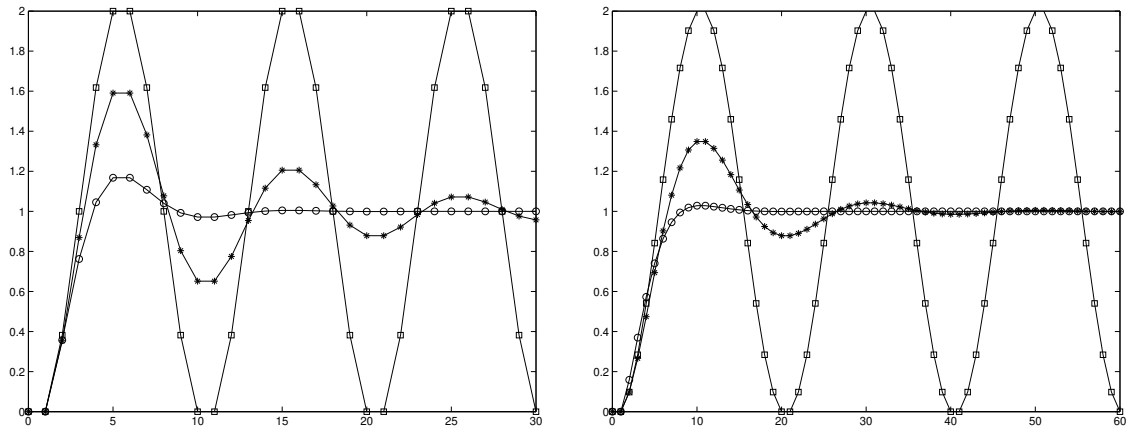


Figure 8.5: The unit step response of the system $u_{k+2} + a_1 u_{k+1} + a_0 u_k = b_0 e_k$, where $b_0 = 1 + a_1 + a_0$, with $r = 0.7$ (circles), $r = 0.9$ (asterisks), and $r = 1.0$ (squares) for $\theta = \pi/5$ (left) and $\theta = \pi/10$ (right), where $r = \sqrt{a_0}$ and $\theta = \cos^{-1}[-a_1/(2r)]$ (cf. Figure 8.3b). The lines in this figure are drawn to improve readability; the DT signals are defined only at each step, indicated by the symbols.

and computing the inverse Z transform of $Y(z)$ via Table 8.1b, noting that $e^{\mathrm{i}\,\theta\,k} = \cos(\theta\,k) + \mathrm{i}\sin(\theta\,k)$, the closed-form solution of y_k for $k > 0$ is

$$y_k = d_+ p_+^k + d_- p_-^k + d_0 = r^k \big[d_c \cos(\theta\,k) + d_s \sin(\theta\,k) \big] + 1, \qquad \begin{cases} d_c = d_+ + d_- = -1, \\ d_s = \mathrm{i}(d_+ - d_-) = -(a_1 + 2)/\sqrt{4a_0 - a_1^2}, \end{cases}$$

as plotted in Figure 8.5 using the corresponding code code in RR.ch08. As in the CT case, since the system $G(z)$ considered in this example is real, the complex poles $\{p_+, p_-\}$ come as a conjugate pair. In addition, as consequence of the fact that the input u_k to this system is also real, the coefficients $\{d_+, d_-\}$ also work out to be a complex conjugate pair, and thus $\{d_c, d_s, d_0\}$, and y_k itself, are real. Again, the speed of oscillation θ and the rate of decay r of this response are a function of the location of the poles of the transfer function $p_\pm = r\,e^{\mathrm{i}\theta}$. Note also that $y_0 = y_1 = 0$; this follows directly from (8.22), noting the $k + 2$ subscript on y on the LHS and the k subscript on u on the RHS.

As evident in Figure 8.5, rise time t_r, settling time t_s, and overshoot M_p characterizations, introduced in the CT case in Figure 8.3, may also be defined in the DT case. Appropriate design guides for the pole locations in the z plane in order to ensure specified maximum values of t_r, t_s, and M_p are presented in the following subsection.

8.3.4 Reconciling the Laplace and Z transforms

We now revisit the Laplace transform as defined in (8.7a) and the Z transform as defined in (8.18a):

$$F(s) = \int_0^\infty f(t) e^{-s\,t} dt, \qquad F(z) = \sum_{k=0}^{\infty} f_k z^{-k}.$$

Note that, if we take $z = e^{sh}$ where h is the timestep [that is, $t_k = hk$ and $f_k = f(t_k)$], and if h is small as compared with the time scales of the variation of $f(t)$, then $F(z)$, scaled by h, is a rectangular-rule approximation of $F(s)$. Another way of making this connection between the CT analysis and the DT analysis is by comparing the closed-form solutions of the step responses of second-order CT and DT systems, as given in Examples 8.1 and 8.2. We see that the latter response is simply a discretization of the former if $r^k = e^{-\sigma t}$ and $\theta k = \omega_d t$; that

is, if the CT second-order pole locations $s_\pm = -\sigma \pm \mathrm{i}\,\omega_d$ and the DT second-order pole locations $z_\pm = re^{\pm \mathrm{i}\theta}$ are related such that $r = e^{-\sigma h}$ and $\theta = \omega_d h$, and thus $z_\pm = re^{\pm \mathrm{i}\theta} = e^{(-\sigma \pm \mathrm{i}\omega_d)h} = e^{s_\pm h}$.

Thus, the pole locations in DT and CT are related by the mapping

$$z = e^{sh} = 1 + sh + \frac{s^2h^2}{2!} + \frac{s^3h^3}{3!} + \ldots, \tag{8.27}$$

as indicated in Figure 8.6. This connection is quite significant. For example, the approximate design guides for CT systems dominated by a pair of second-order poles, as illustrated in Figure 8.4, may be mapped immediately using this relation to obtain corresponding approximate design guides for DT second-order systems, as illustrated in Figure 8.7. It is seen that, for sinusoidal signals (that is, for $r = 1$), the number of timesteps per oscillation is $2\pi/\theta$. It is also seen that the settling time is related to r, with $r = 0.9$ corresponding to a settling time of 43 timesteps, $r = 0.8$ corresponding to a settling time of 21 timesteps, and $r = 0.6$ corresponding to a settling time of 9 timesteps.

For small h, (8.27) provides a simple connection between s-plane pole locations in the vicinity of $s = 0$ and the (scaled) z-plane pole locations in the vicinity of $z = 1$ via **Euler's approximation**

$$z \approx 1 + sh. \tag{8.28}$$

That is, for small h, the neighborhood of $z = 1$ in the z plane may be interpreted in a similar fashion as the (scaled) neighborhood of $s = 0$ in the s plane, and the three families of design guides (for t_r, t_s, and M_p) in these two regions indeed look quite similar.

8.3.4.1 Tustin's approximation

For larger h, Euler's approximation is not accurate. Motivated by the accuracy analysis of the CN method given in §7, the following rational approximation of (8.27), referred to in this setting as **Tustin's approximation** (aka the **bilinear approximation**, is preferred for most applications:

$$z \approx \frac{1 + sh/2}{1 - sh/2} = 1 + sh + \frac{s^2h^2}{2} + \frac{s^3h^3}{4} + \ldots \quad \Leftrightarrow \quad s \approx \frac{2}{h}\frac{z-1}{z+1}. \tag{8.29}$$

Conveniently, both the exact mapping (8.27) and Tustin's approximation (8.29) map the left half plane of s to the interior of the unit circle in z; in particular, the stability boundary of s (the imaginary axis) maps to the stability boundary of z (the unit circle). This is why Tustin's approximation is strongly preferred over Euler's approximation (8.28), or other manners of truncating or approximating (8.27).

To see how to use Tustin's rule to approximate a general CT differential equation [interpreted in §10 as a controller] whose Laplace transform is $D(s)$ with a DT difference equation whose Z transform is $D(z)$, it is useful to consider first the transfer function of the following simple differential equation, with $u(t)$ and $e(t)$ taken to be zero for $t < 0$:

$$\frac{U(s)}{E(s)} = D(s) = \frac{s+a}{s+p} \quad \Rightarrow \quad (s+p)U(s) = (s+a)E(s) \quad \Rightarrow \quad \frac{du}{dt} + pu = \frac{de}{dt} + ae.$$

Approximating the time derivatives in this ODE with the CN method (see §7), we may write

$$\frac{u_k - u_{k-1}}{h} + p\frac{u_k + u_{k-1}}{2} = \frac{e_k - e_{k-1}}{h} + a\frac{e_k + e_{k-1}}{2}.$$

Taking the Z transform of this difference equation and rearranging leads immediately to

$$D(z) = \frac{U(z)}{E(z)} = \frac{\frac{2}{h}\frac{z-1}{z+1} + a}{\frac{2}{h}\frac{z-1}{z+1} + p} = \left.\frac{s+a}{s+p}\right|_{s=\frac{2}{h}\frac{z-1}{z+1}} = \left.D(s)\right|_{s=\frac{2}{h}\frac{z-1}{z+1}}. \tag{8.30}$$

Using Tustin's rule (8.29), higher-order CT transfer functions $D(s)$ may similarly be approximated with a corresponding DT transfer functions $D(z)$, simply replacing each occurence of s in $D(s)$ with $\frac{2}{h}\frac{z-1}{z+1}$, then reducing to a rational expression in z, as illustrated in (8.30).

8.3.4.2 Tustin's approximation with prewarping

The exact mapping (8.27) maps the interval on the imaginary axis between $s = 0$ and $s = \mathrm{i}\pi/h$ to the edge of the upper half of the unit circle (see Figure 8.6); in contrast, Tustin's rule (8.29) maps the entire upper half of the imaginary axis to the same region. Thus, though the stability boundaries of these two mappings coincide, the mapping due to Tustin's rule is **warped**, and is only accurate in the vicinity of $s = 0$ and $z = 1$. When designing controllers for mixed DT/CT systems (see §10.4), there is often a frequency $\overline{\omega}$ of primary concern, such as a gain crossover frequency (see §10.2) or notch frequency (see §10.3.2). It is easy to adjust Tustin's rule via a **prewarping** strategy that *scales the s plane* by a factor $f > 1$ prior to mapping it to the z plane, thus recovering the exact mapping (8.27) for the point $s = \mathrm{i}\overline{\omega}$ (for some $\overline{\omega} < \pi/h$) and providing a rational and accurate approximation of this mapping for points in the vicinity of $s = \mathrm{i}\overline{\omega}$ without disrupting the correspondence of the two stability boundaries given by the exact mapping. To accomplish this, define

$$e^{\mathrm{i}\overline{\omega}h} = \frac{1 + \mathrm{i}f\overline{\omega}h/2}{1 - \mathrm{i}f\overline{\omega}h/2} \quad \Rightarrow \quad f = \frac{2[1 - \cos(\overline{\omega}h)]}{\overline{\omega}h\sin(\overline{\omega}h)}.$$

Note that, when $\overline{\omega}$ is in the range $0 \leq \overline{\omega} < \pi/h$, the factor f is in the range $1 \leq f < \infty$; note specifically that $f \to 1$ as $\overline{\omega} \to 0$. We may then modify Tustin's rule (8.29) such that

$$z \approx \frac{1 + fsh/2}{1 - fsh/2} \quad \Leftrightarrow \quad s \approx \frac{2}{fh}\frac{z-1}{z+1}. \tag{8.31}$$

This is referred to as **Tustin's rule with prewarping**, and is conveniently implemented in code in RR_C2D_tustin; this rule is used in §10.4.1 to develop DT controllers $D(z)$ which have the desired behavior near a particular frequency of interest $\bar{\omega}$, mimicking the behavior of effective CT controllers $D(s)$ designed for CT plants[15].

[15] Though Tustin's rule is the method of choice for converting $D(s)$ into a DT $D(z)$, various simple alternatives to this method are sometimes enlightening to consider. For example, with the heuristic **pole-zero mapping** (a.k.a. **matched z-transform**) approach:
(i) All poles and finite zeros of $D(s)$ are mapped to $D(z)$ via $z = e^{sh}$.
(ii) All infinite zeros of $D(s)$ are mapped $z = -1$ in $D(z)$ (effectively, to the highest-frequency point on the stability boundary in the z plane). If a strictly causal $D(z)$ is required (see §8.3.3.2), one of the infinite zeros is instead mapped to $z = \infty$ in $D(z)$.
(iii) The gain of $D(z)$ at $z = e^{\mathrm{i}\overline{\omega}h}$ is chosen to match the gain of $D(s)$ at $s = \mathrm{i}\overline{\omega}$, either for $\overline{\omega} = 0$, or (better) for some critical $\overline{\omega}$ of interest.

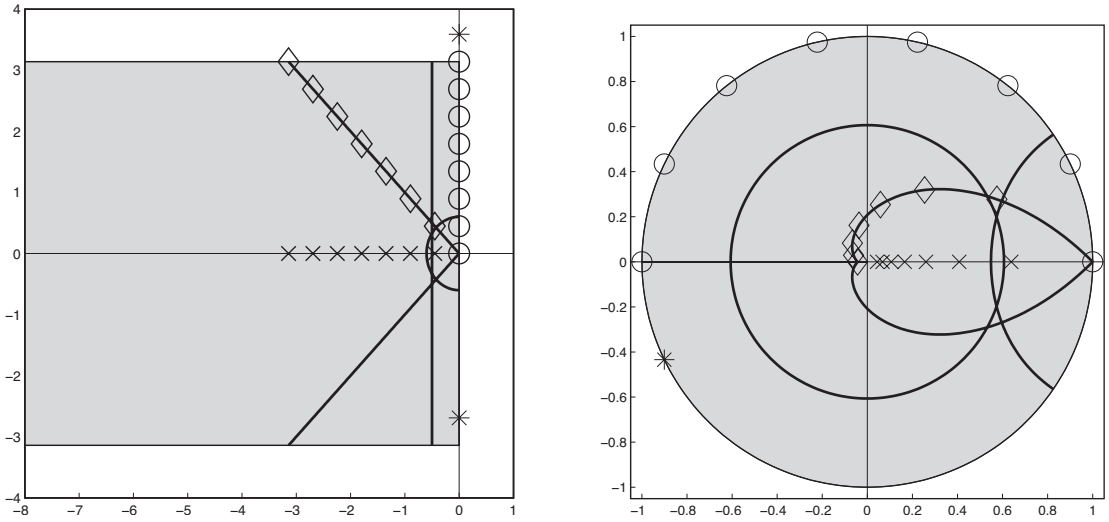


Figure 8.6: The mapping of several curves and points between the s plane (left) and the z plane (right) using (8.27). Taking $s = a + b\mathrm{i}$ and $z = re^{\mathrm{i}\theta}$, the shaded strip in the s plane with $-\infty < a \leq 0$ and $-\pi/h \leq b \leq \pi/h$ maps uniquely to the shaded disk in the z plane with $r \leq 1$. Points above and below this strip in the s-plane do not map uniquely to points in the z-plane; e.g., both points marked by asterisks in the s plane map to the same point marked in the z plane. This is a manifestation of the aliasing phenomenon depicted in Figure 8.7a.

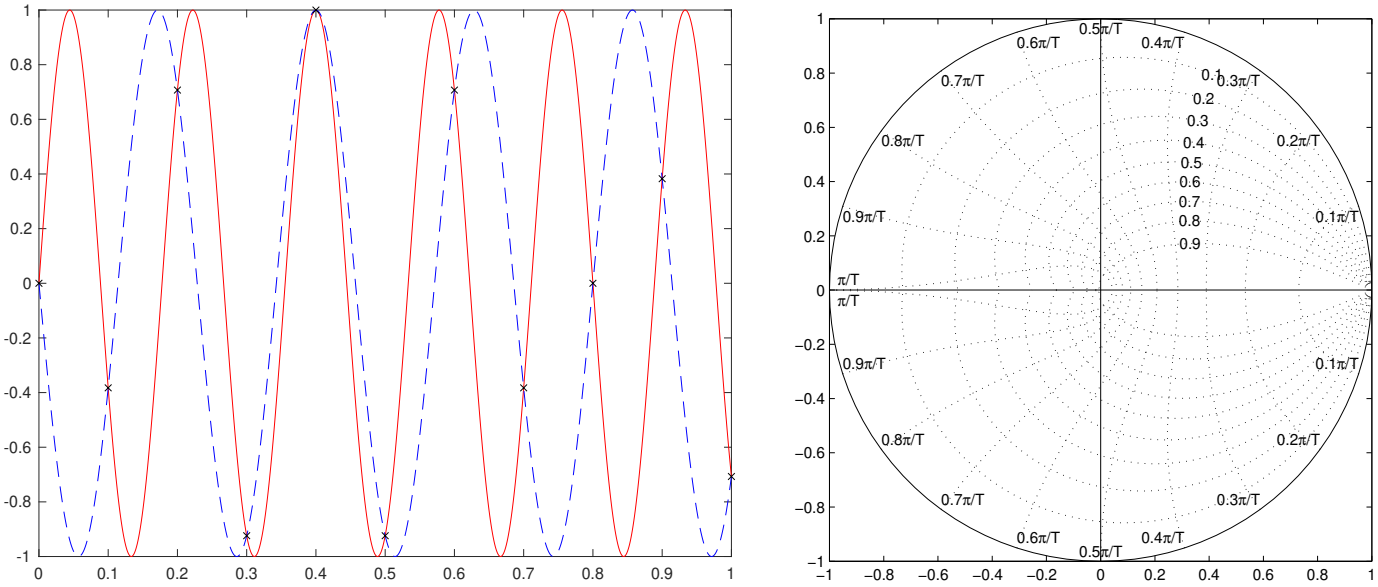


Figure 8.7: (a) Demonstration of the aliasing apparent in Figure 8.6: taking $h = 0.1$, DT samples (indicated by the $\times$ symbols) of $\sin(\omega_1 t)$ and $\sin(\omega_2 t)$, for $\omega_1 = (9/8)\pi/h$ and $\omega_2 = -(7/8)\pi/h$ are coincident, and thus, based on these samples alone, one can not distinguish which sine wave from which they were sampled. (b) Approximate constraints, or design guides, on the admissible pole locations of a DT second-order system (or a higher-order system whose response is dominated by a pair of second-order poles) in the complex plane z in order to not exceed specified constraints on the rise time and overshoot of the system's step response (see Figure 8.5). These DT design guides are found simply by mapping the corresponding CT design guides (see Figure 8.4) using (8.27), and may be drawn with the command `zgrid` in Matlab syntax. Around the circumference are marked the values of ω_d, from $0.1\pi/T$ to π/T (with h denoted T...), and in the upper-right quadrant are marked the values of ζ, from 0.1 to 0.9, for the corresponding CT second-order design guides discussed in Example 8.1.

8.4 Bode plots: thinking in the frequency domain

The **Bode plot** (a.k.a. **open-loop Bode plot**) is best introduced via a simple experiment: if a stable SISO linear system $G(s) = Y(s)/U(s)$, with all poles in the LHP, is excited with a sinusoidal input $u(t) = \cos(\omega\, t)$, then the output $y(t)$ (which, if $G(s)$ is known, may be determined via partial fraction expansion) will be composed of several components which decay exponentially in time, plus a sinusoidal component of the same frequency ω as the input but with a different magnitude and phase, $y(t) = A\cos(\omega\, t+\phi)+$ decaying terms. The Bode plot shows the gain in magnitude, A, and change in phase, ϕ, of this persistent component of the output over a range of sinusoidal input frequencies ω of interest; the plot of the gain A vs. frequency ω is represented in **loglog** form, and the plot of the phase change ϕ versus frequency ω is represented in **semilogx** form. If the system is MIMO, a Bode plot may be developed for every input/output combination. Bode plots of two important simple systems (first-order and second-order low pass filters, as discussed further in §8.5) are given in Figure 8.8; Bode plots of two more complicated systems (with multiple breakpoints) are given in Figure 8.9.

The fact that any sinusoidal input in the experiment described above eventually leads to a sinusoidal output at the same frequency, but at a different magnitude and phase, is clearly seen if we consider first what happens if we put a *complex* input $u_1(t) = e^{\mathrm{i}\omega t}$ into a SISO system. [This is not possible in a real physical experiment, of course, but can easily be done as a *Gedankenexperiment* if we know the transfer function $G(s)$ of the (stable) system under consideration.] In this case, by Table 8.1, $U_1(s) = 1/(s-p_0)$ where $p_0 = \mathrm{i}\,\omega$, and thus the partial fraction expansion of the output $Y_1(s)$ may be written [see Footnote 10 in §8.2.2] as

$$Y_1(s) = G(s)U_1(s) = d_0/(s-p_0) + \text{other terms} \qquad \Rightarrow \qquad y_1(t) = d_0 e^{\mathrm{i}\omega t} + \text{other terms}.$$

The "other terms" in the partial fraction expansion of $Y_1(s)$ all have their poles in the LHP, because $G(s)$ is assumed to be stable, and thus the "other terms" in the corresponding inverse Laplace transform, $y_1(t)$, are all stable. Thus, the magnitude and phase of the persistent component of the output is given by the magnitude and phase of the complex coefficient d_0 which, by the discussion in §8.2.2, may be found simply as follows:

$$d_0 = \Big[Y_1(s)\cdot(s-p_0)\Big]_{s\to \mathrm{i}\omega} = \Big[G(s)\frac{1}{s-p_0}\cdot(s-p_0)\Big]_{s\to \mathrm{i}\omega} = G(\mathrm{i}\,\omega).$$

The magnitude and phase shift of the persistent sinusoidal component $d_0 e^{\mathrm{i}\omega t}$ of the output $y_1(t)$ are simply the magnitude and phase of $G(\mathrm{i}\,\omega)$. For the complex input $u_2(t) = e^{-\mathrm{i}\omega t}$, the result is similar:

$$U_2(s) = 1/(s+p_0) \quad \text{where } p_0 = \mathrm{i}\,\omega, \qquad Y_2(s) = G(s)U_2(s) = c_0/(s+p_0) + \text{other terms} \quad \Rightarrow$$
$$y_2(t) = c_0 e^{-\mathrm{i}\omega t} + \text{other terms}, \quad c_0 = \Big[Y_2(s)\cdot(s+p_0)\Big]_{s\to -\mathrm{i}\omega} = \Big[G(s)\frac{1}{s+p_0}\cdot(s+p_0)\Big]_{s\to -\mathrm{i}\omega} = \overline{G(\mathrm{i}\,\omega)}.$$

Finally, consider what happens if we put the *real* input $u_3(t) = [u_1(t)+u_2(t)]/2 = \cos(\omega\, t)$ into the system. Appealing to superposition and noting that $a\sin(x)+b\cos(x) = \sqrt{a^2+b^2}\sin(x+\psi)$ where $\psi = \text{atan2}(b,a)$,

$$\begin{aligned}
y_3(t) &= [y_1(t)+y_2(t)]/2 = (d_0 e^{\mathrm{i}\omega t} + c_0 e^{-\mathrm{i}\omega t})/2 + \text{other terms} \\
&= \{G(\mathrm{i}\,\omega)[\cos(\omega\, t) + \mathrm{i}\sin(\omega\, t)] + \overline{G(\mathrm{i}\,\omega)}[\cos(\omega\, t) - \mathrm{i}\sin(\omega\, t)]\}/2 + \text{other terms} \\
&= [G(\mathrm{i}\,\omega) + \overline{G(\mathrm{i}\,\omega)}]\cos(\omega\, t)/2 + [G(\mathrm{i}\,\omega) - \overline{G(\mathrm{i}\,\omega)}]\mathrm{i}\sin(\omega\, t)/2 + \text{other terms} \\
&= \Re\{G(\mathrm{i}\,\omega)\}\cos(\omega\, t) - \Im\{G(\mathrm{i}\,\omega)\}\sin(\omega\, t) + \text{other terms} \\
&= |G(\mathrm{i}\,\omega)|\sin[\omega\, t + \text{atan2}(\Re\{G(\mathrm{i}\,\omega)\}, -\Im\{G(\mathrm{i}\,\omega)\})] + \text{other terms} \\
&= |G(\mathrm{i}\,\omega)|\sin[\omega\, t + \frac{\pi}{2} + \angle G(\mathrm{i}\,\omega)] + \text{other terms} = |G(\mathrm{i}\,\omega)|\cos[\omega\, t + \angle G(\mathrm{i}\,\omega)] + \text{other terms}. \qquad (8.32)
\end{aligned}$$

The magnitude A and phase shift ϕ of the persistent sinusoidal component of the output $y_3(t) = A\,\cos(\omega\, t+\phi)$ as compared with the real input $u_3(t) = \cos(\omega\, t)$ are thus, again, simply the magnitude and phase of $G(\mathrm{i}\,\omega)$.

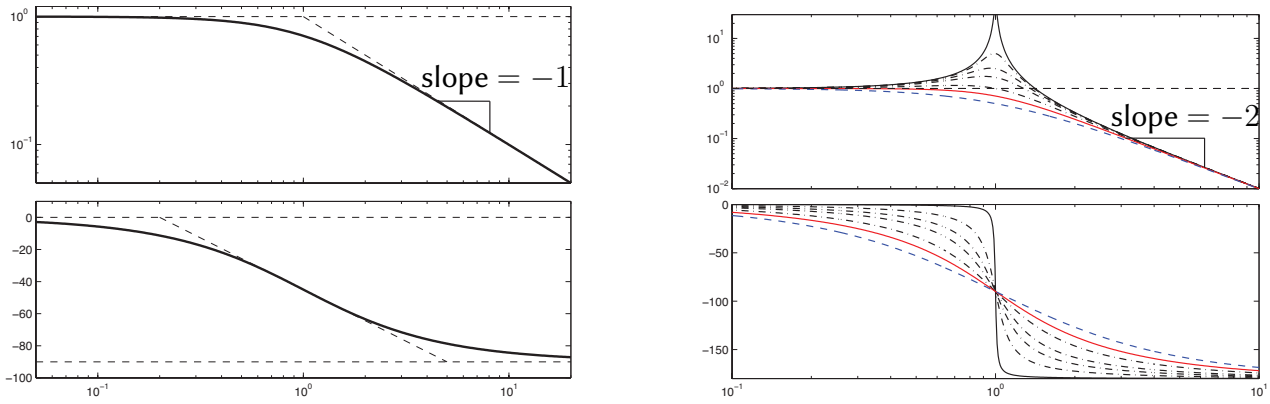


Figure 8.8: Bode plots of (left) a stable first-order low-pass filter $F_1(s) = \omega_c/[s+\omega_c]$ (with $\omega_c > 0$), and (right) a stable second-order low-pass filter $F_2(s) = \omega_c^2/[s^2 + 2\zeta\,\omega_c\,s + \omega_c^2]$ (with $0 < \zeta \leq 1$), illustrating (top) the gain, $|F(\mathrm{i}\,\omega)|$, and (bottom) the phase, $\angle F(\mathrm{i}\,\omega)$, of the filter response as a function of the normalized frequency, ω/ω_c, of a sinusoidal input. In the first-order filter, the asymptotes illustrated as dashed lines are helpful to sketch the curve, noting that the gain at $\omega/\omega_c = 1$ is 0.707 and the phase at $\omega/\omega_c = 0.2$ is $-11°$; if the system were unstable ($F(s) = \omega_c/[s-\omega_c]$), the phase would shift up by $90°$ instead of down by $90°$. In the second-order filter, plotted are curves corresponding to (solid) $\zeta = 0.01$, (dot-dashed) $\zeta = 0.1, 0.2, 0.3, 0.5$, (solid) 0.707, and (dashed) $\zeta = 1$; if the system were unstable (with $-1 < \zeta < 0$), the phase would shift up by $180°$ instead of down by $180°$. The Bode plots of systems with a first-order or second-order zero, $1/F_1(s)$ or $1/F_2(s)$, are obtained by taking the reciprocals of the gain and swapping the sign of the phase in the plots shown above.

Computing the Bode plot of unstable systems†

A Bode plot may also be developed for unstable systems. If the transfer function $G(s)$ of an unstable system is known, the process of computing its Bode plot is identical to that described above: simply calculate the magnitude and phase of $G(\mathrm{i}\,\omega)$ for the relevant range of values of ω. Note that it doesn't matter that some of the components of the partial fraction expansion of $Y(s)$ have RHP poles in this case, because we need not actually perform the experiment described above, and thus need not consider all of the terms $y(t)$.

If the transfer function $G(s)$ of an unstable system is unknown, however, this computation can not be performed, and the response of the experiment described in the first paragraph of §8.4 would be dominated by one or more exponentially-growing component(s), and thus be inconclusive. However, if we can *guess* a simple $D(s)$ [e.g., $D(s) = K$ for some K] that is adequate to stabilize the closed-loop transfer function $T(s) = G(s)\,D(s)/[1+G(s)\,D(s)]$ (see Figure 10.1 and the introduction to §10), then the Bode plot of $T(s)$ may again be determined experimentally and, since $D(s)$ is known, the magnitude and phase of $G(\mathrm{i}\,\omega)$ for the corresponding range of ω may be deduced from the corresponding relation $G(\mathrm{i}\,\omega) = T(\mathrm{i}\,\omega)/[(1-T(\mathrm{i}\,\omega))\,D(\mathrm{i}\,\omega)]$. Based on the Bode plot of $G(s)$ so determined, a better controller $D(s)$ may then be developed using any of the several constructive techniques developed in §10.

Sketching Bode plots of real systems by hand

The Bode plot, together with the root locus plot of §10.2.1, are two essential tools for classical feedback control design (§10). Though quite easily plotted using a computer, it is valuable to first know how to sketch a Bode plot by hand, in order to anticipate how the Bode plot changes when a controller is modified, and to understand how to modify a controller to change a Bode plot in a desired manner.

To proceed, consider a stable or unstable real CT SISO LTI system with ℓ zeros [or $(-\ell)$ poles] at the origin, q real first-order zeros $z_i \neq 0$, r real first-order poles $p_i \neq 0$, Q pairs of complex-conjugate zeros $z^c_{i\pm}$, and R pairs of complex-conjugate poles $p^c_{i\pm}$, written in transfer function form

$$G(s) = K_o\, s^\ell \cdot \frac{(s-z_1)(s-z_2)\cdots(s-z_q)}{(s-p_1)(s-p_2)\cdots(s-p_r)} \cdot \frac{(s-z^c_{1+})(s-z^c_{1-})(s-z^c_{2+})(s-z^c_{2-})\cdots(s-z^c_{Q+})(s-z^c_{Q-})}{(s-p^c_{1+})(s-p^c_{1-})(s-p^c_{2+})(s-p^c_{2-})\cdots(s-p^c_{R+})(s-p^c_{R-})}.$$

Usually, $\ell \leq 0$; if $\ell > 0$, there are one or more *zeros*, rather than poles, at the origin. Multiplying together the factors corresponding to the pairs of complex-conjugate poles & zeros, we have

$$G(s) = K_o\, s^\ell \cdot \frac{(s-z_1)\cdots(s-z_q)}{(s-p_1)\cdots(s-p_r)} \cdot \frac{(s^2 + 2\,Z_1\,\Omega_1\,s + \Omega_1^2)\cdots(s^2 + 2\,Z_Q\,\Omega_Q\,s + \Omega_Q^2)}{(s^2 + 2\,\zeta_1\,\omega_1\,s + \omega_1^2)\cdots(s^2 + 2\,\zeta_R\,\omega_r\,s + \omega_R^2)}, \tag{8.33}$$

where $-1 \leq Z_i \leq 1$, $-1 \leq \zeta_i \leq 1$, $\Omega_i > 0$, and $\omega_i > 0$. A term in $G(s)$ is said to have **multiplicity** k if it is repeated k times. If all $p_i < 0$ and all $\zeta_i > 0$, then all poles are in the LHP and the system is stable, though the following discussion is valid even for neutrally stable or unstable systems. Evaluating (8.33) at $s = \mathrm{i}\,\omega$ gives

$$G(\mathrm{i}\,\omega) = K_o(\mathrm{i}\,\omega)^\ell \cdot \frac{(\mathrm{i}\,\omega-z_1)\cdots(\mathrm{i}\,\omega-z_q)}{(\mathrm{i}\,\omega-p_1)\cdots(\mathrm{i}\,\omega-p_r)} \cdot \frac{(-\omega^2 + 2\,Z_1\,\Omega_1\,\mathrm{i}\,\omega + \Omega_1^2)\cdots(-\omega^2 + 2\,Z_Q\,\Omega_Q\,\mathrm{i}\,\omega + \Omega_Q^2)}{(-\omega^2 + 2\,\zeta_1\,\omega_1\,\mathrm{i}\,\omega + \omega_1^2)\cdots(-\omega^2 + 2\,\zeta_R\,\omega_R\,\mathrm{i}\,\omega + \omega_R^2)}, \tag{8.34}$$

Noting that $G(\mathrm{i}\,\omega)$ above is the product of three types of terms, the Bode plot may be sketched using the following handy rules[16] (Bode 1930), as demonstrated in practice in Example 8.3:

1. For small ω, the gain and phase of the Bode plot approach the gain and phase of the following expression:

$$G(\mathrm{i}\,\omega) \approx (\mathrm{i}\,\omega)^\ell K_o[(-z_1)\,(-z_2)\cdots(-z_q)\cdot\Omega_1^2\,\Omega_2^2\cdots\Omega_Q^2]/[(-p_1)\,(-p_2)\cdots(-p_r)\cdot\omega_1^2\,\omega_2^2\cdots\omega_R^2].$$

2. The (positive) frequencies $\{|p_1|,\ldots,|p_r|;|z_1|,\ldots,|z_q|;\omega_1,\ldots,\omega_R;\Omega_1,\ldots,\Omega_Q\}$ are called the **breakpoints**. Starting from the asymptote at the far left of the gain and phase plots and working from left to right, the gain and phase components of the Bode plot change in an orderly fashion in the vicinity of each breakpoint[17]:

a. Near each 1st-order [pole or zero] of multiplicity k, the slope of the gain [decreases or increases] by k.
b. Near each 1st-order LHP [pole or zero] of multiplicity k, the phase [decreases or increases] by $k\cdot 90°$.
c. Near each 1st-order RHP [pole or zero] of multiplicity k, the phase [increases or decreases] by $k\cdot 90°$.
For 1st-order breakpoints, the phase and the slope of the gain change gradually over a range of frequencies, from an order of magnitude below to an order of magnitude above the breakpoint, as shown in Figure 8.8a.

d. Near each 2nd-order breakpoint [pole or zero] of mult. k, the gain slope [decreases or increases] by $2\cdot k$.
e. Near each 2nd-order LHP breakpoint [pole or zero] of mult. k, the phase [decreases or increases] by $k\cdot 180°$.
f. Near each 2nd-order RHP breakpoint [pole or zero] of mult. k, the phase [increases or decreases] by $k\cdot 180°$.
For 2nd-order breakpoints, the phase and the slope of the gain change gradually over a range of frequencies, from an order of magnitude below to an order of magnitude above the breakpoint, as shown in Figure 8.8b. The behavior of both curves near the breakpoint depends on the damping [ζ_i or Z_i] with small values of damping resulting in a [**resonance** or **anti-resonance**]; i.e., a response with [large or small] gain close to the breakpoint.

When sketching a Bode plot, it is useful to ignore, at first, the fact that the slope of the gain curve and the value of the phase curve change gradually over two decades around the breakpoints, and simply plot straight-line asymptotes between each breakpoint. With these asymptotes as guides, the gain and phase curves may then be sketched by rounding out the corners of these asymptotes, using Figures 8.8a-b as guides.

[16] Many texts cite the gain in terms of **decibels** (**dB**), defined as $20\cdot\log_{10}$ of the value. We avoid this convention, as integer slopes of the gain curve on log-log plots are more readily recognized. If a gain in decibels is used, all slopes are multiplied by a factor of 20.
[17] Each of the rules here is to be read twice, first using the first word in brackets, then using the last word in brackets.

Drawing the asymptotes between the breakpoints of a Bode plot, using rules 2a and 2b above while working from low frequencies to high frequencies, is in fact quite straightforward. As demonstrated in Example 8.3, the slope j of the asymptotes for any given ω between the breakpoints in the system $G(\mathrm{i}\omega)$ in (8.34) may be computed simply by assuming (even if its not true) that ω is much *smaller* than the higher-frequency breakpoints and that ω is much *larger* than the lower-frequency breakpoints, thus allowing each first-order and second-order factor in both the numerator and denominator of (8.34) to be reduced to either its first or last term as appropriate; the slope j is then given simply by the remaining power of ω in the numerator minus the remaining power of ω in the denominator. Further, in systems that are both *stable* (with no RHP poles) and *minimum phase* (with no RHP zeros; see §10.3.4.1), the corresponding phase is simply $j \cdot 90°$ (mod 360); this useful rule of thumb is referred to as **Bode's gain/phase relationship**.

Computing the Bode plot of CT systems numerically

Programming a computer to draw a Bode plot of a given transfer function $F(s)$ is trivial, and is easily done in a few lines of code[18] (see RR_bode): simply loop over several values of frequencies ω, from well below the first breakpoint to well above the last breakpoint, compute the magnitude and phase of $F(\mathrm{i}\omega)$ at each of these frequencies, and then perform a loglog plot of the former and a semilogx plot of the latter.

Give it a try!

Though the previous page might at first look a bit daunting, the process of sketching and/or computer plotting Bode plots is actually fairly straightforward, and becomes easy with practice. For example, taking $\omega_c = 10$, the first-order low-pass filter $F_1(s) = 10/(s+10)$ of Figure 8.8 may be plotted using either the built-in Matlab commands **tf** and **bode**, or the RR commands (see §A.4) RR_tf and RR_bode, as follows:

```
Fa=tf(10,[1 10]), bode(Fa)   or   Fb=RR_tf(10,[1 10]), RR_bode(Fb)
```

The behavior of the corresponding Matlab built-in and RR codes is slightly different[19], and in certain ways those in RR are more powerful. For example, the RR codebase can easily handle the following[20] (Matlab can't...):

```
syms z p, G=RR_tf(1,[1 1]), D=RR_tf([1 z],[1 p]), T=G*D/(1+G*D)
```

As a starting point, after you follow through Example 8.3 on the following page, try sketching the Bode plots of each of the following filters following the rules on the previous page, then checking in Matlab:

high-pass, **low-pass**, **band-pass**: $F_{\text{HPa}}(s) = s/(s+1)$, $F_{\text{LPa}}(s) = 100/(s+100)$, $F_{\text{BP}}(s) = F_{\text{LPa}}(s) \cdot F_{\text{HPa}}(s)$,
lag, **lead**, **lead-lag**: $F_{\text{lag}}(s) = (s+1)/(s+0.1)$, $F_{\text{lead}}(s) = 10(s+100)/(s+1000)$, $F_{\text{lead-lag}}(s) = F_{\text{lead}}(s) \cdot F_{\text{lag}}(s)$,
proportional, **integral**, **derivative**, **PID**: $F_{\text{P}}(s) = 1$, $F_{\text{I}}(s) = 1/s$, $F_{\text{D}}(s) = s$, $F_{\text{PID}}(s) = 0.01\,(s+1)(s+100)/s$,
low-pass, **high-pass**, **band-stop**: $F_{\text{LPb}}(s) = 1/(s+1)$, $F_{\text{HPb}}(s) = s/(s+100)$, $F_{\text{BS}}(s) = F_{\text{LPb}}(s) + F_{\text{HPb}}(s)$,
notch: $F_{\text{notch}}(s) = (s^2+\omega_0^2)/(s^2+\omega_0\, s/Q+\omega_0^2)$, taking $\omega_0 = 10$ for both for $Q = 0.5$ and $Q = 5$,
all-pass: $F_{\text{AP1}}(s) = (s-1)/(s+1)$, $F_{\text{AP2}}(s) = -(s-1)/(s+1)$, $F_{\text{AP3}}(s) = -(s+1)/(s-1)$.

If you've made it this far in this text, I trust you'll get the hang of this process quickly.

Computing the Bode plot of DT systems

By (8.27), a Bode plot in DT may be drawn with the same code as that used in CT, taking $z = e^{\mathrm{i}\omega h}$ rather than $s = \mathrm{i}\omega$ when evaluating the response of the transfer function at various frequencies. Note that the frequency response of a DT system is only defined up to the Nyquist frequency, and thus a Bode plot in DT should only be drawn up to the Nyquist frequency.

[18] However, note that it takes several more to make such a code reasonably user friendly!

[19] Try h=bodeplot(Fa), setoptions(h,'MagUnits','abs','MagScale','log') to get Matlab's built-in commands to drop the extra factor of 20 in the plot of the gain.

[20] Of course, you will need to insert values for z and p before making a Bode plot.

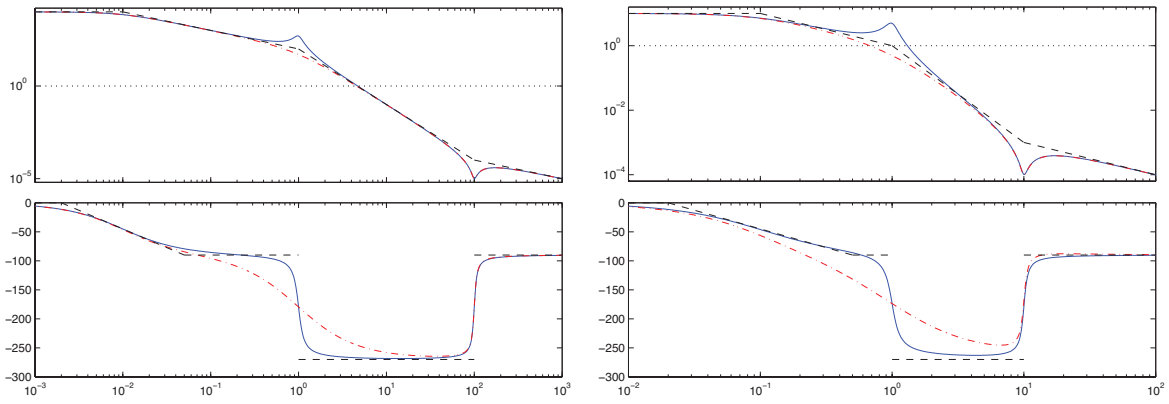


Figure 8.9: Bode plots of two more complicated systems: (left) $G_1(s)$ and (right) $G_2(s)$, as defined in (8.35).

Example 8.3 Sketching the Bode plot of representative systems.

To illustrate how to use the above-listed rules, Figure 8.9 shows the Bode plots of

$$G_1(s) = \frac{s^2 + 10s + 10000}{100\,(s + .01)(s^2 + 2\zeta s + 1)} \qquad \text{and} \qquad G_2(s) = \frac{s^2 + s + 100}{100\,(s + .1)(s^2 + 2\zeta s + 1)} \tag{8.35}$$

for $\zeta = 0.1$ (solid) and $\zeta = 1$ (dot-dashed), and effectively illustrate the process used for drawing the Bode plot of many other systems. Note that $G_1(\mathrm{i}\omega)$ has breakpoints at $\omega = .01$, 1, and 100, whereas $G_2(\mathrm{i}\omega)$ has breakpoints at $\omega = .1$, 1, and 10. In both cases, in order to sketch these Bode plots by hand, we can simply draw the asymptotes (dashed) from left to right as if these breakpoints were far apart, where $G_1(s)$ and $G_2(s)$ act like

$$G_1(\mathrm{i}\omega) = \begin{cases} 10000 & \text{for } \omega \ll .01 \\ 100/(\mathrm{i}\omega) & \text{for } .01 \ll \omega \ll 1 \\ 100/(\mathrm{i}\omega)^3 & \text{for } 1 \ll \omega \ll 100 \\ 1/(100\,\mathrm{i}\omega) & \text{for } 100 \ll \omega \end{cases} \quad \text{and} \quad G_2(\mathrm{i}\omega) = \begin{cases} 10 & \text{for } \omega \ll .1 \\ 1/(\mathrm{i}\omega) & \text{for } .1 \ll \omega \ll 1 \\ 1/(\mathrm{i}\omega)^3 & \text{for } 1 \ll \omega \ll 10 \\ 1/(100\,\mathrm{i}\omega) & \text{for } 10 \ll \omega \end{cases} \tag{8.36}$$

The asymptotes for each of these regions are easily drawn. The Bode plot is then given by "smearing out" the corners of these asymptotes, using the behavior in the vicinity of simple first-order and second-order breakpoints illustrated in Figure 8.8 as guides. Note in particular the resonance (that is, the peak in the magnitude of the Bode plot) in Figures 8.9a-b in the case with $\zeta = 0.1$, and the lack of resonance (no peak) in the case with $\zeta = 1$. Note also that the approach described above (that is, sketching the asymptotes between the breakpoints, then "smearing out" the corners in accordance with Figure 8.8) is generally effective even if, as in the $G_2(s)$ case of Figure 8.9b, the breakpoints are so close together that ω is actually never simultaneously "far" from both of the neighboring breakpoints. △

A final important point to note about Bode plots is that:

Fact 8.15 *Bode plots are additive.*

In other words, if $L(s) = G(s)\,D(s)$, and if for some ω we have $G(\mathrm{i}\omega) = R_1\,e^{\mathrm{i}\phi_1}$ and $D(\mathrm{i}\omega) = R_2\,e^{\mathrm{i}\phi_2}$, then $L(\mathrm{i}\omega) = R_3\,e^{\mathrm{i}\phi_3} = R_1\,R_2\,e^{\mathrm{i}(\phi_1+\phi_2)}$; that is, $\log(R_3) = \log(R_1) + \log(R_2)$ and $\phi_3 = \phi_1 + \phi_2$ for each ω. Thus, given log-log plots of R_1 and R_2 versus ω, and semilogx plots of ϕ_1 and ϕ_2 versus ω [i.e., given the Bode plots of $G(s)$ & $D(s)$], the corresponding log-log plot of R_3 versus ω and semilogx plot of ϕ_3 versus ω [i.e., the Bode plot of $L(s)$] is easily drawn. Alternatively, given the Bode plot of $G(s)$ and a desired *target* for $L(s)$, it is easy to determine what the Bode plot of $D(s)$ must look like. This useful fact is leveraged in the loop shaping control design technique presented in §10.2.2.

8.5 Low-pass, high-pass, band-pass, and band-stop filters

We now explore the concept of **rational CT filters**; that is, of tunable systems (usually implemented as electric circuits, as discussed in §9) which selectively "accept" and "reject" the various frequency components of a signal. The goal an **ideal filter** is to have a gain of 1 and a phase of 0 over a specified **passband** of frequencies, and a gain of nearly 0 over the remaining frequencies (referred to as the **stopband**). Unfortunately, *no rational filters ever attain this ideal*; this section discusses a few of the common families of filters available which attempt to approximate this ideal behavior.

An ideal **low-pass filter** has a passband of all signal components below some "cutoff" frequency ω_c, and a stopband of all components above this frequency. The simplest realizable low-pass filters are the first-order filter $F_1(s) = 1/[1 + (s/\omega_c)]$ depicted in Figure 8.8a and the second-order filter $F_2(s) = 1/[1 + 2\zeta(s/\omega_c) + (s/\omega_c)^2]$ depicted in Figure 8.8b (generally, $\zeta = 0.707$ is a good choice). For both filters, the gain approaches 1 and the phase approaches 0 for frequencies ω much smaller than ω_c, and the gain **rolls off** (on a log-log plot, at a slope of -1 in the first-order case and a slope of -2 in the second-order case) for frequencies much larger than ω_c. Neither filter has the ideal sharp "cutoff" at the boundary between the passband and the stopband as described above; in the remainder of this text, we thus refer to this boundary ω_c as the **corner frequency** of the corresponding filter.

The higher-order filters discussed below provide a variety of ways of achieving a sharper corner at the boundary between the passband and the stopband, at the cost of sometimes significant phase loss, even at frequencies down to an order of magnitude below the corner frequency[21].

Note that an ideal **high-pass filter** is a filter with a passband of all frequencies *above* the corner frequency ω_c, and a stopband of all frequencies *below* the corner frequency. *Any realizable low-pass filter may be converted into a high pass filter simply by replacing* (s/ω_c) *with* (ω_c/s) *in its transfer function and simplifying*; we thus focus exclusively on low-pass filter design in the discussion that follows.

Note also that an ideal **band-pass filter** is a filter with a passband of all frequencies between two critical frequencies, and a stopband at all other frequencies, whereas an ideal **band-stop filter** is a filter with a stopband of all frequencies between two critical frequencies, and a passband at all other frequencies. *A band-pass filter may be constructed from a low-pass filter and a high-pass filter connected in series, whereas a band-stop filter may be constructed from a low-pass filter and a high-pass filter connected in parallel.*

8.5.1 Maximal flatness filters: Butterworth and Bessel

Recalling Figure B.1b and defining r_k for $k = 1, \ldots, 2n$ as the $(2n)$'th roots of -1, a **Butterworth filter** is defined by a transfer function with poles given by those values of $p_k \triangleq (\mathrm{i} r_k)$ which have a negative real part:

$$F_n^{Bu}(\overline{s}) = \frac{1}{B_n(\overline{s})} \quad \text{where} \quad \overline{s} = s/\omega_c, \quad B_n(x) = \prod_{k=1}^{n}(x - p_k), \quad \text{and} \quad p_k = e^{\mathrm{i}\,\pi(2k-1+n)/(2n)}, \tag{8.37}$$

as conveniently implemented in RR_LPF_butterworth. Noting that the complex poles come in complex-conjugate pairs, $B_n(x)$ may be written with real coefficients by grouping together those factors in the above expression with complex-conjugate poles, such as p_1 and p_n; the first eight of the resulting **normalized Butterworth**

[21] Such phase loss can have important negative consequences; as discussed in §10.2.2, loss of phase at crossover can lead to a significant loss of closed-loop system performance, and even closed-loop instability. Thus, if a low-pass filter is used to reject high-frequency measurement noise in a feedback control loop, the cutoff frequency of the low-pass filter should be placed at least an order of magnitude above the crossover frequency of the closed-loop system.

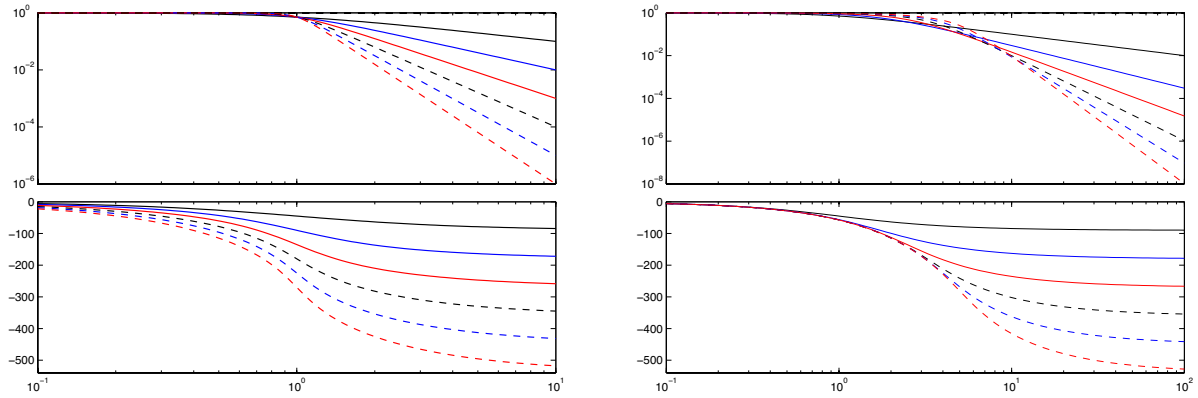


Figure 8.10: Bode plots of (solid) first-order to (dashed) sixth-order (left) Butterworth filters and (right) Bessel filters, illustrating (top) the magnitude, $|G(\mathrm{i}\,\overline{\omega})|$, and (bottom) the phase, $\angle G(\mathrm{i}\,\overline{\omega})$, of the system response as a function of the normalized frequency, $\overline{\omega} = \omega/\omega_c$, of a sinusoidal input. In the vicinity of ω_c, the Butterworth filter is optimal in terms of the flatness of the gain, whereas the Bessel filter is optimal in terms of the flatness of the group delay (that is, the phase). Note that the slope of the magnitude plot of the n'th-order filter in both cases is monotonic, and approaches $-n$ for $\overline{\omega} \gg 1$.

polynomials $B_n(x)$ are:

$$
\begin{aligned}
B_1(x) &= (x+1),\\
B_2(x) &= (x^2 + 1.41421\,x + 1),\\
B_3(x) &= (x+1)(x^2 + x + 1),\\
B_4(x) &= (x^2 + 0.76537\,x + 1)(x^2 + 1.84776\,x + 1),\\
B_5(x) &= (x+1)(x^2 + 0.61803\,x + 1)(x^2 + 1.61803\,x + 1),\\
B_6(x) &= (x^2 + 0.51764\,x + 1)(x^2 + 1.41421\,x + 1)(x^2 + 1.93185\,x + 1),\\
B_7(x) &= (x+1)(x^2 + 0.44504\,x + 1)(x^2 + 1.24698\,x + 1)(x^2 + 1.80194\,x + 1),\\
B_8(x) &= (x^2 + 0.39018\,x + 1)(x^2 + 1.11114\,x + 1)(x^2 + 1.66294\,x + 1)(x^2 + 1.96157\,x + 1).
\end{aligned}
$$

Bode plots of the first 6 Butterworth filters are given in Figure 8.10a. For sinusoidal inputs at normalized frequency $\overline{\omega} = \omega/\omega_c$, the gain of the n'th-order Butterworth filter is given by the square root of

$$
|F_n^{Bu}(\mathrm{i}\,\overline{\omega})|^2 = F_n^{Bu}(\mathrm{i}\,\overline{\omega})\,F_n(-\mathrm{i}\,\overline{\omega}) = \frac{1}{\prod_{k=1}^{n}[(\mathrm{i}\,\overline{\omega}) - p_k]\,\prod_{k=1}^{n}[(-\mathrm{i}\,\overline{\omega}) - p_k]} = \frac{1}{\overline{\omega}^{2n}+1},
$$

where the expression on the right follows simply because $\{\mathrm{i}\,p_1, \ldots, \mathrm{i}\,p_n, -\mathrm{i}\,p_1, \ldots, -\mathrm{i}\,p_n\}$ is the set of all roots of the equation $\overline{\omega}^{2n} + 1 = 0$. Defining the gain $G_n^{Bu}(\overline{\omega}) = 1/(\overline{\omega}^{2n}+1)^{1/2}$, it follows that

$$
\frac{dG_n^{Bu}(\overline{\omega})}{d\overline{\omega}} = -n\left[G_n^{Bu}(\overline{\omega})\right]^3 \overline{\omega}^{2n-1} < 0 \quad \text{and} \quad G_n^{Bu}(\overline{\omega}) = 1 - \frac{1}{2}\,\overline{\omega}^{2n} + \frac{3}{8}\,\overline{\omega}^{4n} + \ldots; \tag{8.38}
$$

that is, $G_n^{Bu}(\overline{\omega})$ decreases monotonically with $\overline{\omega}$, and the first $(2n-1)$ derivatives of $G_n^{Bu}(\overline{\omega})$ evaluated at $\overline{\omega} = 0$ are zero; this property is referred to as **maximal flatness of the gain curve**, and is the central strength of the Butterworth filter. Unfortunately, as seen in Figure 8.10a, the higher-order Butterworth filters with sharp rolloff for $\overline{\omega} > 1$ suffer from significant phase loss over a large range of frequencies below ω_c.

A **Bessel filter** is defined by the transfer function

$$
F_n^{Be}(\overline{s}) = \frac{\theta_n(0)}{\theta_n(\overline{s})} \quad \text{where} \quad \overline{s} = s/\omega_c \quad \text{and} \quad \theta_n(x) = \sum_{k=0}^{n} \frac{(2n-k)!}{(n-k)!\,k!}\,\frac{x^k}{2^{n-k}}, \tag{8.39}
$$

as conveniently implemented in RR_LPF_bessel; the $\theta_n(x)$ functions are known as **reverse Bessel polynomials**, the first eight of which are given by

$$\begin{aligned}
\theta_1(x) &= x + 1, \\
\theta_2(x) &= x^2 + 3\,x + 3, \\
\theta_3(x) &= x^3 + 6\,x^2 + 15\,x + 15, \\
\theta_4(x) &= x^4 + 10\,x^3 + 45\,x^2 + 105\,x + 105, \\
\theta_5(x) &= x^5 + 15\,x^4 + 105\,x^3 + 420\,x^2 + 945\,x + 945, \\
\theta_6(x) &= x^6 + 21\,x^5 + 210\,x^4 + 1260\,x^3 + 4725\,x^2 + 10395\,x + 10395, \\
\theta_7(x) &= x^7 + 28\,x^6 + 378\,x^5 + 3150\,x^4 + 17325\,x^3 + 62370\,x^2 + 135135\,x + 135135, \\
\theta_8(x) &= x^8 + 36\,x^7 + 630\,x^6 + 6930\,x^5 + 51975\,x^4 + 270270\,x^3 + 945945\,x^2 + 2027025\,x + 2027025.
\end{aligned}$$

Bode plots of the first 6 Bessel filters are given in Figure 8.10b. Defining $\overline{\omega} = \omega/\omega_c$ as before and (for F_6^{Be}) the phase $\phi_6^{Be}(\overline{\omega}) = -\text{atan}\left[(21\,\overline{\omega}^5 - 1260\,\overline{\omega}^3 + 10395\,\overline{\omega})/(-\overline{\omega}^6 + 210\,\overline{\omega}^4 - 4725\,\overline{\omega}^2 + 10395)\right]$, it follows that

$$\frac{d\phi_6^{Be}(\overline{\omega})}{d\overline{\omega}} = -\frac{21\,\overline{\omega}^{10} + 630\,\overline{\omega}^8 + 18900\,\overline{\omega}^6 + 496125\,\overline{\omega}^4 + 9823275\,\overline{\omega}^2 + 108056025}{\overline{\omega}^{12} + 21\,\overline{\omega}^{10} + 630\,\overline{\omega}^8 + 18900\,\overline{\omega}^6 + 496125\,\overline{\omega}^4 + 9823275\,\overline{\omega}^2 + 108056025} < 0 \tag{8.40a}$$

$$= 1 - \frac{\overline{\omega}^{12}}{108056025} + \frac{\overline{\omega}^{14}}{1188616275} + O(\overline{\omega}^{16}); \tag{8.40b}$$

that is, $\phi_6^{Be}(\overline{\omega})$ decreases monotonically with $\overline{\omega}$, and the first 11 derivatives of the **group delay** $D_6^{Be}(\overline{\omega}) \triangleq -d\phi_6^{Be}/d\overline{\omega}$ evaluated at $\overline{\omega} = 0$ are zero; this property is referred to as **maximal flatness of the group delay curve**, and is the central strength of the Bessel filter. [To verify the correctness of (8.40), as well as to confirm that the group delay of Bessel filters at other orders are similarly flat, see the Exercises.] Unfortunately, as seen in Figure 8.10b, Bessel filters have significantly less attenuation at any given frequency $\overline{\omega} > 1$ than do the corresponding Butterworth filters at the same order; although the both $|F_n^{Bu}(\text{i}\overline{\omega})|$ and $|F_n^{Be}(\text{i}\overline{\omega})|$ eventually roll-off at slope $-n$ on a log-log plot versus $\overline{\omega}$ for $\overline{\omega} \gg 1$, Bessel filters approach this asymptote at frequencies roughly an order of magnitude higher than do Butterworth filters at the same order.

8.5.2 Equiripple filters: Chebyshev, inverse Chebyshev, and elliptic†

The Butterworth and Bessel filter gains illustrated Figure 8.10 decrease monotonically with frequency. A comparison of these filters indicates an interesting tradeoff between the flatness of the gain in the passband, the flatness of the phase in the passband, and the rate of roll-off of the gain above the passband. Different tradeoffs between such generally competing objectives may be considered; we indulge ourselves here with a brief discussion of one additional such family, known as **equiripple filters**.

Equiripple filters are based on the **Chebyshev function** [see Figures 8.11a-c], which are defined iteratively [see §5.13 of *NR* for further discussion] such that

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_n(x) = 2\,x\,T_{n-1}(x) - T_{n-2}(x) \quad \text{for } n = 2, 3, \ldots \tag{8.41a}$$

$$\text{noting that} \quad T_n(x) = \cos[n\,\theta(x)] \quad \text{where} \quad \theta = \text{acos}\,(x), \tag{8.41b}$$

and a powerful relative of the Chebyshev function known as the **elliptic function** [see (8.44) and Figure 8.11d].

For the remainder of this subsection (only), we focus our attention on the filter gain, plotting the square of this gain on a linear plot rather than a log-log plot, as this convention illustrates well the criteria considered in equiripple filter design, as shown in Figures 8.12 and 8.13. We also define the **transition band** as the region between the passband and the stopband. In equiripple filter design (see Figures 8.12 and 8.13), one attempts to

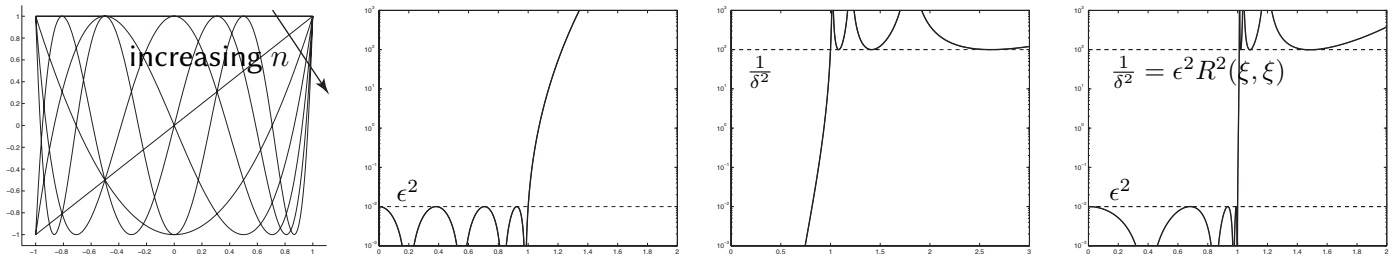


Figure 8.11: Plots of (a) the Chebyshev function T_n for $n = 0, \ldots, 6$ [see (8.41a)], and [taking $n = 8, \epsilon = \delta = 0.1$, and $\xi = 1.011$] the squared and scaled (b) Chebyshev function $f_n^C(\overline{\omega}) = \epsilon^2 T_n^2(\overline{\omega})$, (c) inverse Chebyshev function $f_n^I(\overline{\omega}) = 1/[\delta^2 T_n^2(1/\overline{\omega})]$, and (d) elliptic function $f_n^E(\overline{\omega}) = \epsilon^2 R^2(\xi, \overline{\omega})$ [see (8.44)]. The corresponding Chebyshev, inverse Chebyshev, and elliptic filters, collectively known as **equiripple filters**, are characterized by the filter gain $|F_n(\overline{\omega})|^2 = 1/(1 + f_n(\overline{\omega}))$ [see Figures 8.12-8.13].

make this transition band as narrow as possible by sacrificing the monotonic behavior of the filter gain seen in Figure 8.10. That is, equiripple filters achieve rapid roll-off in the transition band by allowing the gain to **ripple** between minimum and maximum admissible values: in particular, **Chebyshev filters** allow ripples in the passband, **inverse Chebyshev filters** allow ripples in the stopband, and the (most general) **elliptic filters** allow ripples in both the passband and the stopband. The Chebyshev and inverse Chebyshev filters are both special cases of the elliptic filter, and the Butterworth filter is a special case of all three.

Chebyshev filters

For sinusoidal inputs at normalized frequency $\overline{\omega}$, the **Chebyshev filter** $F_n^C(\overline{s}; \epsilon)$ is characterized by the gain

$$|F_n^C(\mathrm{i}\overline{\omega}; \epsilon)| = \frac{1}{\sqrt{1 + \epsilon^2 T_n^2(\overline{\omega})}} \quad \text{where} \quad \overline{s} = s/\omega_c, \;\; \overline{\omega} = \omega/\omega_c; \tag{8.42}$$

note the tunable parameter ϵ in addition to the order parameter n and corner frequency ω_c.

In order to write the Chebyshev filter in transfer function form

$$F_n^C(\overline{s}; \epsilon) = c^C \frac{1}{(\overline{s} - p_1^C)(\overline{s} - p_2^C) \cdots (\overline{s} - p_n^C)},$$

we must identify the transfer function poles p_m^C and gain c^C (the zeros of the Chebyshev filter are all at infinity). Noting (8.42) for $\overline{s} = \mathrm{i}\overline{\omega}$, and additionally noting (8.41b), the poles of $F_n^C(\overline{s}; \epsilon)$ are given by

$$1 + \epsilon^2 T_n^2(\overline{\omega}) = 1 + \epsilon^2 T_n^2(\cos\theta) = 1 + \epsilon^2 \cos^2(n\theta) = 0 \quad \text{where} \quad \overline{\omega} = -\mathrm{i}\overline{s} \triangleq \cos\theta,$$

and thus the (stable) transfer function poles (with negative real part) may be written

$$p_m^C = \mathrm{i}\cos(\theta_m) \quad \text{where} \quad \theta_m = \frac{1}{n}\mathrm{acos}\,\frac{\mathrm{i}}{\epsilon} + \frac{m\pi}{n} \quad \text{for} \quad m = 0, \ldots, n-1.$$

The transfer function gain is given simply by $c^C = \prod p_m^C = 1/(\epsilon 2^{n-1})$.

These equations are implemented in RR_LPF_chebyshev and visualized in Figures 8.12b and 8.13b.

Inverse Chebyshev filters

For sinusoidal inputs at normalized frequency $\overline{\omega}$, the **inverse Chebyshev filter** $F_n^I(\overline{s}; \delta)$ is characterized by

$$|F_n^I(\mathrm{i}\overline{\omega}; \delta)| = \frac{1}{\sqrt{1 + 1/[\delta^2 T_n^2(1/\overline{\omega})]}} \quad \text{where} \quad \overline{s} = s/\omega_c, \;\; \overline{\omega} = \omega/\omega_c; \tag{8.43}$$

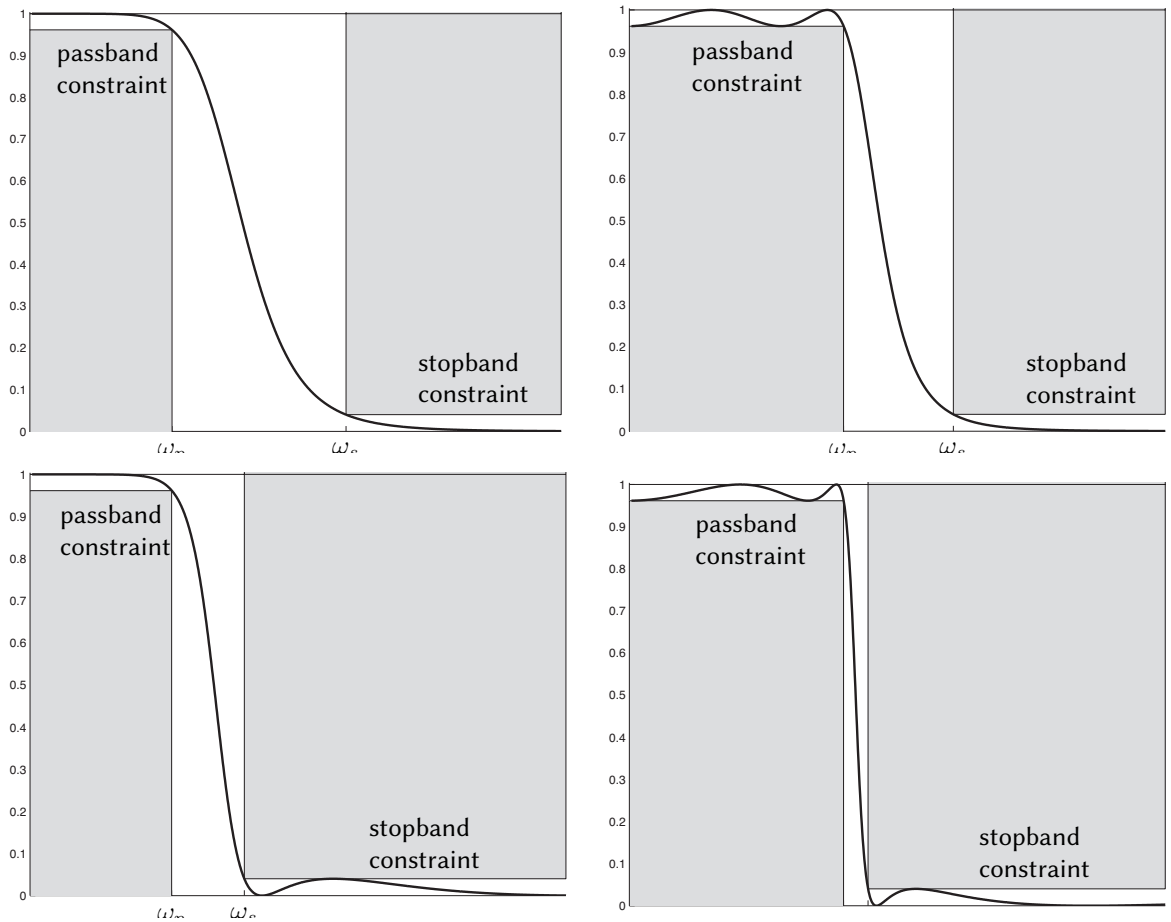


Figure 8.12: Linear plots of the square of the filter gain of the (a) Butterworth, (b) Chebyshev, (c) Inverse Chebyshev, and (d) Elliptic filters, along with the criteria used for the equiripple filter designs with $\epsilon = \delta = 0.2$ and order $n = 4$. The interval $(0, \omega_p)$ is referred to as the **passband**, where the square of the filter gain is constrained to lie between $1/(1+\epsilon^2)$ and 1, whereas the interval (ω_s, ∞) is referred to as the **stopband**, where the square of the filter gain is constrained to lie between 0 and δ^2. The interval (ω_p, ω_s) is referred to as the **transition band**. By allowing small ripples in the gain in the passband (Chebyshev), stopband (inverse Chebyshev), or both (elliptic), the width of the transition band is substantially reduced as compared with the nonrippled (Butterworth) case at a given order n.

note the tunable parameter δ in addition to the order parameter n and corner frequency ω_c.

In order to write the inverse Chebyshev filter in transfer function form

$$F_n^I(\overline{s};\epsilon) = c^I \frac{(\overline{s}-z_1^I)(\overline{s}-z_2^I)\cdots(\overline{s}-z_n^I)}{(\overline{s}-p_1^I)(\overline{s}-p_2^I)\cdots(\overline{s}-p_n^I)},$$

we must identify the transfer function zeros z_m^I, poles p_m^I, and gain c^I. Noting (8.43) for $\overline{s} = -\mathrm{i}\,\overline{\omega}$, and additionally noting (8.41b), the poles of $F_n^I(\overline{s};\epsilon)$ are given by

$$1 + \frac{1}{\delta^2 T_n^2(1/\overline{\omega})} = 1 + \frac{1}{\delta^2 T_n^2(\cos\theta)} = 1 + \frac{1}{\delta^2\cos^2(n\theta)} = 0 \quad \text{where} \quad \frac{1}{\overline{\omega}} = \frac{1}{\mathrm{i}\overline{s}} \triangleq \cos\theta,$$

and thus the (stable) transfer function poles (with negative real part) may be written

$$p_m^I = \frac{-\mathrm{i}}{\cos\theta_m} \quad \text{where} \quad \theta_m = \frac{1}{n}\mathrm{acos}\,\frac{\mathrm{i}}{\delta} + \frac{m\pi}{n} \quad \text{for} \quad m = 0,\ldots,n-1.$$

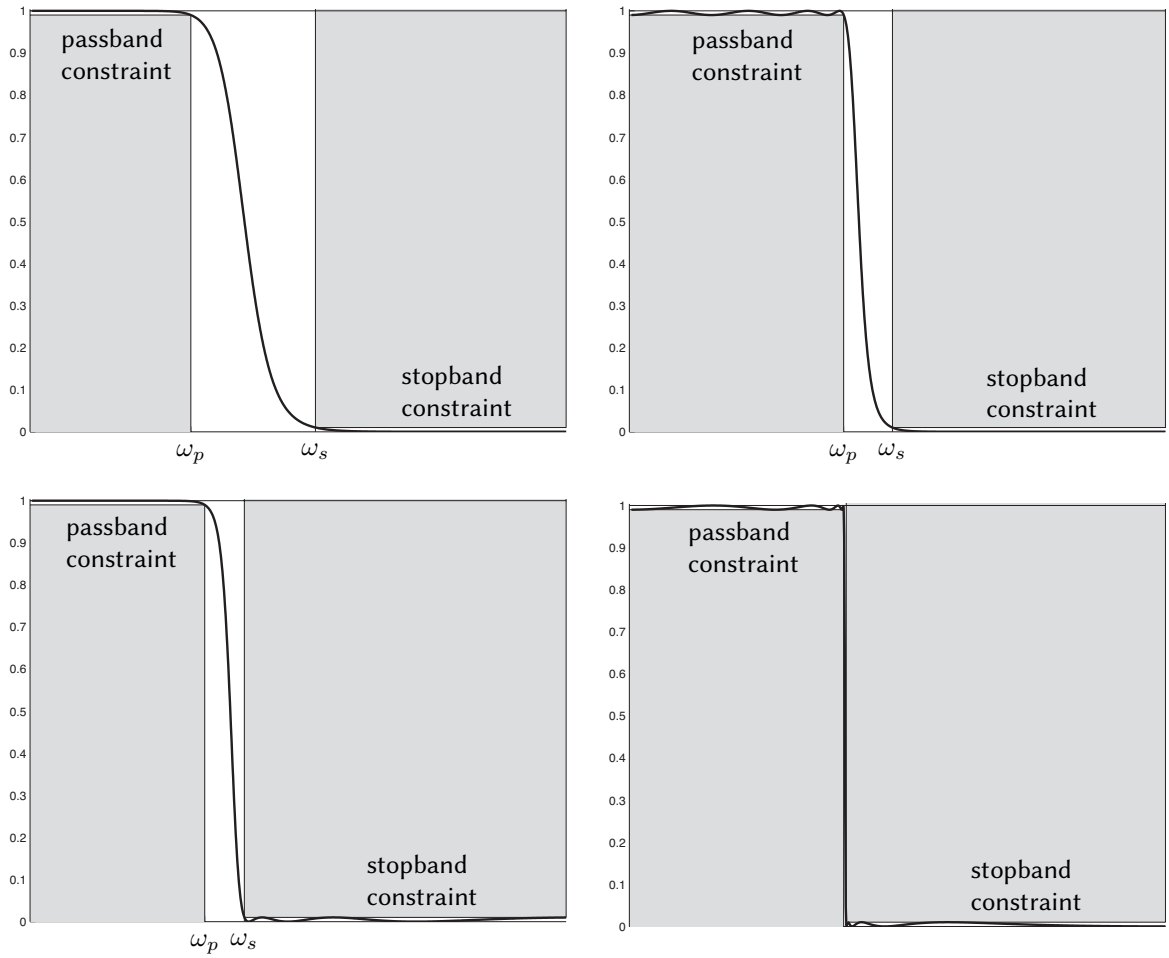


Figure 8.13: Linear plots of the square of the filter gain of the (a) Butterworth, (b) Chebyshev, (c) Inverse Chebyshev, and (d) Elliptic filters with $\epsilon = \delta = 0.1$ and order $n = 8$ (cf. Figure 8.12).

By (8.43), the transfer function zeros are simply the inverse of the zeros of the Chebyshev polynomial:

$$T_n\Big(\frac{1}{\overline{\omega}}\Big) = \cos(n\phi) = 0 \quad \text{where} \quad \frac{1}{\overline{\omega}} \triangleq \cos\phi \quad \Rightarrow \quad z_m^I = \frac{\mathrm{i}}{\cos\phi_m}, \quad \phi_m = \frac{(2m-1)\pi}{2n} \quad \text{for} \quad m = 1, \ldots, n.$$

The transfer function gain is given by $c^I = \prod p_m^I / \prod z_m^I$.

These equations are implemented in RR_LPF_inv_chebyshev and visualized in Figures 8.12c and 8.13c.

Elliptic filters

The **elliptic filter** (a.k.a. **Cauer filter**) $F_n^E(\overline{s}; \epsilon, \xi)$ is a remarkably flexible filter design characterized, for sinusoidal inputs at normalized frequency $\overline{\omega}$, by the gain function

$$|F_n^E(\mathrm{i}\overline{\omega}; \epsilon, \xi)| = \frac{1}{\sqrt{1 + \epsilon^2 R_n^2(\xi, \overline{\omega})}} \quad \text{where} \quad \overline{s} = s/\omega_c, \;\; \overline{\omega} = \omega/\omega_c;$$

with tunable parameters ϵ and ξ in addition to the order parameter n and corner frequency ω_c, where $R_n(\xi, x)$ is a special function known as the **elliptic rational function** (a.k.a. **Chebyshev rational function**), which is normalized such that $R_n(\xi, 1) = 1$. A complete exposition of the elliptic rational function for all orders n is

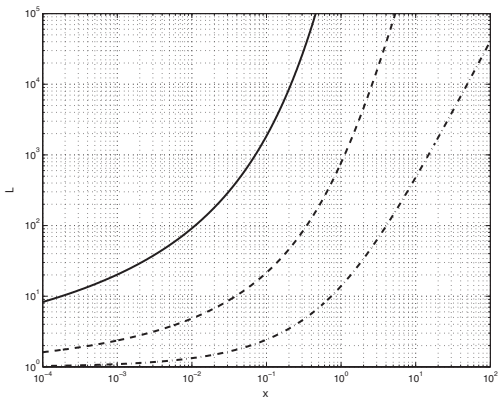


Figure 8.14: The factor $L_n(\xi)$ of elliptic filter design for (solid) $n = 8$, (dashed) $n = 4$, (dot-dashed) $n = 2$.

quite involved and a bit peripheral to the present discussion[22]; suffice it to note here that the elliptic rational function may be defined for order $n = 2^s$ for integer s via the recursive **nesting property**

$$R_{m\cdot p}(\xi, x) = R_m(R_p(\xi, \xi), R_p(\xi, x)) \quad \text{where} \quad R_2(\xi, x) = \frac{(t+1)x^2 - 1}{(t-1)x^2 + 1} \quad \text{and} \quad R_1(\xi, x) = x, \tag{8.44a}$$

and, defining the **discrimination factor** $L_n(\xi) \triangleq R_n(\xi, \xi)$ [see Figure 8.14], the factor t defined according to

$$t_m(\xi) \triangleq \sqrt{1 - 1/L_m^2(\xi)} \quad \text{and} \quad t \triangleq t_1(\xi) = \sqrt{1 - 1/\xi^2}. \tag{8.44b}$$

In order to write the elliptic filter in transfer function form

$$F_n^E(\overline{s}; \epsilon, \xi) = c^E \frac{(\overline{s} - z_1^E)(\overline{s} - z_2^E) \cdots (\overline{s} - z_n^E)}{(\overline{s} - p_1^E)(\overline{s} - p_2^E) \cdots (\overline{s} - p_n^E)},$$

we must identify the transfer function zeros z_m^E, poles p_m^E, and gain c^E. The zeros z_m^E of the elliptic filter $F_n^E(\overline{s}; \epsilon, \xi)$ are i times the poles p_m^R of the elliptic rational function, which may be written in the form

$$R_n(\xi, x) = c^R \frac{(x - z_1^R)(x - z_2^R) \cdots (x - z_n^R)}{(x - p_1^R)(x - p_2^R) \cdots (x - p_n^R)}.$$

The poles p_m^R of the elliptic rational function, in turn, are given by the reciprocal of the zeros z_m^R of the elliptic rational function, scaled by ξ, according to the **inversion relationship**

$$R_n(\xi, \xi/x) = \frac{R_n(\xi, \xi)}{R_n(\xi, x)} \quad \Rightarrow \quad p_m^R z_m^R = \xi \quad \Rightarrow \quad z_m^E = \mathrm{i}\xi / z_m^R.$$

For $n = 2^s$, the zeros of $R_n(\xi, x)$, $z_m^R \triangleq z_m^{R,1}$ for $m = 1, \ldots, n$, may be determined by initializing $z^{R,n} = 0$ and iterating

$$\left[z_m^{R,2^r} = 1 \Big/ \sqrt{1 + t_{2^r} \frac{1 - z_m^{R,2^{r+1}}}{1 + z_m^{R,2^{r+1}}}} \quad \text{and} \quad z_{m+2^{s-1-r}}^{R,2^r} = -z_m^{R,2^r} \quad \text{for } m = 1, \ldots, 2^{s-1-r} \right] \quad \text{for } r = s-1, \ldots, 0.$$

The poles p_m^E of the elliptic filter $F_n^E(\overline{s}; \epsilon, \xi)$ are given by $p_m^E = (a_m + \mathrm{i}b_m)/c_m$ for $m = 1, \ldots, n$ where

$$a_m = -\zeta_n \sqrt{1 - \zeta_n^2} \sqrt{1 - (z_m^R)^2} \sqrt{1 - (z_m^R)^2/\xi^2}, \quad b_m = z_m^R \sqrt{1 - \zeta_n^2 (1 - 1/\xi^2)}, \quad c_m = 1 - \zeta_n^2 \Big(1 - \frac{(z_m^R)^2}{\xi^2}\Big),$$

[22] The interested reader is referred to Lutovac (2001) for a comprehensive discussion of elliptic rational functions at other orders.

where the ζ_n function may be found for $n = 2^s$ via the recursive formula

$$\zeta_n(\xi,\epsilon) = \zeta_2\left(\xi, \sqrt{\frac{1}{\zeta_{n/2}^2(L_2(\xi),\epsilon)} - 1}\right) \quad \text{with} \quad \zeta_2(\xi,\epsilon) = \frac{2}{(1+t)\sqrt{1+\epsilon^2} + \sqrt{(1-t)^2 + \epsilon^2(1+t)^2}}.$$

The transfer function gain is given by

$$c^E = \frac{1}{\sqrt{1+\epsilon^2}} \frac{\prod p_m^E}{\prod z_m^E}.$$

These equations are implemented in RR_LPF_elliptic and visualized in Figures 8.12d and 8.13d. Given constraints on ϵ and δ and a choice for n, the necessary value for ξ may be calculated via the discrimination factor (see Figure 8.14), using a bisection search (see §3.1.2 of *NR*) to find that value of ξ such that $L_n(\xi) - 1/(\epsilon\delta) = 0$.

In audio applications, the sharp cutoff in the transition band of equiripple filters is sometimes useful, as long as the ripples in their response characteristics are kept sufficiently small so as to not be noticable.

Chebyshev and Elliptic filters have ripples in the amplitude response in the passband, along with corresponding ripples in the phase response. In the feedback setting, these ripples can have significant spurious effects, and thus the use of these filters is not recommended. However, Inverse Chebyshev filters only have such ripples down in the stopband, where the magnitude of the response of the filter is substantially diminished; these ripples have negligible spurious effects in the feedback setting, and thus higher-order Inverse Chebyshev filters are an attractive alternative to higher-order Butterworth and Bessel filters for feedback applications.

8.5.3 Complementary filters and audio crossovers

Complementary filters are matched pairs of low-pass and high-pass filters that add to unity across all frequencies, $F_{\text{LPF}}(\mathrm{i}\,\omega) + F_{\text{HPF}}(\mathrm{i}\,\omega) = 1$. Typical implementations use first-order filters: $F_{\text{LPF1}} = \omega_c/(s+\omega_c)$ and $F_{\text{HPF1}} = s/(s+\omega_c)$, though higher-order filters that add to unity can also be constructed, such as $F_{\text{LPF2}} = (3\,\omega_c^2\,s + \omega_c^3)/(s+\omega_c)^3$ and $F_{\text{HPF2}} = (s^3 + 3\,\omega_c\,s^2)/(s+\omega_c)^3$. Complementary filters are typically used for **sensor fusion**, when combining the measurements from two sensors that measure the same thing, but with one sensor (e.g., an accelerometer) more accurate at lower frequencies, and the other sensor (e.g., a gyro) more accurate at higher frequencies. The drawback of high-order complementary filters in such implementations [see RR_complementary_filters] is that they typically have a resonant peak[23]; this means that, for certain frequencies near the corner frequency, the measurement from once sensor is actually *subtracting* significantly from the measurement of the other, which can have the effect of amplifying measurement noise.

Crossover filters are matched pairs of low-pass and high-pass filters in which the *magnitude* of the sum of the two filters is unity across all frequencies[24], $|F_{\text{LPF}}(\mathrm{i}\,\omega) + F_{\text{HPF}}(\mathrm{i}\,\omega)| = 1$. Crossover filters are typically used in two-way audio systems, sending the bass frequencies of an audio signal to the woofers and the treble frequencies to the tweeters[25]. The best crossover filters, known as (2nd-, 4th-, and 8th-order[26]) **Linkwitz-Riley** (**LR**) filters, are just two Butterworth filters in series [see RR_linkwitz_riley_filters]. When implemented properly (using an inverter on one of the outputs in the 2nd-order case), $F_{\text{LPF,LR}}(\mathrm{i}\,\omega)$ and $F_{\text{HPF,LR}}(\mathrm{i}\,\omega)$ have the same phase (modulo $360°$) at any given frequency ω, which prevents strange nodes and antinodes from forming across an auditorium when the woofers and tweeters are placed at slightly different spatial locations on the stage.

[23] In the second-order case listed here, these resonant peaks are pretty mild, with a maximum amplification of about 30%. These resonant peaks typically get worse as the order of the complementary filter designs is increased.

[24] By relaxing the constraint that the phase shift of the sum of the filters be zero across all frequencies, which is perhaps unimportant in audio systems, the (undesirable) resonant peak of higher-order complementary filters can be eliminated.

[25] In high-performance multi-way audio systems, **midrange** speakers and **subwoofers** are also used, implementing multiple pairs of audio crossover filters to divide up the signal appropriately.

[26] 1st, 2nd, 4th, and 8th order filters are commonly referred to in the audio setting as 6, 12, 24, and 48 dB per octave, respectively.

Chapter 9

Circuits

Contents

9.1 Introduction

9.1.1 Length, mass, and time in the SI system

From basic mechanics, the reader should already be familiar with the fundamental units of {length, mass, time} as {meter (m), kilogram (kg), second (s)} in the International System of Units (SI, the modern form of the metric system), as well as several derived units, such as force (newton, N = kg m/s^2), pressure (pascal, Pa = kg/m/s^2), energy or work (joule, J = N m), power (watt, W = J/s), frequency (hertz, Hz = 1/s), speed (m/s), acceleration (m/s^2), angular velocity (rad/s) and acceleration (rad/s^2), momentum (N s), angular momentum (N m s), torque (N m), etc. Recall also the usual prefixes of the SI system listed in Table 1.1.

9.1.2 Electric charge, energy, power, and potential in the SI system

The SI units of the various quantities encountered in electric circuits is now summarized:

- **Charge** is denoted q. The fundamental unit of charge is that of an electron; the (negative) charge of 6.2415×10^{18} electrons is called a **coulomb** (C), which is the SI unit for charge.
- Electric charge passing a given point per unit time is called **current**. The cufent at any instant is denoted $I = dq/dt$. The SI unit for current is the **ampere** (A, a.k.a. **amp**), which is a flow of 1 C/s.
- **Energy** (aka **work**) is denoted w, and the SI unit for (mechanical or electrical) energy is the **joule** (J). In mechanical terms, a joule of energy is 1 kg m^2/s^2, which may be interpreted as 1 N m when applying a force to a mass over a distance, or as 0.2390 calories of thermal energy, where 1 **calorie** (cal) is the amount of thermal energy it takes to warm 1 g (that is, 1 mL, or 1 cm^3) of water by 1°C at standard atmospheric conditions[1]. Electric energy, also measured in joules, is the electric equivalent, as electrical energy can easily be converted to heat, or to mechanical energy (to apply a force over a distance) plus heat.
- **Power** is the rate of change of energy at any instant (that is, energy is the integral of power over time), and is denoted $P = dw/dt$; the SI unit for power is the **watt** (W), which is 1 J / s. In mechanical terms, a watt of power is 1 kg m^2 / s^3, which may be interpreted as 1 N m / s when applying a force to a mass moving at a certain speed, or as 0.2390 cal / s when warming a material.
- In an electric circuit, associated with any electron is its potential to do work[2] relative to some convenient (yet, arbitrarily-defined) base state, called the **ground** state. This definition is analogous to the gravitational potential energy associated with any mass at any given height relative to an (arbitrarily-defined) gravitational ground state. The **potential** of a charge to do work, also called the **voltage** of this charge, is denoted $V = dw/dq$, and is defined analogously, relative to an (arbitrarily-defined) electrical ground state. The SI unit for potential is the **volt** (V), which is 1 J / C.

Via the above definitions and the chain rule for differentiation, it follows immediately that

$$P = \frac{dw}{dt} = \frac{dw}{dq}\frac{dq}{dt} \quad \Rightarrow \quad \boxed{P = VI} \tag{9.1}$$

Current may be envisioned as a flow of electrons, as described above; however, by convention, the (positive) direction of the current is defined as the direction *opposite* to the flow of electrons. This is known as the **passive sign convention**. Using this (at first, somewhat peculiar[3]) convention, when considering the voltage V across a device and the current I through a device, multiplying V times I as suggested by (9.1) results in

- *positive* power P if the device *absorbs* electric power from the rest of the circuit, as in a resistor[4], with current flowing from *higher* voltage to *lower* voltage, and
- *negative* power P if the device *delivers* electric power to the rest of the circuit, as in a battery, with current flowing from *lower* voltage to *higher* voltage.

As a departure from the SI convention, on the electric grid of a city, energy is usually billed in **kilowatt hours** (kW h) instead of megajoules (MJ); note that 1 kW h = 3.6 MJ. Similarly, battery charge is usually measured as **milliamp hours** (mA h) instead of coulombs (C); note that 1 mA h = 3.6 C.

[1] A Calorie (with a capital C), of food, is the amount of thermal energy it takes to warm 1 kg (that is, 1 L) of water by by 1°C.

[2] As an example, consider two identical metal spheres, one with an excess of electrons (said to be of lower voltage), and one with a depletion of electrons (said to be of higher voltage). If a resistor is connected between the two spheres, the excess repulsive force between the electrons on the first sphere tends to push electrons through the resistor and onto the second sphere until a balanced distribution of electrons is reached. In the process, the electrons being pushed through the resistor do work, generating heat.

[3] The reason for this peculiar convention is that the fundamental charge associated with an electron is defined as being *negative*.

[4] The power absorbed may be converted into **heat**, as in a resistor, a combination of **heat & electromagnetic radiation**, as in a lightbulb, laser, or RF transmitter, a combination of **heat & mechanical power**, as in a motor, fluid pump, or speaker coil, etc., or it may alternatively be *stored* (and, later, released), as in a capacitor or inductor (see §9.1.3.1), a rechargeable battery, a flywheel, etc.

The change of energy of a single electron if it is moved across a potential difference of one volt is defined as an **electron volt** (eV), and is given by $1/(6.2415 \times 10^{18}) = 1.6022 \times 10^{-19}$ J. Note that the energy E of a **photon** is given by $E = hc/\lambda$, where **Planck's constant** $h = 6.626 \times 10^{-34}$ J s and the **speed of light** $c = 2.99792 \times 10^{8}$ m / s; thus, if a single electron moves across a 1.91 V potential difference, then releases its excess energy as a photon, the resulting photon has wavelength $\lambda = 650$ nm, and is thus red in color.

9.1.3 Fundamental analog circuit elements

9.1.3.1 Resistors, capacitors, & inductors

Idealized current-voltage relationships for three common components used in analog circuits are

resistor (denoted ⎓): $\boxed{V = R\,I} \quad \Rightarrow \quad P_{\mathsf{R}} = V I = I^2\,R = V^2/R \geq 0,$ (9.2a)

capacitor (denoted ⎓): $\boxed{I = C\,dV/dt} \quad \Rightarrow \quad P_{\mathsf{C}} = V I = (C/2)\,dV^2/dt = dw_C/dt,$ (9.2b)

inductor (denoted ⎓): $\boxed{V = L\,dI/dt} \quad \Rightarrow \quad P_{\mathsf{L}} = V I = (L/2)\,dI^2/dt = dw_L/dt,$ (9.2c)

where $w_C = C\,V^2/2$ and $w_L = L\,I^2/2$. Approximate values of R, C, and L for such devices are identified with **color bands** or **numerical codes**, the interpretation of which are easy to find online. Note that:

- The SI unit for resistance R is the **ohm** (Ω) [thus, (9.2a) is known as **Ohm's law**]; an ohm is 1 V / A.
- The SI unit for capacitance C is the **farad** (F); a farad is 1 A s / V.
- The SI unit for inductance L is the **henry** (H); a henry is 1 V s / A.

When operating, electric power is always *absorbed* (i.e., dissipated as heat) by a resistor, but at any instant may either be *absorbed from* or *delivered to* the rest of the circuit by a capacitor or inductor [see (9.2)].

The idealized linear models listed above are accurate only for sufficiently small V and I inside what are known as the **rated limits** of the corresponding device; outside these limits, nonlinearities become significant (and, far outside these limits, the corresponding device will fail).

The flow of electrons along a metal wire, like that of water through a pipe, is almost always[5] associated with some loss of potential per unit length (and, therefore, some resistance), as energy is lost as heat to sustain the flow when the magnetic fields generated by the flowing electrons interact with the electromagnetic fields of the atoms within the material. *Wires* are simply made from an appropriate metal, like copper, with relatively *low* (often, negligible) resistance per unit length, whereas *resistors* are made from an appropriate metal, like Nichrome (a non-magnetic alloy of nickel and chromium), that exhibits a relatively *high* resistance per unit length, with a resistance that is fairly insensitive to the inevitable temperature fluctuations caused by driving a current through a resistor. Note that, when running a large current through a resistor, the metal warms up, and thus atoms within the metal start vibrating more energetically; this generally reduces the effective resistance R of the resistor when $|I|$ is large, eventually leading to a nonlinear relationship between V and I [cf. (9.2a)].

Though they are packaged in a variety of compact geometries, *capacitors* are perhaps best visualized as two parallel metal plates with a nonconducting material, called a **dielectric**, between them. If a current is directed through a capacitor, electrons flow in one wire and accumulate on one of the plates, repelling the electrons on the other (nearby) plate, which then flow out the other wire. As electrons accumulate on the first plate and are depleted from the second, an *electric potential difference* is gradually built up, thus inhibiting the further flow of electrons; at steady state, the current through the capacitor therefore reduces to zero. The resulting (linearized) relationship between between V and I is given in (9.2b). As shown in (9.2b), the power absorbed by or released from a capacitor at any instant, P_C, is simply the rate of change of the energy,

[5] The exception to this statement is the class of materials known as **superconductors**, which at temperatures below a material-dependent critical temperature exhibit essentially **zero resistance**, due to the expulsion of magnetic fields from within the material.

$w_C = C\,V^2/2$, stored[6] in the capacitor, where V is the voltage across the capacitor, which quantifies the accumulated charge difference across its two plates. Further, if the current through the capacitor is $I = \cos(\omega t) = \sin(\omega t + \pi/2)$, then the voltage across the capacitor is $V = (1/C)\sin(\omega t)/\omega$ [the voltage "lags" behind the current by $\pi/2$, and its magnitude reduces like $1/\omega$, as the charge difference between the two sides of the capacitor takes time to accumulate; think of the *current* variation as the "cause", and the *voltage* variation as the subsequent "effect"]. The absorbed power P_C, averaged over any multiple of periods $T = 2\pi/\omega$, is exactly zero.

Conversely, *inductors* are perhaps best visualized as tightly-wound (often, toroidal) copper wire coils wrapped around an air or (better) a ferromagnetic **core**; when a current flows through the wire, a compatible magnetic field develops within this core. If a voltage is applied across an inductor, the existing magnetic field in the core, or lack thereof, exerts an *electromotive force* on the flow of electrons which initially opposes a corresponding change in the current. As a voltage difference is maintained across the inductor (which, in turn, is generated by the circuit that is connected to it), the current through the inductor, and the corresponding magnetic field, grows in response; at steady state, the voltage across the inductor reduces to zero[7]. The resulting (linearized) relationship between between V and I is given in (9.2c). As shown in (9.2c), the power absorbed by or released from an inductor at any instant, P_L, is simply the rate of change of the energy, $w_L = L\,I^2/2$, stored[7] in the inductor, where I is the current through the inductor, which quantifies the accumulated magnetic field through its core. Further, if the voltage across an inductor is $V = \cos(\omega t) = \sin(\omega t + \pi/2)$, then the current through the inductor is $I = (1/L)\sin(\omega t)/\omega$ [the current "lags" behind the voltage by $\pi/2$, and its magnitude reduces like $1/\omega$, as the magnetic field within its core takes time to accumulate; think of the *voltage* variation as the "cause", and the *current* variation as the subsequent "effect"]. The absorbed power P_L, averaged over any multiple of periods $T = 2\pi/\omega$, is exactly zero.

The prepackaged resistors, capacitors, and inductors that are commercially available are manufactured with significant variation. Resistors are commonly available with the following tolerances on their nominal resistance: $\{\pm 20\%,\ \pm 10\%,\ \pm 5\%,\ \pm 2\%,\ \pm 1\%,\ \pm 0.5\%,\ \pm 0.25\%,\ \pm 0.1\%\}$. Associated with each of these tolerance levels is a family of resistance values denoted Ex, where x is the number of resistance values per decade that are available in that family, as listed in Tables 9.1-9.6. Available resistors in, e.g., the E6 family include 1.0 kΩ, 1.5 kΩ, 2.2 kΩ, 3.3 kΩ, 4.7 kΩ, 6.8 kΩ, 10 kΩ, 15 kΩ etc. The process of converting (rounding up or down) a given resistance to a value in one of these families is, of course, easily automated (see RR_common_RLC_value.m). Note that higher-precision resistors are more expensive and less commonly stocked at PCB fabrication facilities, and should be avoided. Note also that **calibration** may be used to eliminate the error associated with the use of lower-precision (less expensive) resistors in, e.g., voltage divider circuits, as discussed in §5.7.4.

Resistors at various tolerance levels are often produced as the result of a single manufacturing process, then tested to determine their precise resistance (using, for example, the Wheatstone bridge circuit analyzed in Example 9.6). They are then binned accordingly and, of course, those resistors most closely matching the target resistance of the higher-precision class sold at a higher price. The result of this manufacturing/sorting process is that the distribution of the actual resistance of those resistors marked at, say, 2% tolerance are often **bimodal**, as those units that more accurately match target resistance values at 1% tolerance are not placed in the looser-tolerance (2%) bins. The manufacture of (more expensive) high-precision resistors is often accomplished by accurate **laser trimming** of resistors that are initially slightly below the target resistance.

[6] A mechanical spring stores and releases the energy associated with its compression; as a rough analog, a capacitor can be thought of as a sort of "spring" on the voltage, storing and releasing the energy associated with an accumulated charge, whereas an inductor can be thought of as a "spring" on the current, storing and releasing the energy associated with an accumulated magnetic field.

[7] As a mnemonic, a *capacitor has low voltage across it at high frequencies*, as electric charge doesn't have enough time build up on it, whereas an *inductor has low current through it at high frequencies*, as a compatible magnetic field doesn't have time to form.

1.0	**1.5**	**2.2**	**3.3**	**4.7**	**6.8**

Table 9.1: The 6 values per decade in the **E6** family of $\pm 20\%$ tolerance RLC components.

1.0	**1.2**	**1.5**	**1.8**	**2.2**	**2.7**	**3.3**	**3.9**	**4.7**	**5.6**	**6.8**	**8.2**

Table 9.2: The 12 values per decade in the **E12** family of $\pm 10\%$ tolerance RLC components.

1.0	**1.1**	**1.2**	**1.3**	**1.5**	**1.6**	**1.8**	**2.0**	**2.2**	**2.4**	**2.7**	**3.0**
3.3	**3.6**	**3.9**	**4.3**	**4.7**	**5.1**	**5.6**	**6.2**	**6.8**	**7.5**	**8.2**	**9.1**

Table 9.3: The 24 values per decade in the **E24** family of $\pm 5\%$ tolerance RLC components.

1.00	**1.05**	**1.10**	**1.15**	**1.21**	**1.27**	**1.33**	**1.40**	**1.47**	**1.54**	**1.62**	**1.69**	**1.78**	**1.87**	**1.96**	**2.05**
2.15	**2.26**	**2.37**	**2.49**	**2.61**	**2.74**	**2.87**	**3.01**	**3.16**	**3.32**	**3.48**	**3.65**	**3.83**	**4.02**	**4.22**	**4.42**
4.64	**4.87**	**5.11**	**5.36**	**5.62**	**5.90**	**6.19**	**6.49**	**6.81**	**7.15**	**7.50**	**7.87**	**8.25**	**8.66**	**9.09**	**9.53**

Table 9.4: The 48 values per decade in the **E48** family of $\pm 2\%$ tolerance RLC components.

1.00	**1.02**	**1.05**	**1.07**	**1.10**	**1.13**	**1.15**	**1.18**	**1.21**	**1.24**	**1.27**	**1.30**	**1.33**	**1.37**	**1.40**	**1.43**
1.47	**1.50**	**1.54**	**1.58**	**1.62**	**1.65**	**1.69**	**1.74**	**1.78**	**1.82**	**1.87**	**1.91**	**1.96**	**2.00**	**2.05**	**2.10**
2.15	**2.21**	**2.26**	**2.32**	**2.37**	**2.43**	**2.49**	**2.55**	**2.61**	**2.67**	**2.74**	**2.80**	**2.87**	**2.94**	**3.01**	**3.09**
3.16	**3.24**	**3.32**	**3.40**	**3.48**	**3.57**	**3.65**	**3.74**	**3.83**	**3.92**	**4.02**	**4.12**	**4.22**	**4.32**	**4.42**	**4.53**
4.64	**4.75**	**4.87**	**4.99**	**5.11**	**5.23**	**5.36**	**5.49**	**5.62**	**5.76**	**5.90**	**6.04**	**6.19**	**6.34**	**6.49**	**6.65**
6.81	**6.98**	**7.15**	**7.32**	**7.50**	**7.68**	**7.87**	**8.06**	**8.25**	**8.45**	**8.66**	**8.87**	**9.09**	**9.31**	**9.53**	**9.76**

Table 9.5: The 96 values per decade in the **E96** family of $\pm 1\%$ tolerance RLC components.

1.00	**1.01**	**1.02**	**1.04**	**1.05**	**1.06**	**1.07**	**1.09**	**1.10**	**1.11**	**1.13**	**1.14**	**1.15**	**1.17**	**1.18**	**1.20**
1.21	**1.23**	**1.24**	**1.26**	**1.27**	**1.29**	**1.30**	**1.32**	**1.33**	**1.35**	**1.37**	**1.38**	**1.40**	**1.42**	**1.43**	**1.45**
1.47	**1.49**	**1.50**	**1.52**	**1.54**	**1.56**	**1.58**	**1.60**	**1.62**	**1.64**	**1.65**	**1.67**	**1.69**	**1.72**	**1.74**	**1.76**
1.78	**1.80**	**1.82**	**1.84**	**1.87**	**1.89**	**1.91**	**1.93**	**1.96**	**1.98**	**2.00**	**2.03**	**2.05**	**2.08**	**2.10**	**2.13**
2.15	**2.18**	**2.21**	**2.23**	**2.26**	**2.29**	**2.32**	**2.34**	**2.37**	**2.40**	**2.43**	**2.46**	**2.49**	**2.52**	**2.55**	**2.58**
2.61	**2.64**	**2.67**	**2.71**	**2.74**	**2.77**	**2.80**	**2.84**	**2.87**	**2.91**	**2.94**	**2.98**	**3.01**	**3.05**	**3.09**	**3.12**
3.16	**3.20**	**3.24**	**3.28**	**3.32**	**3.36**	**3.40**	**3.44**	**3.48**	**3.52**	**3.57**	**3.61**	**3.65**	**3.70**	**3.74**	**3.79**
3.83	**3.88**	**3.92**	**3.97**	**4.02**	**4.07**	**4.12**	**4.17**	**4.22**	**4.27**	**4.32**	**4.37**	**4.42**	**4.48**	**4.53**	**4.59**
4.64	**4.70**	**4.75**	**4.81**	**4.87**	**4.93**	**4.99**	**5.05**	**5.11**	**5.17**	**5.23**	**5.30**	**5.36**	**5.42**	**5.49**	**5.56**
5.62	**5.69**	**5.76**	**5.83**	**5.90**	**5.97**	**6.04**	**6.12**	**6.19**	**6.26**	**6.34**	**6.42**	**6.49**	**6.57**	**6.65**	**6.73**
6.81	**6.90**	**6.98**	**7.06**	**7.15**	**7.23**	**7.32**	**7.41**	**7.50**	**7.59**	**7.68**	**7.77**	**7.87**	**7.96**	**8.06**	**8.16**
8.25	**8.35**	**8.45**	**8.56**	**8.66**	**8.76**	**8.87**	**8.98**	**9.09**	**9.20**	**9.31**	**9.42**	**9.53**	**9.65**	**9.76**	**9.88**

Table 9.6: The values in the **E192** families of $\pm 0.5\%$, $\pm 0.25\%$, and $\pm 0.1\%$ tolerance RLC components.

Capacitors are commonly available[8] from 1 pF through 10 nF in 24 capacitance values per decade (in the E24 series in Table 9.3), and from 10 nF=0.01 μF through 10 mF=10^4 μF in 6 capacitance values per decade (in the E6 series in Table 9.1). Inductors are commonly available[8] from 1 nH through 1 mH in 24 inductance values per decade (in the E24 series in Table 9.3). Capacitors and inductors in the higher ends of these ranges are both large and expensive; increased voltage ratings on capacitors, and increased current ratings on inductors, also increase their size and cost significantly.

9.1.3.2 Power sources

In order to make an electric circuit do something, of course, you need a source[9] of electric power. Such sources come in two types, voltage sources (which are most common) and current sources, either of which may drive the connected circuit in a constant or time-varying manner, and are denoted as indicated in Figure 9.1a-d.

[8] Time constants in RLC circuits (incorporating resistors, inductors, and capacitors) may be tuned by selecting the resistors in the circuit (see, e.g., Example 9.2), so a finer granularity in available values per decade is not necessary for capacitors and inductors.

[9] Note that some devices that normally act as *sources* of electric power, like rechargeable batteries, may also from time to time be used safely as *sinks* of electric power, like a capacitor. The practical distinction between a capacitor and a rechargeable battery is that a capacitor, which simply stores and releases electrons, typically loses its charge fairly quickly when not being used, whereas a battery, which stores and releases charge via internal chemical reactions, typically holds its charge for much longer periods of time.

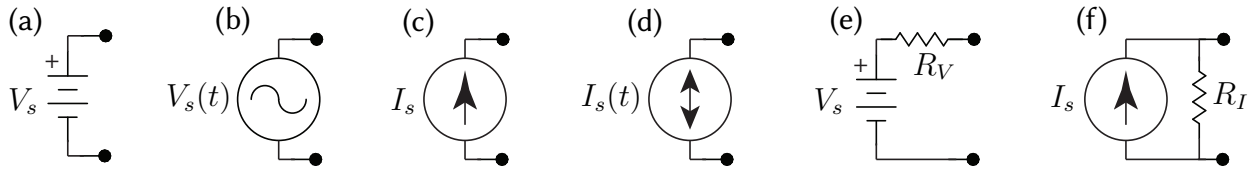


Figure 9.1: Symbols for various power sources: (a) ideal constant-voltage (direct current, or DC) source, (b) ideal time-varying voltage (alternating current, or AC) source, (c) ideal constant-current source, (d) ideal time-varying current source, (e) practical constant-voltage (DC) source, (f) practical constant-current source.

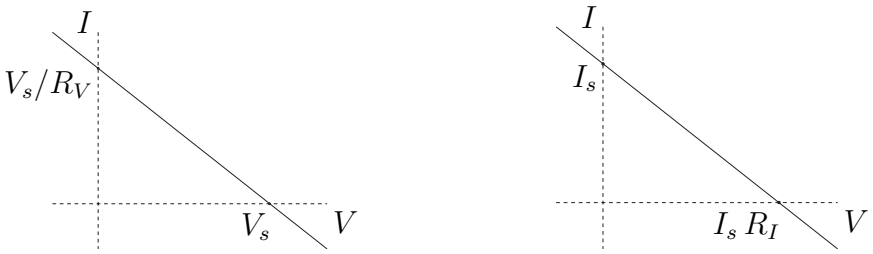


Figure 9.2: Current-voltage relationship of (left) the practical DC voltage source of Figure 9.1e (that is, a common battery), and (right) the practical current source of Figure 9.1f.

The current-voltage relationships of ideal voltage and current sources may be written

$$\textbf{ideal voltage source: } \boxed{V = V_s \text{ (regardless of } I)} \quad \textbf{ideal current source: } \boxed{I = I_s \text{ (regardless of } V)} \tag{9.2d}$$

Note that an ideal **voltage source** generates a specified *voltage* across its terminals[10] regardless of the current drawn by the rest of the circuit; an ideal voltage source can not function correctly if a wire (with zero resistance) is connected across its terminals (a.k.a. a **short circuit**), as that would cause the ideal voltage source to produce infinite current. Similarly, an ideal **current source** generates a specified *current* through the device(s) connected across its terminals regardless of the voltage required over the rest of the circuit in order to maintain it; an ideal current source can not function correctly if the circuit connected across its terminals is not closed (a.k.a. an **open circuit**), as that would cause the ideal current source to produce infinite voltage.

Despite the above-mentioned limitations, ideal voltage and current sources are reasonably good models in many situations when a circuit is properly configured. More accurate (yet still linear) models of real-world voltage and current sources are indicated in Figure 9.1e-f. In these more practical models,

- a (preferably, small) resister R_V is included in *series* with the voltage source, which thus generates a finite current of $I = V_s/R_V$, instead of an infinite current, in the case of a short circuit across its terminals, and
- a (preferably, large) resister R_I is included in *parallel* with the current source, which thus generates a finite voltage of $V = I_s\,R_I$, instead of an infinite voltage, in the case of a open circuit across its terminals.

The current-voltage relationship of the practical voltage and current sources indicated in Figures 9.1e-f are given in Figure 9.2. Note that, taking $I_s = V_s/R_V$ and $R_I = R_V$, these two relationships are identical, and thus these two sources are, consistent with the following definition, said to be[11] "**equivalent**".

Fact 9.1 (Equivalent circuit definition) *Two circuits are said to be "***equivalent***" at a specified pair of terminals if they exhibit an identical current-voltage relationship, which may be static or dynamic.*

[10] Note that a **terminal** of an electric circuit or individual circuit element is a point at which other electric circuits are intended to be attached, as denoted by black dots in Figure 9.1.

[11] Though these two practical source models are "equivalent" from the perspective of the current-voltage relationship at their terminals, they are **not at all equivalent** in terms of their internal operation:

- a practical voltage source (e.g., a common battery) expends essentially no power whatsoever if there is an *open* circuit across its terminals (since the current through, and therefore the power consumed by, its internal resistor is zero in this case), but it expends significant power if there is a *short* circuit across its terminals (since the power consumed by its internal resistor in this case is $P = V_s\,I$, where $I = V_s/R_V$ for small R_V);

9.1.3.3 Sensors & actuators for interfacing with the physical world

To connect an electric circuit to the physical world, sensors and actuators[12] are needed, as surveyed in §3. Actuators are often built from some type of **electric motor** (see §??-3.5); others include **linear actuators** (like **voice coils**), **electroactive polymers**, etc. Common sensors include **accelerometers** (**accels**) to measure linear acceleration, **gyroscopes** (**gyros**) to measure angular acceleration, **encoders** to measure wheel rotation, **thermocouples** to measure temperature, etc. Note that some actuators which convert electrical energy to mechanical energy (like motors and piezoelectric actuators[13]) can also be used as sensors or **energy scavengers** to convert mechanical energy back into electrical energy (like generators and piezoelectric sensors[13]); this concept is central to the efficient operation of hybrid and fully electric cars, in which the motor normally used to drive the wheels may be operated as a generator during regenerative braking.

9.1.4 Kirchoff's laws

A **node** of an electric circuit is defined as any point at which two or more circuit elements (and, thus, two or more current paths) are connected. In a complex electric circuit with several circuit elements and several current paths, the following two simple rules facilitate analysis:

Fact 9.2 (Kirchoff's Current Law, or KCL) *The sum of the currents entering a node equals the sum of the currents leaving that node at any instant.*

Fact 9.3 (Kirchoff's Voltage Law, or KVL) *The sum of the voltages across the elements around any closed loop in a circuit is zero at any instant.*

Note that KVL may be satisfied *by construction*, simply by keeping track of the voltage at each *node* of the circuit, rather than the voltage drop across each circuit element. Note also that, in a circuit with n nodes, there are only $(n-1)$ independent KCL equations for the currents between these nodes, as the KCL at the last node may be derived by combining appropriately the KCL relations at the other $(n-1)$ nodes.

Defining the voltage at each node (thus implicitly satisfying the KVL equations) and the current between each node, writing KCL at all but one of the nodes, and writing the current-voltage relationship across each circuit element [see, e.g., (9.2a)-(9.2d)] leads to a set of ODEs which, together with the initial conditions, completely describe the time evolution of the circuit. This is sometimes referred to as **nodal analysis** of a circuit, and is illustrated by the following examples, most of which have corresponding implementations in code in RR.ch09; please experiment with these (many!) codes when reading these examples.

Example 9.1 Voltage divider. Consider first a simple circuit formed as a series connection of two resistors, R_1 and R_2, with an ideal voltage source applied between the first node V_1 and the last node V_2, and denote by V_{mid} the voltage at the middle node. Since the current through the first resistor equals the current through the second resistor by KCL, applying (9.2a) to R_1 and R_2 leads immediately to

$$I_1 = I_2 \quad \Rightarrow \quad \frac{V_1 - V_{\text{mid}}}{R_1} = \frac{V_{\text{mid}} - V_2}{R_2} \quad \Rightarrow \quad V_{\text{mid}} = \frac{R_2\, V_1 + R_1\, V_2}{R_1 + R_2}.$$

It is seen that $V_{\text{mid}} = (V_1 + V_2)/2$ if $R_1 = R_2$, that $V_{\text{mid}} \to V_1$ if $R_1 \ll R_2$, and that $V_{\text{mid}} \to V_2$ if $R_2 \ll R_1$. △

- in contrast, a practical current source (e.g., as developed in Examples 9.19 and 9.20) expends essentially no power whatsoever if there is a *short* circuit across its terminals (since the voltage across, and therefore the power consumed by, its internal resistor is zero in this case), but it expends significant power if there is an *open* circuit across its terminals (since the power consumed by its internal resistor in this case is $P = V\, I_s$, where $V = I_s\, R_I$ for large R_I).

[12] More broadly, devices which convert one form of energy (electric, mechanical, thermal, etc.) to another are called **transducers**.

[13] That is, actuators/sensors built on materials exhibiting the **piezoelectric effect**, generating an electric field in response to applied mechanical strain, and the **reverse piezoelectric effect**, generating mechanical strain in response to an applied electric field.

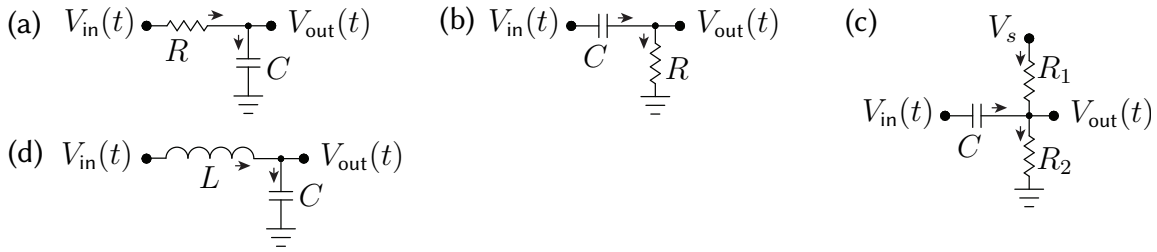


Figure 9.3: Four simple passive filters: (a) a first-order low-pass RC filter, (b) a first-order high-pass RC filter, (c) a voltage-biased first-order high-pass RC filter, and (d) a second-order low-pass LC filter.

Example 9.2 Passive low-pass and high-pass filters. Consider the circuits in Figures 9.3, assuming that

(a) the input voltage $V_{\text{in}}(t)$ is precisely specified regardless of the current drawn by the filter, and
(b) if anything else is connected to the output terminal $V_{\text{out}}(t)$, it draws negligible current [cf. Figure 9.4].

It follows (see codes in RR.ch09) that the **first-order low-pass RC filter** in Figure 9.3a is governed by

$$I_R = I_C, \quad V_{\text{in}} - V_{\text{out}} = I_R\,R, \quad I_C = C\,\frac{d\,V_{\text{out}}}{dt} \quad \Rightarrow \quad V_{\text{in}}(s) - V_{\text{out}}(s) = R\,C\,s\,V_{\text{out}}(s) \quad \Rightarrow \quad \frac{V_{\text{out}}(s)}{V_{\text{in}}(s)} = \frac{\omega_1}{s+\omega_1}$$

where $\omega_1 = 1/(R\,C)$, the **first-order high-pass RC filter** in Figure 9.3b is governed by

$$I_C = I_R, \quad I_C = C\,\frac{d[V_{\text{in}} - V_{\text{out}}]}{dt}, \quad V_{\text{out}} = I_R\,R \quad \Rightarrow \quad R\,C\,s\,[V_{\text{in}}(s) - V_{\text{out}}(s)] = V_{\text{out}}(s) \quad \Rightarrow \quad \frac{V_{\text{out}}(s)}{V_{\text{in}}(s)} = \frac{s}{s+\omega_1},$$

the **voltage-biased first-order high-pass RC filter** in Figure 9.3c is governed by

$$I_C + I_1 = I_2, \quad I_C = C\,\frac{d[V_{\text{in}} - V_{\text{out}}]}{dt}, \quad (V_s - V_{\text{out}}) = I_1\,R_1, \quad V_{\text{out}} = I_2\,R_2 \quad \Rightarrow$$

$$V_{\text{out}}(s) = \frac{R_2\,V_s}{R_1 + R_2 + R_1\,R_2\,C\,s} + \frac{R_1\,R_2\,C\,s\,V_{\text{i}}(s)}{R_1 + R_2 + R_1\,R_2\,C\,s} = \frac{R_2}{R_1+R_2}\,\frac{\omega_2}{s+\omega_2}\,V_s + \frac{s}{s+\omega_2}\,V_{\text{i}}(s)$$

where the effective resistance R_{e} is $R_{\text{e}} = R_1 R_2/(R_1 + R_2)$ [that is, $1/R_{\text{e}} = 1/R_1 + 1/R_2$] and $\omega_2 = 1/(R_{\text{e}}\,C)$, and the **undamped second-order low-pass LC filter** in Figure 9.3d is governed by

$$I_L = I_C, \quad V_{\text{in}} - V_{\text{out}} = L\,\frac{d[I_L]}{dt}, \quad I_C = C\,\frac{d\,V_{\text{out}}}{dt} \quad \Rightarrow \quad V_{\text{in}}(s) - V_{\text{out}}(s) = L\,C\,s^2\,V_{\text{out}}(s) \quad \Rightarrow \quad \frac{V_{\text{out}}(s)}{V_{\text{in}}(s)} = \frac{\omega_3^2}{s^2+\omega_3^2}$$

where $\omega_3 = 1/\sqrt{L\,C}$. Bode plots of first- and second-order low-pass filters (which are clearly quite useful for, e.g., cleaning up signals that are corrupted by high-frequency noise) are given in Figure 8.8. Unfortunately, assumptions (a) and (b) are quite restrictive: if the inputs of such **passive filters** are attached to real sensors, if they are cascaded, or if their outputs are attached to real loads (actuators, etc), one or both of these assumptions are violated, and the resulting dynamic behavior changes. To tune the dynamic behavior of filter circuits precisely when assumptions (a) and/or (b) are relaxed, we need **active filters**, as developed in §9.2.

Nevertheless, the simple low-pass and high-pass filters illustrated in Figure 9.3 are useful *building blocks* for the circuits developed in the remainder of this chapter. As noted in §9.1.3, when excited by a sinusoid (or, by a square wave, which amounts to a sum of sinusoids), the power absorbed by an ideal inductor or capacitor, averaged over a multiple of periods $T = 2\pi/\omega$, is exactly zero. Thus, in contrast with resistors (which always dissipate power), both inductors and capacitors act like "springs" of sorts, storing and releasing energy when excited sinusoidally (see Footnote 6 on page 9-4). Second-order low-pass LC filters in particular thus operate with high energetic efficiency, even though their precise dynamic behavior depends on the load applied when assumptions (a) or (b) are relaxed. Indeed, in many cyber-physical systems, the only place that you commonly see inductors being used in the associated circuits is for low-pass filters in the power regulation circuitry; elsewhere, resistors and capacitors (which are smaller and cheaper) are more commonly used instead. △

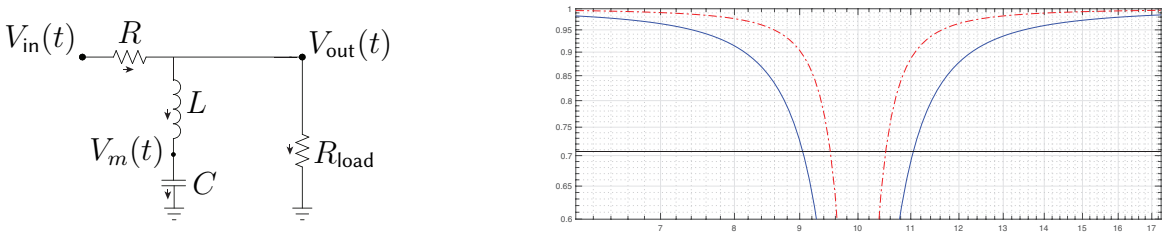


Figure 9.4: A notch filter with a resistive load $R_{load} = R/c_1$ applied, and the magnitude part of its Bode plot $V_{out}(s)/V_{in}(s)$, normalized by K, (solid) with $c_1 = 0$ and with $\{R, L, C\}$ such that $\omega_0 = 10$ and $Q = 5$, and (dot-dashed) with the same $\{R, L, C\}$, but with $c_1 = 1$ (i.e., modified R_{load}), and thus $Q = 10$. In this Bode plot, the magnitude is about 1, and the phase shift is about zero, outside the frequency range shown.

Example 9.3 A passive notch filter. As mentioned above, if the inputs of **passive filters** are attached to real sensors, if they are cascaded, or if their outputs are attached to real loads, one or both of the assumptions upon with their simplified analyses are based are violated, and the resulting dynamic behavior changes. To illustrate, consider the filter circuit with resistive load depicted in Figure 9.4, which is governed [see Algorithm 9.1] by

$$I_R = I_L + I_{load}, \quad I_C = I_L, \quad V_{in} - V_{out} = I_R\, R, \quad I_L = L\,\frac{d[V_{out} - V_m]}{dt}, \quad I_C = C\,\frac{dV_m}{dt}, \quad V_{out} = I_{load}\, R/c_1$$

$$\Rightarrow \quad \frac{V_{out}(s)}{V_{in}(s)} = \frac{L\,C\,s^2 + 1}{L\,C\,(1+c_1)\,s^2 + R\,C\,s + (1+c_1)} \triangleq K\frac{s^2+\omega_0^2}{s^2 + 2\,\zeta\,\omega_0\,s + \omega_0^2} \triangleq K\frac{s^2+\omega_0^2}{s^2 + (1/Q)\,\omega_0\,s + \omega_0^2},$$

where $\omega_0 = 1/\sqrt{L\,C}$, $K = 1/(1+c_1)$, $\zeta = R\,\sqrt{C/L}/[2\,(1+c_1)]$, $Q = 1/(2\,\zeta) = (1+c_1)\,\sqrt{L/C}/R$. The variable Q, dubbed the "quality" of this notch filter, defines the sharpness of the valley in the vicinity of the frequency ω_0 that is attenuated by the filter; such a circuit is useful for removing a tonal "buzz" from a signal.

An equivalent definition of quality is $Q = \omega_0/BW$, where BW is the range of frequencies for which the magnitude of the output is reduced by $3\,\text{dB} = 0.707$ or more, corresponding to the power of the output being reduced by 50% or more. Taking a look at the Bode plot in Figure 9.4, designed (for $c_1 = 0$) with $\{R, L, C\}$ such that $\omega_0 = 10$ and $Q = 5$, attenuation by 3 dB is seen between 9.05 rad/s and 11.05 rad/s. Thus, $\omega_0/BW = 10/(11.05 - 9.05) = 5$, which is exactly the value of Q that this second-order notch filter was designed for. △

Algorithm 9.1: Code for solving Example 9.3, first by setting up $A\mathbf{x} = \mathbf{b}$ by hand and calling $\mathbf{x} = A\backslash\mathbf{b}$, then by using an equivalent, more human-readable approach leveraging a convenient built-in "solve" algorithm.

```
clear; syms s R L C c1 Vin      % <- Laplace variable s, parameters, input Vin
% x={Vout; Va; Ir; Ic; Iload}     <- unknown vector
A  =[  1    0   R   0     0;    % Vout +R*Ir          = Vin    resistor eqn
       1   -1   0 -L*s     0;    % Vout -Va -L*s*Ic    = 0      inductor eqn [note: Ic=IL]
       0 -C*s   0   1      0;    %  -C*s*Va   +Ic      = 0      capacitor eqn
      -1    0   0   0   R/c1;    % -Vout  +Iload*R/c1  = 0      load eqn
       0    0   1  -1     -1];   %      Ir -Ic  -Iload = 0      KCL1  [KCL2 is just Ic=IL]
b  =[ Vin; 0;  0;  0;  0]; x=A\b;
F_notch1=simplify(x(1)/Vin)     % transfer fn of filter = Vout/Vin via Ax=b method.
clear; syms s R L C c1 Vin      % <- Laplace variable s, parameters, input Vin
syms Vout Va Ir Ic Iload        % <- unknown variables
eqn1= Vin-Vout == R*Ir;         % resistor eqn [written here in "easily readable" form]
eqn2=  Vout-Va == L*s*Ic;       % inductor eqn [note: Ic=IL]
eqn3=       Ic == C*s*Va;       % capacitor eqn
eqn4=     Vout == Iload*R/c1;   % load eqn
eqn5=       Ir == Ic + Iload;   % KCL1  [KCL2 is just Ic=IL]
sol=solve(eqn1,eqn2,eqn3,eqn4,eqn5,Vout,Va,Ir,Ic,Iload); % rearrange automatically, solve!
F_notch2=simplify(sol.Vout/Vin) % transfer fn of filter = Vout/Vin via "solve" method.
```

As see in the above examples, circuits with just a handful of components appear to be useful for various types of **signal conditioning** (removing high-frequency noise, removing a tonal buzz, etc), and are easily characterized by taking the Laplace transform of the set of linear equations governing them (n equations in n unknowns) and combining by hand. Such sets of linear equations are also easily converted to the form $A\mathbf{x} = \mathbf{b}$ and solved using a computer[14], leveraging a symbolic version of Gaussian elimination, as discussed in §2.2 of *NR*. One of the benefits of this approach is that it is readily apparent under what conditions the set of equations being combined cease to be independent from each other, and thus the matrix A becomes singular, which can be monitored by keeping an eye on the **condition number** of A, as discussed in §2.5 of *NR*.

As the number of components in the circuits of interest increases, the use of a computer to perform such **symbolic manipulations** becomes essential, both to save time, and to *entirely prevent* the possibility of algebra mistakes. Converting a set of linear equations to $A\mathbf{x} = \mathbf{b}$ form, though straightforward, becomes increasingly prone to transcription error as the size of the problem grows. The code given in Algorithm 9.1 thus illustrates an alternative, more human-readable approach of writing such equations in code, leveraging a built-in "solve" algorithm to do the necessary reorganizing of each equation automatically, in order to arrive at an equivalent $A\mathbf{x} = \mathbf{b}$ problem. In the case of linear systems of equations, both approaches (either setting up and solving $A\mathbf{x} = \mathbf{b}$, or writing the equations in more human-readable form and calling the automated "solve" algorithm) give the same result, and work in both Matlab and Octave. Significantly, the latter approach works for both linear and even certain nonlinear systems of equations, as shown in Example 9.4 below.

Figure 9.5: Two pairs of electrical networks with identical voltage/current relationships between their 3 nodes.

Example 9.4 Y-Δ transformations are techniques for analysis of electric circuits based on the equivalent voltage/current between each pair of nodes of two different appropriately-configured networks, one shaped like a Y (upside down, sorry) and one shaped like a Δ. In the resistor networks in Figure 9.5a-b, setting

$$R_b+R_c = 1/[1/R_1+1/(R_2+R_3)], \quad R_a+R_c = 1/[1/R_2+1/(R_1+R_3)], \quad R_a+R_b = 1/[1/R_3+1/(R_1+R_2)],$$

achieves this equivalence (see Example 9.5). These 3 relations may be rearranged (see code in RR.ch09) to determine the resistors of the Δ network in terms of those in the Y network, or vice versa, resulting in

$$R_a = \frac{R_1\,R_2}{R_1+R_2+R_3}, \qquad R_b = \frac{R_1\,R_3}{R_1+R_2+R_3}, \qquad R_c = \frac{R_2\,R_3}{R_1+R_2+R_3}, \tag{9.3a}$$

$$R_1 = \frac{R_a\,R_b + R_a\,R_c + R_b\,R_c}{R_c}, \quad R_2 = \frac{R_a\,R_b + R_a\,R_c + R_b\,R_c}{R_b}, \quad R_3 = \frac{R_a\,R_b + R_a\,R_c + R_b\,R_c}{R_a}; \tag{9.3b}$$

the relations in (9.3a) may be used to determine $\{R_a, R_b, R_c\}$ from $\{R_1, R_2, R_3\}$, and the relations in (9.3b) may be used to determine $\{R_1, R_2, R_3\}$ from $\{R_a, R_b, R_c\}$. $\triangle$

Similar equivalence relations may be determined for the Y and Δ configurations of capacitors in Figures 9.5c-d (see code in RR.ch09). The Y-Δ transformation developed in Example 9.4 is applied to find the equivalent resistance of a network of resistors (see Figure 9.6d-e) at the end of Example 9.5 below.

[14] Following the recommendations of Appendix A, make certain that you comment your codes sufficiently, in English, to make such codes easy to understand and debug. As seen in Algorithm 9.1, note in particular that well-structured comments in the vicinity of the lines of code that define A and $\mathbf{b}$ significantly improve readability, minimizing transcription errors.

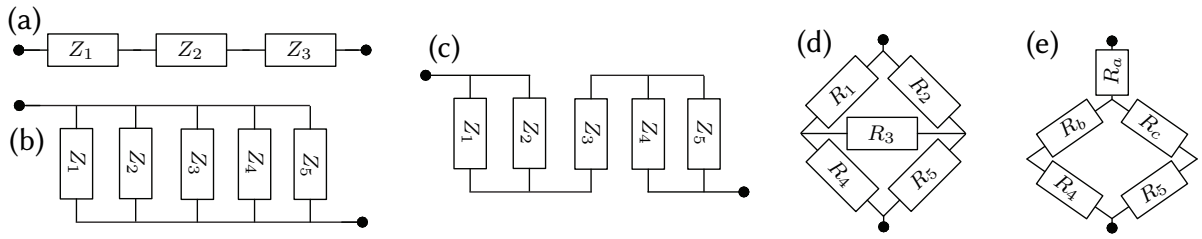


Figure 9.6: (a) Series, (b) parallel, and (c) "reducible" networks of a single type of component, with the Z_k denoting either resistors R_k, capacitors C_k, or inductors L_k; (d) a so-called "irreducible" network of resistors, and (e) an equivalent "reducible" network of resistors, with $\{R_a, R_b, R_c\}$ given by (9.3a).

Example 9.5 Equivalent resistance, capacitance, and inductance. The notion of equivalent circuits, with identical current-voltage relationships at one or more pairs of terminals, was defined in Fact 9.1. By the KCL and KVL, it follows that a set of n resistors, capacitors, or inductors in a **series connection** (see Figure 9.6a), in which the current I through the devices is equal and the voltages add, $\Delta V_1 + \Delta V_2 + \ldots \Delta V_n = \Delta V$, have the **equivalent resistance** R, **equivalent capacitance** C, or **equivalent inductance** L, respectively, of:

$$\begin{array}{lllll}
\Delta V_1 = I\,R_1, & \Delta V_2 = I\,R_2, & \ldots & \Delta V_n = I\,R_n & \Rightarrow \quad \Delta V = I\,R \quad \text{where} \quad R = R_1 + R_2 + \ldots R_n, \\
\dfrac{d\,\Delta V_1}{dt} = \dfrac{I}{C_1}, & \dfrac{d\,\Delta V_2}{dt} = \dfrac{I}{C_2}, & \ldots & \dfrac{d\,\Delta V_n}{dt} = \dfrac{I}{C_n} & \Rightarrow \quad \dfrac{d\,\Delta V}{dt} = \dfrac{I}{C} \quad \text{where} \quad \dfrac{1}{C} = \dfrac{1}{C_1} + \dfrac{1}{C_2} + \ldots \dfrac{1}{C_n}, \\
\Delta V_1 = L_1 \dfrac{dI}{dt}, & \Delta V_2 = L_2 \dfrac{dI}{dt}, & \ldots & \Delta V_n = L_n \dfrac{dI}{dt} & \Rightarrow \quad \Delta V = L\,\dfrac{dI}{dt} \quad \text{where} \quad L = L_1 + L_2 + \ldots L_n.
\end{array}$$

Similarly, a set of n resistors, capacitors, or inductors in a **parallel connection** (see Figure 9.6b), in which the voltage ΔV across the devices is equal and the currents add, $I_1 + I_2 + \ldots I_n = I$, have the **equivalent resistance** R, **equivalent capacitance** C, or **equivalent inductance** L, respectively, of:

$$\begin{array}{lllll}
I_1 = \dfrac{\Delta V}{R_1}, & I_2 = \dfrac{\Delta V}{R_2}, & \ldots & I_n = \dfrac{\Delta V}{R_n} & \Rightarrow \quad I = \dfrac{\Delta V}{R} \quad \text{where} \quad \dfrac{1}{R} = \dfrac{1}{R_1} + \dfrac{1}{R_2} + \ldots \dfrac{1}{R_n}, \\
I_1 = C_1 \dfrac{d\,\Delta V}{dt}, & I_2 = C_2 \dfrac{d\,\Delta V}{dt}, & \ldots & I_n = C_n \dfrac{d\,\Delta V}{dt} & \Rightarrow \quad I = C\,\dfrac{d\,\Delta V}{dt} \quad \text{where} \quad C = C_1 + C_2 + \ldots C_n, \\
\dfrac{dI_1}{dt} = \dfrac{\Delta V}{L_1}, & \dfrac{dI_2}{dt} = \dfrac{\Delta V}{L_2}, & \ldots & \dfrac{dI_n}{dt} = \dfrac{\Delta V}{L_n} & \Rightarrow \quad \dfrac{dI}{dt} = \dfrac{\Delta V}{L} \quad \text{where} \quad \dfrac{1}{L} = \dfrac{1}{L_1} + \dfrac{1}{L_2} + \ldots \dfrac{1}{L_n}.
\end{array}$$

In "reducible" networks of a single type of components, repeated application of the above simple rules for series and parallel connections of a single type of components is sufficient to determine an equivalent single component value. For example, if the Z_k in Figure 9.6c denote resistors, then

- the equivalent resistance of the parallel connection of R_1 and R_2 is $R_a = R_1 R_2/(R_1 + R_2)$,
- the equivalent resistance of the parallel connection of R_4 and R_5 is $R_b = R_4 R_5/(R_4 + R_5)$, and
- the equivalent resistance of the entire series connection (of R_a, R_3, and R_b) is $R = R_a + R_3 + R_b$.

On the other hand, repeated application of the above rules for parallel and series connections of a single type of components is not sufficient to simplify a so-called "irreducible" network, such as that shown in Figure 9.6d, to an equivalent single component value. However, additionally applying Y-Δ transformations, such as that derived in Example 9.4, can often convert such a network to a reducible form. To illustrate, with $\{R_a, R_b, R_c\}$ as given in (9.3a), Figure 9.6d can be equivalently realized as Figure 9.6e with (applying the above rules) the equivalent resistance of $R = R_a + 1/[1/(R_b + R_4) + 1/(R_c + R_5)]$, which upon (computer-aided) simplification reduces to

$$R = \frac{R_1R_2R_3 + R_1R_2R_4 + R_1R_2R_5 + R_1R_3R_5 + R_2R_3R_4 + R_1R_4R_5 + R_2R_4R_5 + R_3R_4R_5}{R_1R_3 + R_1R_4 + R_2R_3 + R_1R_5 + R_2R_4 + R_2R_5 + R_3R_4 + R_3R_5}. \qquad \triangle$$

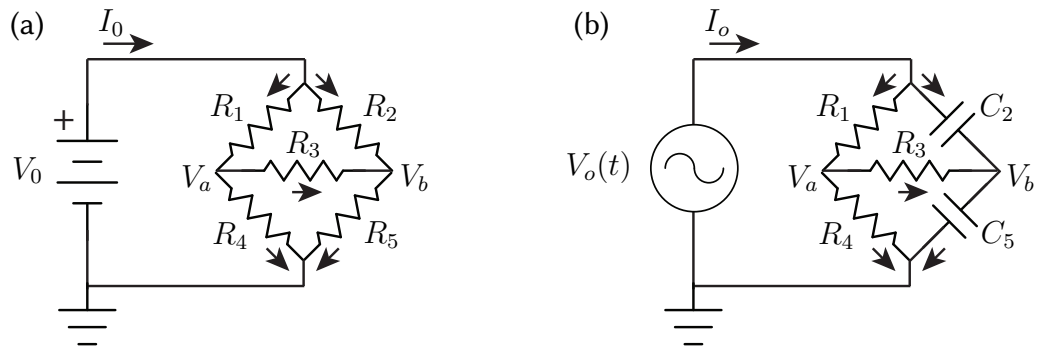


Figure 9.7: Wheatstone bridges for accurate measurement of (a) resistance, and (b) capacitance.

Example 9.6 Full analysis of a Wheatstone bridge. Consider the **Wheatstone bridge** of Figure 9.7a. For convenience, we number the elements of a circuit sequentially, and denote by I_k the current through element k, with positive current indicated by the direction of the arrow (this keeps us from having to label each current component individually in the figure). The (constant) voltage at the top of the bridge is V_0 (as it is connected to the top of the battery), and the voltage at the bottom of the bridge is 0 (as it is connected to the bottom of the battery, which is defined as ground), thus leaving two undetermined nodal voltages, $\{V_a, V_b\}$. We may thus determine the eight unknowns $\{I_0, I_1, I_2, I_3, I_4, I_5, V_a, V_b\}$ given the six parameters $\{V_0, R_1, R_2, R_3, R_4, R_5\}$ via KCL at three of the four nodes and the current-voltage relationship across each of the five resistors:

$$I_0 = I_1 + I_2, \quad I_1 = I_3 + I_4, \quad I_2 + I_3 = I_5, \tag{9.4a}$$

$$V_o - V_a = I_1 R_1, \quad V_o - V_b = I_2 R_2, \quad V_a - V_b = I_3 R_3, \quad V_a = I_4 R_4, \quad V_b = I_5 R_5. \tag{9.4b}$$

These eight linear equations in the eight unknowns $\mathbf{x} = \{I_0, I_1, I_2, I_3, I_4, I_5, V_a, V_b\}$ may easily be written in the form $A\mathbf{x} = \mathbf{b}$ and solved for $\mathbf{x}$, as illustrated in the corresponding code in RR.ch09, which is provided for $R_1 = R_4 = 1\,\text{kohm}$, $R_3 = 100\,\text{kohm}$, and $V_0 = 5\,\text{V}$, keeping R_2 and R_5 as symbolic (all choices that are easily changed), from which it follows immediately that $I_3 = (R_2 - R_5)/(201000\,R_2 + 201000\,R_5 + 2\,R_2\,R_5)$ amps.

The Wheatstone bridge is particularly useful for the precise measurement of an unknown resistor value (taken here as R_5) given three accurately known (often, equal) resister values (taken here as $\{R_1, R_2, R_4\}$), and a center resistor R_3. Indeed, if $R_1/R_4 = R_2/R_5$, then the bridge is said to be **balanced**, and the current through the resister in the center of the bridge, I_3 (which may be measured precisely using a **galvanometer**), will be exactly zero, as $V_a = V_b$ in this case. △

Example 9.7 A Wheatstone bridge for measuring capacitance. If we simply replace the resistors R_2 and R_5 in Figure 9.7a with capacitors C_2 and C_5, where C_2 is known and C_5 is unknown, and observe the circuit at steady state, we run into a problem: setting the time derivatives of the current through the capacitors equal to zero at steady state (that is, taking $I_2 = I_5 = 0$), it follows that $I_3 = 0$ and thus $V_a = V_b$ regardless of the value of C_5! Thus, C_5 can *not* be determined in this simplistic manner.

However, as indicated in Figure 9.7b, if we replace resistors R_2 and R_5 with capacitors C_2 and C_5, and we also replace the constant voltage source with a (sinusoidal) time-varying voltage source, then we can now easily determine C_5. Our eight equations for the eight unknowns $\{I_0, I_1, I_2, I_3, I_4, I_5, V_a, V_b\}$ now take the form

$$I_0 = I_1 + I_2, \quad I_1 = I_3 + I_4, \quad I_2 + I_3 = I_5,$$

$$V_o - V_a = I_1 R_1, \quad I_2 = C_2\, d(V_o - V_b)/dt, \quad V_a - V_b = I_3 R_3, \quad V_a = I_4 R_4, \quad I_5 = C_5\, d(V_b)/dt,$$

where only two equations (those for the capacitors) have changed. Assuming $R_1 = R_2 = 1\,\text{k}\Omega$, $C_3 = 10\,\mu\text{F}$, and $R_5 = 100\,\text{k}\Omega$, assuming that $\{I_0, I_1, I_2, I_3, I_4, I_5, V_a, V_b, V_o\}$ are all initially zero, taking the Laplace transform [see §9.1.5], and performing an analogous symbolic manipulation (see code in RR.ch09), it follows that, in

Laplace transform space,

$$\frac{I_3(s)}{V_0(s)} = G(s) = \frac{(C_5 - C_2)s}{2.01 \times 10^5 (C_5 + C_2)s + 2}.$$

As in the case of drawing Bode plots (see §8.4), we are interested in the magnitude and phase of the persistent sinusoidal component of the output current $I_3(t)$ when the input voltage $V_0(t)$ is sinusoidal. That is, for the transfer function $G(s)$ given above, taking $V_0(t) = V\sin(\omega t)$ will result in $I_3(t) = I\sin(\omega t + \phi)+$ terms that decay with time, where $I = |G(\mathrm{i}\omega)|$ and $\phi = \angle G(\mathrm{i}\omega)$, and thus

$$\frac{I}{V} = \frac{|C_5 - C_2|\,\omega}{\sqrt{[2.01 \times 10^5 (C_5 + C_2)]^2\omega^2 + 4}}, \quad \phi = \begin{cases} 90^\circ - \text{atan2}\{[2.01 \times 10^5 (C_5 + C_2)]\omega, 2\} & \text{if } C_5 > C_2, \\ -90^\circ - \text{atan2}\{[2.01 \times 10^5 (C_5 + C_2)]\omega, 2\} & \text{if } C_5 < C_2. \end{cases}$$

If $\omega = 0$, then $I = 0$ regardless of C_5, consistent with the comments made in the previous paragraph.
If $\omega > 0$ and $I = 0$, it follows immediately that the bridge is in balance, and thus $C_5 = C_2 = 10\ \mu$F.
If $\omega > 0$ and $I \neq 0$, C_5 may be determined from I according to the above expressions (with $C_5 > C_2$ if $\phi > 0$, and $C_5 < C_2$ if $\phi < 0$; note the 180° phase shift in ϕ when C_5 is changed from below C_2 to above C_2).

Inductance may be quantified with a Wheatstone bridge in an analogous fashion. △

Example 9.8 Equivalent sources Any combination of voltage sources, current sources, and resistors leads to a linear current-voltage relationship like those in Figure 9.2; the following facts follow as consequence.

Fact 9.4 (Thévenin's theorem) *Any circuit containing only voltage sources, current sources, and resistors can be converted to a* **Thévenin equivalent circuit**, *with one ideal voltage source and one resistor in series.*

Fact 9.5 (Norton's theorem) *Any circuit containing only voltage sources, current sources, and resistors can be converted to a* **Norton equivalent circuit**, *with one ideal current source and one resistor in parallel.*

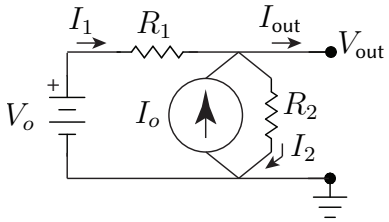


Figure 9.8: A circuit with both voltage and current sources as well as two resistors.

To illustrate, consider the circuit shown in Figure 9.8. Writing down KCL at the node at the top of the circuit and Ohm's law across each resistor, the (linear) current-voltage relationship at the terminals may be determined:

$$\left.\begin{aligned} I_1 + I_o &= I_{\text{out}} + I_2 \\ V_o - V_{\text{out}} &= I_1 R_1 \\ V_{\text{out}} &= I_2 R_2 \end{aligned}\right\} \quad \Rightarrow \quad I_{\text{out}} = I_o + I_1 - I_2 = \left(I_o + \frac{V_o}{R_1}\right) - \left(\frac{1}{R_1} + \frac{1}{R_2}\right)V_{\text{out}}. \tag{9.5}$$

It follows from this analysis of the circuit in Figure 9.8 that

- its Thévenin equivalent (Figure 9.1e) sets $R_V = R_1 R_2/(R_1 + R_2)$ and $V_s = (I_o + V_o/R_1)R_V$, and
- its Norton equivalent (Figure 9.1f) sets $R_I = R_1 R_2/(R_1 + R_2)$ and $I_s = I_o + V_o/R_1$;

note that all of these circuits have "equivalent" current-voltage relationships, as illustrated in Figure 9.2.

Of particular interest in this example is the question of how much power is actually provided by the two sources in Figure 9.8. To simplify, assume first that $R_1 = I_{\text{out}} = 0$; in this case, $V_{\text{out}} = V_o$, and

- the power absorbed by the current source is $-I_o V_o < 0$ (that is, power is *provided* by the current source regardless of the relative magnitudes of I_o and V_o/R_2), whereas
- the power absorbed by the voltage source is $-I_1 V_o = -(I_2 - I_o)V_o = -(V_o/R_2 - I_o)V_o$ (that is, power is *provided* by the voltage source if $V_o/R_2 > I_o$, and is *absorbed* by the voltage source if $V_o/R_2 < I_o$).

Taking $R_1 > 0$ and $I_{\text{out}} \neq 0$, similar conclusions may be drawn. △

9.1.5 Laplace transform analysis of circuits and the definition of impedance

In simple circuits without capacitors or inductors, combining KCL and the current-voltage relationship across each component leads to straightforward systems of *linear algebraic equations*, which may be solved by hand or with symbolic numerical tools. In more interesting circuits incorporating capacitors and/or inductors, combining KCL and the current-voltage relationship across each component often leads, more generally, to sets of *linear constant-coefficient ODEs* together with various algebraic constraints (jointly referred to as **descriptor systems**). Without the Laplace transform, as developed in §8, the analysis of such systems would be difficult. However, as seen in the various examples presented above, application of the Laplace transform to such systems converts them back to straightforward systems of algebraic equations, incorporating the Laplace transform variable s, that are again easy to solve by hand or with simple symbolic numerical tools.

It is thus seen that, when analyzing electric circuits, *working in the Laplace domain is essential*. Further, one is often interested in the *frequency response* of an electric circuit subject to sinusoidal excitation. As shown in §8.4, the gain and phase shift of the persistent sinusoidal component of the output of a linear system $G(s)$ when excited by a sinusoidal input may be summarized by the Bode plot of $G(s)$, and may be calculated simply by evaluating the magnitude and phase of $G(\mathrm{i}\,\omega)$ as a function of the frequency ω of the input sinusoid.

Example 9.9 Impedance of the fundamental circuit elements. Taking the Laplace transform of the current-voltage relationships of resistors, capacitors, and inductors [see (9.2)] and evaluating at $s = \mathrm{i}\,\omega$ gives

$$G_{\text{resistor}}(\mathrm{i}\,\omega) = \frac{V(\mathrm{i}\,\omega)}{I(\mathrm{i}\,\omega)} = R \triangleq Z_R, \quad G_{\text{capacitor}}(\mathrm{i}\,\omega) = \frac{V(\mathrm{i}\,\omega)}{I(\mathrm{i}\,\omega)} = \frac{-\mathrm{i}}{\omega\, C} \triangleq Z_C, \quad G_{\text{inductor}}(\mathrm{i}\,\omega) = \frac{V(\mathrm{i}\,\omega)}{I(\mathrm{i}\,\omega)} = \mathrm{i}\,\omega\, L \triangleq Z_L.$$

The quantities Z_R, Z_C, and Z_L are used often, and are commonly referred as the **impedance** of each of these components. In other texts, the concept of impedance is often discussed somewhat loosely as a *complex, frequency-dependent generalization of resistance* even before Laplace transforms are properly introduced. This approach is, perhaps, unnecessarily convoluted; pedagogically, the author recommends instead mastering the Laplace transform (§8.2) and Bode plot (§8.4) *before* reading the present discussion (and that which follows); the frequency response of the current-voltage relationships represented by the (complex) transfer functions $G_{\text{resistor}}(s)$, $G_{\text{capacitor}}(s)$, and $G_{\text{inductor}}(s)$ as listed above [and, $G_{\text{speaker}}(s)$ and $G_{\text{piezo}}(s)$, as discussed in Examples 9.10 and 9.11 below] are then quite easy to interpret. In particular, these characterizations are consistent with the phenomenological description of the general behavior of capacitors and inductors given in §9.1.3.1:

- the voltage across a capacitor *lags* the current through the capacitor by 1/4 cycle [$\phi = -90°$, associated with the $-\mathrm{i}$ factor in $G_{\text{capacitor}}(\mathrm{i}\,\omega)$], with the magnitude of the sinusoidal voltage across the capacitor divided by the magnitude of the sinusoidal current through the capacitor *decreasing* with frequency ω;
- the voltage across an inductor *leads* the current through the inductor by 1/4 cycle [$\phi = 90°$, associated with the i factor in $G_{\text{inductor}}(\mathrm{i}\,\omega)$], with the magnitude of the sinusoidal voltage across the inductor divided by the magnitude of the sinusoidal current through the inductor *increasing* with frequency ω. △

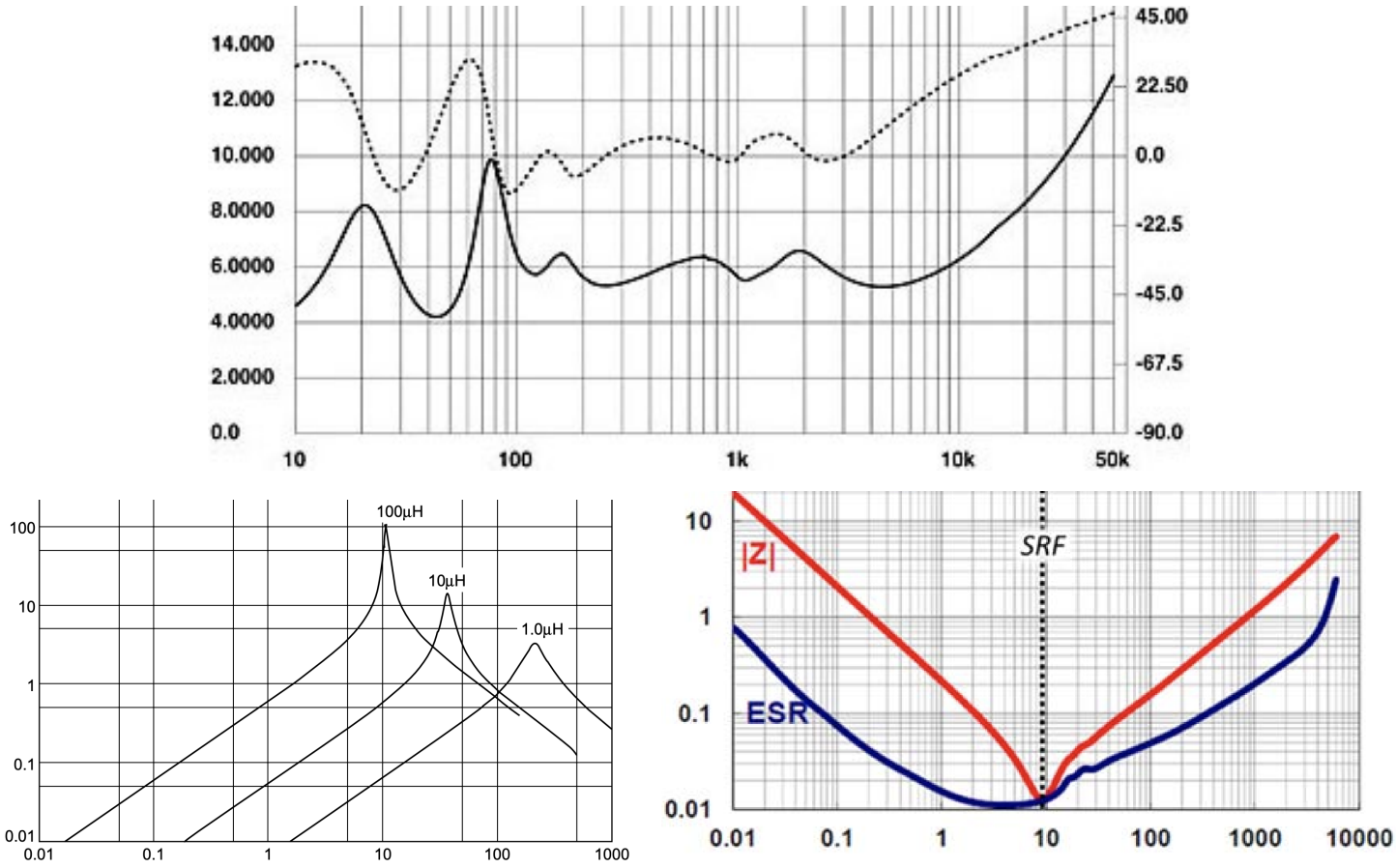


Figure 9.9: Bode plot of the transfer function $G(s) = V(\mathrm{i}\omega)/I(\mathrm{i}\omega)$ from current to voltage, also referred to as the **impedance**, of (a) a Intimus 533-T speaker, (b) Murata LQH43C 1 μH, 10 μH, & 100 μH inductors, and (c) a Samsung CL21A226 22 μF capacitor. In (a), the solid curve indicates the magnitude, with its linear scale at left (in ohms), and the dotted curve indicates the phase, with its scale at right (in degrees), w.r.t. the frequency in Hz; note the magnitude is around 6 ohms (varying between about 5.3 ohms and 6.7 ohms) over audio frequencies from 100 Hz to 10 kHz, and the phase is around 0° (varying between about -10° and 23°) over frequencies in this range. In (b) the magnitude of the impedance is shown in kohms, w.r.t. the frequency in MHz; and in (c) the magnitude of the impedance is shown in ohms, w.r.t. the frequency in MHz.

Example 9.10 Impedance of real devices. Figure 9.9 illustrates the (experimentally determined) Bode plots of the transfer function $G(s) = V(s)/I(s)$ from current to voltage [commonly referred to as the (frequency-dependent) **impedance**] of a typical COTS (a) audio speaker, (b) inductor, and (c) capacitor.

If the speaker in Figure 9.9a presented a purely resistive load to an electric circuit, the magnitude of its impedance (measured in ohms; 4 ohm, 6 ohm, and 8 ohm speakers are common) would be constant across all ω, and its phase would be zero. Of course, the dynamics of a speaker depends on several mechanical and electrical details of its construction; nonetheless, over typical audio frequencies (from 100 Hz to 10 kHz), this speaker approximates a simple resistive load of about 6 ohm. The magnitude of this Bode plot, $|G(\mathrm{i}\omega)|$, reveals peaks and valleys ("resonances" and "antiresonances") as a function of ω, while the phase $\angle G(\mathrm{i}\omega)$ is generally positive wherever $|G(\mathrm{i}\omega)|$ increases with ω, and negative wherever $|G(\mathrm{i}\omega)|$ decreases with ω.

The devices in Figure 9.9b and Figure 9.9c act, respectively, like inductors (with impedance increasing linearly with ω) and capacitors (with impedance decreasing linearly with ω) up to near a **Series Resonant Frequency** (SRF) of between 10 and 200 MHz; above this SRF, their behavior generally switches (with the inductor beginning to act more like a capacitor, and the capacitor beginning to act more like an inductor). △

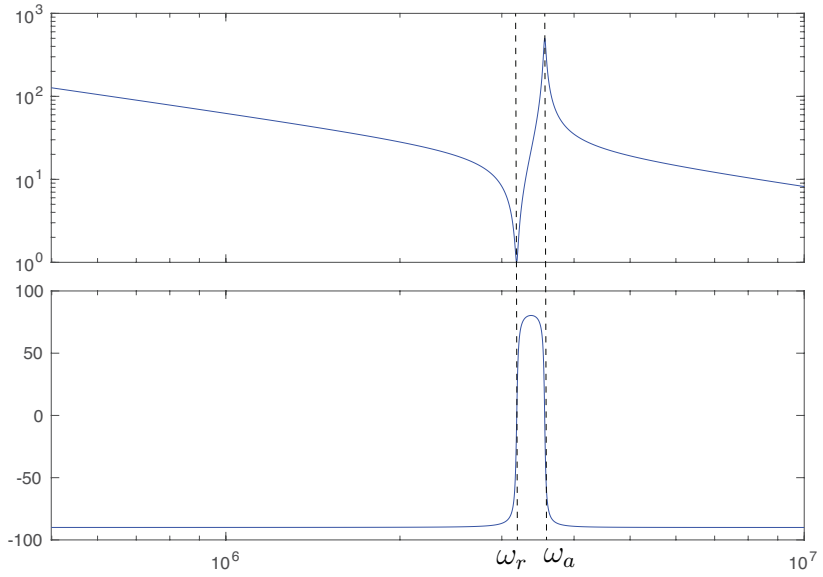


Figure 9.10: Bode plot of the transfer function $G(s) = V(\mathrm{i}\,\omega)/I(\mathrm{i}\,\omega)$ from current to voltage, also referred to as the **impedance**, of the Butterworth/van Dyke circuit model of a piezoelectric material. The system is said to be essentially **inductive** ($\phi \approx 90°$) for $\omega_r < \omega < \omega_a$, and essentially **capacitive** ($\phi \approx -90°$) outside this range.

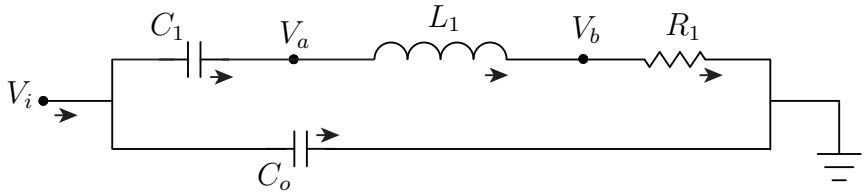


Figure 9.11: The Butterworth/van Dyke circuit model of a piezoelectric material (see Example 9.11).

Example 9.11 Impedance of piezos. Many crystalline materials, such as quartz crystals, exhibit a **piezoelectric effect** such that

- when a voltage is applied across the material, it mechanically deforms, and
- when the material is deformed, a charge accumulates on its surface that generates a measurable voltage.

Electromechanical devices (both sensors and actuators) that exhibit strong resonant oscillations may be constructed using such piezoelectric materials. A model electric circuit capturing the essence of such electromechanical devices is illustrated in Figure 9.11. The Laplace-transformed equations governing this circuit are

$$I_i = I_o + I_1, \quad I_o = C_o\, s\, V_i, \quad I_1 = C_1\, s\, [V_i - V_a], \quad [V_a - V_b] = s\, L_1\, I_1, \quad V_b = R_1\, I_1;$$

that is, 5 eqns in the 5 variables $\mathbf{x}(s) = \{I_o(s), I_1(s), V_i(s), V_a(s), V_b(s)\}$, treating the current $I_i(s)$ as an input and $\{C_o, C_1, L_1, R_1\}$ as parameters. Solving (see code in RR.ch09) gives

$$G_{\text{piezo}}(s) = \frac{V_i(s)}{I_i(s)} = \frac{L_1\, C_1\, s^2 + R_1\, C_1\, s + 1}{s\,(L_1\, C_o\, C_1\, s^2 + R_1\, C_o\, C_1\, s + C_o + C_1)} = K \frac{s^2 + 2\,\zeta_r\,\omega_r\, s + \omega_r^2}{s(s^2 + 2\,\zeta_a\,\omega_a\, s + \omega_a^2)},$$

where $\omega_r = 1/\sqrt{L_1\, C_1}$, $\zeta_r = R_1/(2\,\omega_r\, L_1)$, $\omega_a = 1/\sqrt{L_1\, C_a}$ with $C_a = C_o\, C_1/(C_o + C_1)$, $\zeta_a = R_1/(2\,\omega_a\, L_1)$, $K = 1/C_o$; for $\{C_o, C_1, L_1, R_1\} = \{2\,\text{nF}, 0.5\,\text{nF}, 5\,\mu\text{H}, 1\,\text{ohm}\}$, the Bode plot of $G_{\text{piezo}}(s)$ is given in Figure 9.9b, indicating a lightly-damped resonance and antiresonance at $\omega_r = 3.18\,\text{MHz}$ and $\omega_a = 3.56\,\text{MHz}$. △

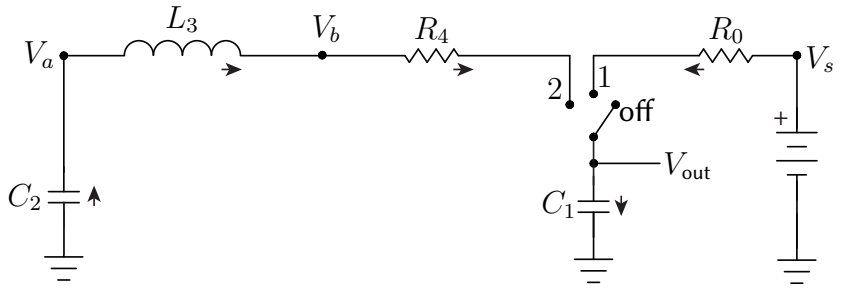


Figure 9.12: The **LC tank oscillator** considered in Example 9.12.

Example 9.12 LC tank oscillator. We now consider the oscillator circuit in Figure 9.12, initialized with the switch in the **off** position, current equal to zero everywhere, and all capacitors fully discharged.

Startup. At $t = 0$, with $V_{\text{out}}(t=0) = 0$, we turn the switch from off to **position 1**. Current flows from the battery, at a (constant) voltage V_s, through R_0, through C_1, to ground. Note that the current through R_0 and C_1 are equal, and are thus denoted here as simply

$$I_0(t) = I_1(t) = I(t),$$

with positive current in the direction of the arrows shown. Note by Table 8.1 that the Laplace transform of $V_{\text{battery}}(t) = V_s$ (i.e., constant in time) is $V_{\text{battery}}(s) = V_s/s$. The component equations for R_0 and C_1 are thus

$$V_s/s - V_{\text{out}}(s) = R_0\, I(s), \quad s\, C_1 V_{\text{out}}(s) = I(s).$$

These two eqns in $\{I(s), V_{\text{out}}(s)\}$ are easily reduced to one eqn in $V_{\text{out}}(s)$. Defining $a = 1/(R_0\, C_1)$, we have:

$$\begin{aligned} V_s/s - V_{\text{out}}(s) = s\, R_0\, C_1 V_{\text{out}}(s) \quad &\Rightarrow \quad (s\, R_0\, C_1 + 1) V_{\text{out}}(s) = V_s/s \quad \Rightarrow \\ V_{\text{out}}(s) = V_s \frac{a}{s(s+a)} = V_s\Big(\frac{1}{s} - \frac{1}{s+a}\Big) \quad &\Rightarrow \quad \boxed{V_{\text{out}}(t) = V_s(1 - e^{-t/(R_0 C_1)})}. \end{aligned} \tag{9.6a}$$

By the component equation for C_1, the corresponding current $I(t)$ is

$$I(t) = C_1\, dV_{\text{out}}(t)/dt \quad \Rightarrow \quad I(t) = (V_s/R_0)\, e^{-t/(R_0 C_1)}. \tag{9.6b}$$

The responses of $V_{\text{out}}(t)$ and $I(t)$ are characterized by exponential decay, with

$$V_{\text{out}}(t) \to V_s \quad \text{and} \quad I(t) \to 0 \quad \text{as} \quad t \to \infty. \tag{9.6c}$$

If you know values for $\{V_s, R_0, C_1\}$, the time t_s after which $V_{\text{out}}(t)$ settles to within, say, 95% of its steady state value is given by setting $0.95\, V_o = V_o(1 - e^{-t/(R_0 C_1)})$ and solving for t, giving

$$t = -R_0\, C_1 \ln(0.05) \approx 3\, R_0\, C_1. \tag{9.6d}$$

Decaying oscillations. We now move the switch from position 1 to **position 2**, and for simplicity we also reset the clock, so that $t = 0$ now corresponds to the time that we flip the switch to position 2. Starting from the steady-state values of the startup phase, we have $V_{\text{out}}(t=0) = V_s$ and $I(t=0) = 0$ [see (9.6c)]. KCL now gives simply

$$I_2(t) = I_3(t) = I_4(t) = I_1(t) \triangleq I(t).$$

Note by (8.9a) that the Laplace transform of $V'_{\text{out}}(t) = d[V_{\text{out}}(t)]/dt$ is $V'_{\text{out}}(s) = s\, V_{\text{out}}(s) - V_s$. Implementing KCL, given above, into the Laplace transform of the component equations for $\{C_2, L_3, R_4, C_1\}$, we have four eqns with three intermediate variables, $\{I(s), V_a(s), V_b(s)\}$, to be eliminated:

$$I(s) = s\, C_2\, [-V_a(s)], \quad V_a(s) - V_b(s) = s\, L_3\, I(s), \quad V_b(s) - V_{\text{out}}(s) = R_4 I(s), \quad I(s) = C_1[s\, V_{\text{out}}(s) - V_s].$$

Reducing these four algebraic eqns in the four variables $\{I(s), V_a(s), V_b(s), V_{\text{out}}(s)\}$ to one eqn in $V_{\text{out}}(s)$ is easily done by hand (or, better, in Matlab; see code in RR.ch09), giving immediately:

$$V_{\text{out}}(s) = \frac{b_2 s^2 + b_1 s + b_0}{s[s^2 + a_1 s + a_0]} \quad \text{with} \tag{9.7}$$
$$a_1 = \frac{R_4}{L_3}, \quad a_0 = \frac{1}{L_3 C}, \quad \frac{1}{C} = \frac{1}{C_1} + \frac{1}{C_2} = \frac{C_1 + C_2}{C_1 C_2}, \quad b_2 = V_s, \quad b_1 = \frac{V_s R_4}{L_3}, \quad b_0 = \frac{V_s}{L_3 C_2}.$$

Noting the $e^{-\sigma t}\cos(\omega_d t)$ and $e^{-\sigma t}\sin(\omega_d t)$ entries in Table 8.1, and setting

$$s^2 + a_1 s + a_0 = (s+\sigma)^2 + \omega_d^2 = s^2 + 2\,\sigma\, s + (\sigma^2 + \omega_d^2) \quad \Rightarrow \quad \sigma = a_1/2, \quad \omega_d = \sqrt{a_0 - a_1^2/4},$$

we may rewrite (9.7) via partial faction expansion as

$$V_{\text{out}}(s) = \frac{b_2\, s^2 + b_1\, s + b_0}{s\,[(s+\sigma)^2 + \omega_d^2]} = B_2\,\frac{1}{s}\cdot\frac{(s+\sigma)^2 + \omega_d^2}{(s+\sigma)^2 + \omega_d^2} + B_1\frac{(s+\sigma)}{(s+\sigma)^2 + \omega_d^2}\cdot\frac{s}{s} + B_0\frac{\omega_d}{(s+\sigma)^2 + \omega_d^2}\cdot\frac{s}{s}. \tag{9.8}$$

This may be solved for $\{B_2, B_1, B_0\}$ by forming a common denominator, as shown above, and setting like powers of s in the numerator as equal (again, see code in RR.ch09), which immediately gives

$$B_2 = \frac{b_0}{\sigma^2 + \omega_d^2}, \quad B_1 = b_2 - B_2, \quad B_0 = \frac{b_1 - b_2\sigma}{\omega_d} - B_2\frac{\sigma}{\omega_d}.$$

Thus, for $t \geq 0$,

$$V_{\text{out}}(t) = B_2 + B_1\, e^{-\sigma t}\cos(\omega_d\, t) + B_0\, e^{-\sigma t}\sin(\omega_d\, t), \tag{9.9a}$$
$$I(t) = C_1 dV_{\text{out}}(t)/dt = C_1 e^{-\sigma t}[(-\sigma B_1 + \omega_d B_0)\,\cos(\omega_d\, t) + (-\sigma B_0 - \omega_d B_1)\,\sin(\omega_d\, t)], \tag{9.9b}$$

where the constants $\{\omega_d, \sigma, B_2, B_1, B_0\}$ depend on $\{V_s, R_0, C_1, C_2, L_3, R_4\}$ via the various equations above.

If $R_4 = 0$ and $C_1 = C_2$, then $C/C_1 = C/C_2 = 1/2$, $a_1 = b_1 = \sigma = B_0 = 0$, and $B_2 = B_1 = V_s/2$, and thus a sort of balanced "see-saw" effect sets in, with

$$V_{\text{out}}(t) = V_s[1 + \cos(\omega_d\, t)]/2, \qquad I(t) = -(C_1 V_s/2)\,\omega_d \sin(\omega_d\, t),$$
$$V_a(t) = V_{\text{out}}(t) + [L_3\, d/dt]I(t) = (V_s/2)[1 + \cos(\omega_d\, t) - L_3\, C_1\, \omega_d^2 \cos(\omega_d t)] = V_s[1 - \cos(\omega_d\, t)]/2];$$

that is, the charge shifts from C_1 to C_2 (i.e., from V_{out} to V_a), and back, sinusoidally (undamped) over the period

$$T = 2\pi/\omega_d = 2\pi\sqrt{L_3 C} \quad \Rightarrow \quad \boxed{T = 2\pi\sqrt{L_3 C_1 C_2/(C_1 + C_2)}.}$$

Again, there will always be some resistance in practical capacitors and inductors; we model its net effect in this circuit with the (small) resistor R_4. Any such resistance (i.e., taking $R_4 > 0$) will result in $\sigma > 0$, and thus cause the oscillations to decay in time. This decay may be offset with an op amp, as explored in Example 9.34. △

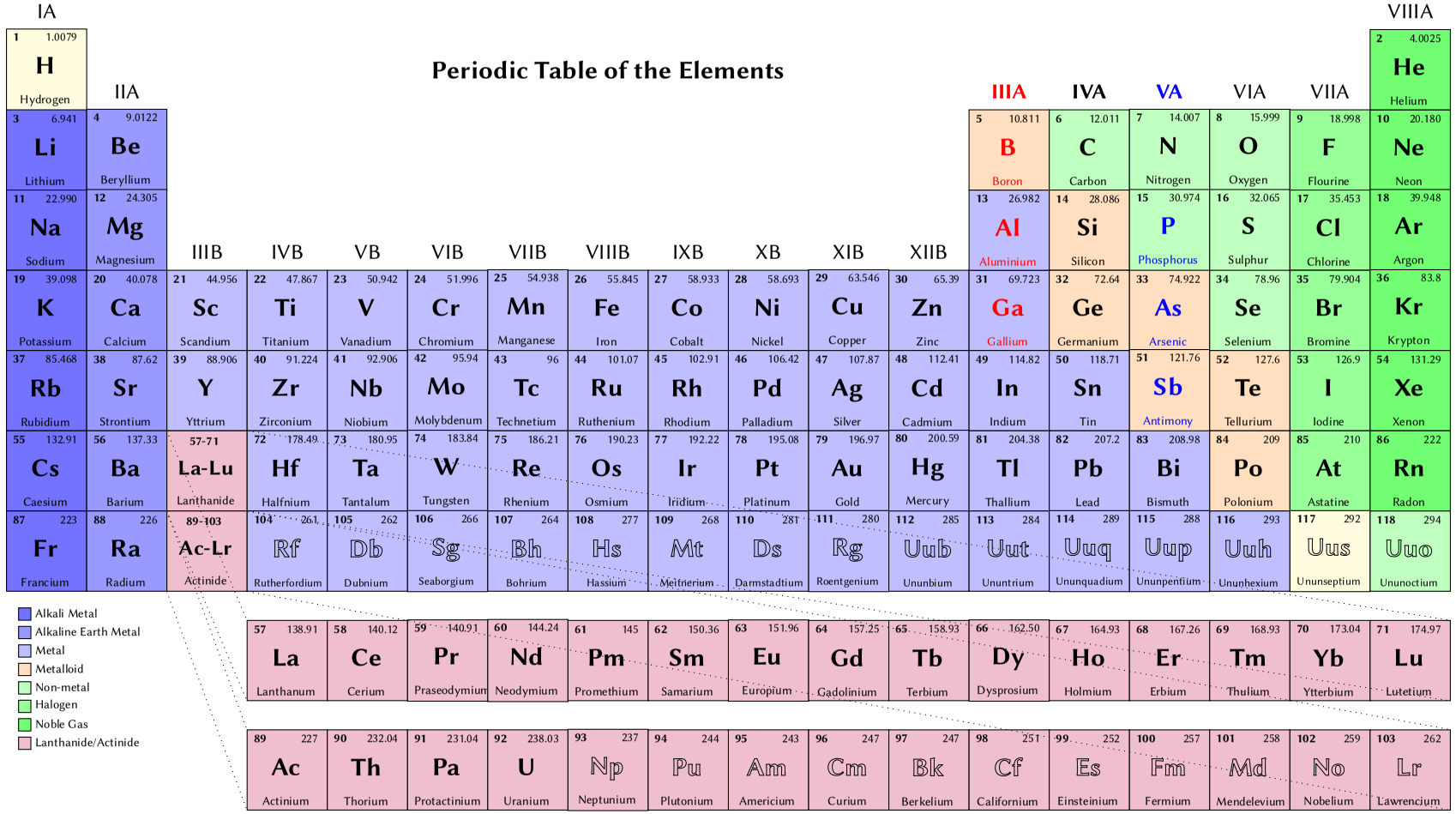


Figure 9.13: Periodic Table of the Elements, for providing context related to the structure of common semiconductors. Carbon, Silicon, and Germanium, which are all in column IVA (that is, the "carbon column"), crystallize in a **diamond crystal structure**. Gallium arsenide (equal parts Gallium, from column IIIA, and Arsenic, from column VA), crystallizes in a **zincblende crystal structure**, which is simply a diamond crystal structure with gallium and arsenic in alternating lattice sites. Silicon carbide can take any of about 250 crystalline forms, including zincblende. Note that **n-type dopants**, such as **phosphorus**, **arsenic**, or **antimony**, are taken from column VA (just to the right of the carbon column), thus introducing extra valence electrons (not bound by the crystal structure) that can move fairly easily when a voltage is applied. Similarly, **p-type dopants**, such as **boron**, **aluminum**, or **gallium**, are selected from column IIIA (just to the left of the carbon column), thus introducing "holes" in the crystal's electron structure that can also move fairly easily when a voltage is applied.

9.2 Active analog circuits & filters

A **semiconductor** is a material (often, a single crystal[15]) that has conduction properties that may be tuned during fabrication in various useful ways. A single pure crystal of semiconductor material, such as silicon (see Figure 9.13), is generally nonconductive (that is, it acts as an **insulator**), as all of the **valence** (outer-shell) electrons of the constituent atoms are tied up by the **covalent bonds** of the crystal.

However, if a semiconductor crystal is formed, or **doped**, with **n-type dopant** atoms such as **phosphorus** (see Figure 9.13), an extra valence electron is introduced in the crystal lattice for each atom of the n-type dopant in the crystal. These extra valence electrons can move fairly easily when a voltage is applied across the material, thus making an n-doped semiconductor, such as phosphorus-doped silicon, a conductor.

Similarly, if a semiconductor crystal is formed with **p-type dopant** atoms such as **boron** (see Figure 9.13), a valence electron is missing from the crystal lattice for each atom of the p-type dopant present in the crystal, forming what is known as a "**hole**" in the electron structure of the crystal. These holes in the electron structure of the crystal can also move fairly easily[16] when a voltage is applied across the material, thus making a p-doped semiconductor, such as boron-doped silicon, also a conductor.

Many useful devices may be made with semiconductors, some of which are reviewed in this section.

9.2.1 p-n junctions & diodes

When a semiconductor crystal has various neighboring sections, some that are p-doped and some that are n-doped, thus forming **p-n junctions** within the semiconductor, useful electrical characteristics arise. For example, a semiconductor crystal which has just two adjacent doped regions, with a single p-n junction in the middle, is called a **semiconductor diode**, which behaves in the ideal setting as follows:

- If a semiconductor diode is put under **forward bias**, with positive voltage applied to the p side of the semiconductor and negative voltage applied to the n side, then electrons will flow into the n side of the semiconductor, pushing free valence electrons in the crystal lattice towards the p-n junction. These electrons, in turn, flow into the nearby holes on the p side of the semiconductor and create, in effect, a flow holes on the p side of the semiconductor that is equal in magnitude and opposite in direction to the flow of electrons on the n side of the semiconductor, thus sustaining an electric current through the material with very little (ideally, zero) resistance[17].
- If a semiconductor diode is put under **reverse bias**, with positive voltage applied to the n side of the semiconductor and negative voltage applied to the p side, then some of the extra valence electrons on the n side of the semiconductor are pulled away from the p-n junction and out of the semiconductor, and some of the holes in the electron structure of the crystal on the p side of the semiconductor are pulled away from the p-n junction and, effectively, out of the semiconductor, creating a so-called **depletion layer** with neither holes nor free valence electrons to carry moving charge (that is, to sustain the current) in the vicinity of the p-n junction. As a result, an ideal diode under negative bias does not conduct.

[15] **Amorphous** semiconductors, which lack a long-range ordered crystal structure, can also be manufactured, and can be done so in especially thin layers over large areas. Such semiconductors may be doped in a manner similar to the single-crystal semiconductors discussed here, and can be switched from one physical state to another (e.g., from translucent to opaque), which makes them especially useful in a variety of applications, such as **CD**s/**DVD**s/**BD**s, **liquid-crystal displays** (**LCD**s), and **solar cells**.

[16] Note that it is actually a neighboring electron in the electron structure of a crystal lattice that moves, thereby changing the "hole" location in this electron structure; several successive movements of electrons into neighboring hole locations give the appearance that it is the hole itself that is moving.

[17] As a loose physical analogy of current flow in a diode under forward bias, one might visualize the electron motion (on the n-doped side) towards the p-n junction as tiny drops of rain falling through air toward an air/water interface, and the corresponding hole motion (on the p-doped side) towards the p-n junction as tiny bubbles of air rising through water toward the air/water interface at the same rate, resulting in zero net accumulation of negative or positive charge (raindrops or bubbles) at the interface.

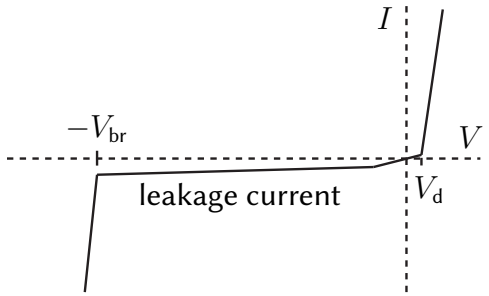


Figure 9.14: Typical current-voltage relationship of a real diode, with forward bias given by $V > 0$ and reverse bias given by $V < 0$, indicating the **breakdown voltage** V_{br}, the leakage current, and the **cut-in voltage** V_d.

Though the ideal model of a semiconductor diode described above is sometimes adequate, the deviations of real semiconductor diodes from this behavior, as indicated in Figure 9.14, are important to quantify:

- Within the depletion layer described above in the reverse bias setting, the n-doped side, now lacking its extra valence electrons, is positively charged, and the p-doped side, now lacking its holes, is negatively charged by the same amount. This sets up an electric field across the p-n junction. When the applied voltage exceeds a certain **breakdown threshold** V_{br} (typically 5 to 20 volts), one of two phenomena occurs (which phenomena sets in first depends on various particular details of the diode).
  - In **Zener breakdown**, this electric field directly breaks some of the covalent bonds in the semiconductor crystal, thus allowing the resulting freed electrons to act as charge carriers.
  - In **avalanche breakdown**, on the other hand, the electric field accelerates free valence electrons near the edge of the depletion layer to sufficient energies that their subsequent collision with bound electrons can break covalent bonds within the depletion layer, resulting in the creation of additional charge carriers (pairs of free electrons and holes), which in turn collide with other bound electrons within the depletion layer to create still more charge carriers, etc.

  Note that avalanche breakdown is **hysteretic** (that is, after it sets in and the additional charge carriers are created within the depletion layer, the semiconductor continues to conduct even after the voltage is reduced below the breakdown threshold), whereas Zener breakdown is not. Diodes designed to undergo these types of breakdown at specific voltages without being damaged, called **Zener diodes** and **avalanche diodes** respectively, are both useful in electric circuit design.
- **Diffusion** of charge carriers (electrons and holes) across the p-n junction in a diode sets up a small depletion zone and a corresponding **built-in voltage** even when the external voltage applied to the diode is zero. Thus, under forward bias, the applied voltage must exceed a certain **cut-in voltage** V_{d} (0.6 to 0.7 V for silicon diodes, 0.25 to 0.3 V for germanium diodes, and 0.15 to 0.45 volts for Schottky diodes) before current will begin to flow.
- Due to a weak thermodynamic process of **carrier generation and recombination** inside the depletion layer, a small **leakage current** always flows through a diode under reverse bias even when the applied voltage is below the breakdown threshold. Note in particular that
  - **carrier generation** due to the absorption of energy of incident photons, and the resulting current when under forward bias, is how **photodiodes** respond to the intensity of incident light, whereas
  - in **light-emitting diodes** (**LED**s), during **carrier recombination**, energy is released as photons.
- Finally, under both forward bias (with the applied voltage exceeding the cut-in voltage V_{d}) and reverse bias (with the magnitude of the applied voltage exceeding the breakdown voltage V_{br}), a diode exhibits very small **effective resistance**, thus creating a very steep slope in a plot of I versus V (see Figure 9.14).

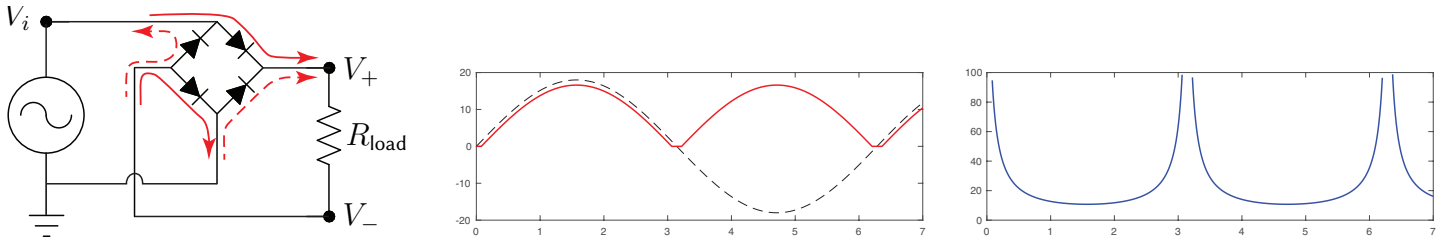


Figure 9.15: (a) A full-wave bridge rectifier circuit, with direction of current flow through diode bridge indicated as solid when $V_i > 2\,V_d$, and as dashed when $V_i < -2\,V_d$; when $|V_i| \leq 2\,V_d$, no current flows. (b) Output voltage, $V_+(t) - V_-(t)$, and (c) the percent of power lost in the diodes, $\epsilon(t)$, for $V_i(t) = 18\,\sin(\omega t)$ and $V_d = 0.7\,\text{V}$.

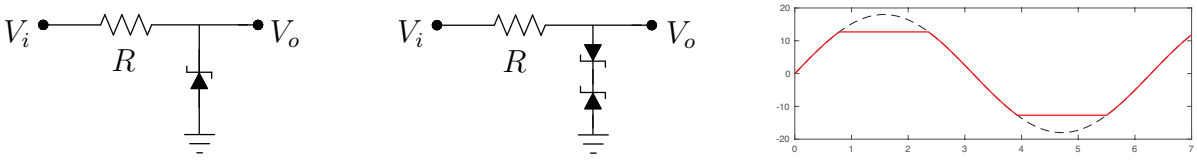


Figure 9.16: Circuits implementing zener diodes to trim (a) flutuating DC and (b) AC inputs $V_i(t)$; (c) the (solid) output $V_o(t)$ of the AC trimming circuit in in response to the (dashed) input $V_i(t) = A\,\sin(\omega t)$, with $A > V_{\text{max}}$.

Regular diodes are denoted by the symbol —▶|—, and both avalanche and Zener diodes by the symbol —▶|— (Zener diodes are much more common), with the arrow pointing in the direction of the current when under forward bias. Real semiconductor diodes are often **axial** (small cylinders with a wire out each end), with the n-doped end marked with a single bar, consistent with the bar at the end of the diode symbol.

To analyze circuits with diodes, assume at any moment that any given diode is in one of two states:

(i) either the circuit outside of the diode of interest is such that it can bring the magnitude of the voltage across the diode just high enough (V_d if under forward bias or, if zener diode, V_{br} if under reverse bias) such that current flows through the diode, or

(ii) it isn't, and the current through that diode is zero.

If case (i), the current through the diode is established by the rest of the circuit. If case (ii), the voltage across the diode is established by the rest of the circuit. Following these simple rules, the following two examples illustrate how to analyze electric circuits with diodes. Note that, in such analyses, both the leakage current and the effective resistance, as discussed above, are typically assumed to be negligible.

Example 9.13 A **full-wave diode bridge rectifier** is illustrated in Figure 9.15a, with the current direction through the bridge indicated. Taking $V_i(t) = A\,\sin(\omega\,t)$, the corresponding voltage across the load, $V_+(t) - V_-(t)$, is illustrated in Figure 9.15b. This output is **rectified**; that is, $V_i(t)$ is converted from AC to a (fluctuating) DC. Substantial low-pass filtering is needed at the output to minimize its voltage fluctuations. Note that, when the current is flowing, it passes through two diodes; thus, if $-2\,V_d \leq V_i \leq 2\,V_d$, no current flows, and $V_+ - V_- = 0$. Noting (9.1), the percentage of power lost in the diodes when current is flowing, $\epsilon(t) = P_{\text{diodes}}/(P_{\text{load}} + P_{\text{diodes}})$, is plotted in Figure 9.15c; if V_d/V_{max} is not small, this loss is substantial. △

Example 9.14 Figure 9.16a illustrates the use of a zener diode to trim a fluctuating DC signal $V_i(t) \geq V_{\text{max}}$ down to a specified value $V_{\text{max}} = V_{\text{br}}$, and Figure 9.16b-c illustrates the use of a pair of zener diodes, installed back to back, to trim (aka "clip") an AC signal $V_i(t) = A\,\sin(\omega\,t)$ down to the range $V_{\text{max}} \leq V_o(t) \leq V_{\text{max}}$, where $V_{\text{max}} = V_{\text{br}} + V_d$. When using zener diodes to trim such signals, it is essential to use a resistor upstream of the zener to limit the magnitude of the current through the zener(s) to $|I_{\text{quiescent}}(t)| = [|V_i(t)| - V_{\text{max}}]/R$; if R is reduced or omitted, the resulting current increases and the associated components may fail. A drawback of this simplistic approach is that, even with no load applied, these circuits waste $P_{\text{lost}} = |V_i \cdot I_{\text{quiescent}}|$ as heat; with a resistive load R_{load} applied at V_o, this power loss increases by $|(V_i - V_o) \cdot I_{\text{load}}|$ where $I_{\text{load}} = V_o/R_{\text{load}}$. △

9.2.2 Bipolar Junction Transistors (BJTs)

A semiconductor crystal designed to behave as an **amplifier** or an **electronically-activated switch** is called a **transistor**, and is built from adjacently doped regions of alternating type. As depicted in Table 9.7, there are essentially eight basic types of transistors, which are all somewhat similar in their application, though they differ considerably in their physical construction and internal operation. The simple and robust **bipolar junction transistor** (**BJT**) was once the most common. A BJT is, in effect, two oppositely-facing diodes placed back-to-back in a single semiconductor crystal. As suggested by their respective names and symbols,

- in a p-n-p type BJT, the emitter-base connection is a p-n diode nominally under forward bias, whereas
- in an n-p-n type BJT, the base-emitter connection is a p-n diode nominally under forward bias.

If the middle section of a BJT, called the **base**, is left unconnected, then (in the ideal setting, assuming no breakdown) there will be zero current between the two ends of the BJT (called the **emitter** and the **collector**), as one of its two p-n junctions will always be under reverse bias. If (in the p-n-p case) a small **emitter** $\rightarrow$ **base** current (or, in the n-p-n case, a small **base** $\rightarrow$ **emitter** current) is initiated, this current populates the central region of the transistor (including the depletion zone in the p-n junction between the base and the collector, which is nominally under reverse bias) with charge carriers, thus causing a much larger current between the emitter and collector to flow, proportional to the current at the base. We denote the voltages and the magnitude of the currents of the emitter, base, and collector as, respectively, $\{V_\mathsf{E}, V_\mathsf{B}, V_\mathsf{C}\}$ and $\{I_\mathsf{E}, I_\mathsf{B}, I_\mathsf{C}\}$; note that, by KCL, $I_\mathsf{E} = I_\mathsf{C} + I_\mathsf{B}$ in both n-p-n and p-n-p type BJTs. Assuming the voltage differences are sufficiently small that breakdown does not set in, the three useful modes of operation of a **n-p-n transistor** depend on where V_B is with respect to V_C and V_E, with $V_C > V_E$ (the p-n-p case is similar, with all inequalities reversed):

- **Saturation** or "**on**" mode: $V_B > V_C > V_E$. Both p-n junctions are forward biased; current flows freely (small effective resistance $R_{CE(\text{on})}$), limited by resisters elsewhere in the circuit. Energetically efficient!
- **Forward active** or "**linear**" mode: $V_\mathsf{C} > V_\mathsf{B} > V_\mathsf{E}$. This is the nominal setting in which the transistor acts as a current amplifier, as used in many audio systems. The **current gain** from I_B to I_C in this mode is denoted h_{FE} or β_F, and is typically about $\beta_\mathsf{F} = I_\mathsf{C}/I_\mathsf{B} \approx 100$, whereas the ratio $I_\mathsf{C}/I_\mathsf{E}$ is denoted α_F, and (since $I_\mathsf{E} = I_\mathsf{C} + I_\mathsf{B}$) is typically about $\alpha_\mathsf{F} = I_\mathsf{C}/I_\mathsf{E} = \beta_\mathsf{F}/(1+\beta_\mathsf{F}) \approx 0.99$. *Not* energetically efficient.
- **Cutoff** mode: $V_C > V_E > V_B$. Both p-n junctions are reverse biased; very little current flows. Efficient!

The fourth mode of an n-p-n transistor, **reverse active** or "**backwards**" mode, given by $V_\mathsf{C} < V_\mathsf{E}$, is the result of installing the transistor backwards. The construction of a BJT (see Table 9.7) is *not* symmetric (notwithstanding introductory explanations of how a BJT functions, that might initially seem to imply the contrary). In particular, the surface area of the p-n junction between the base and the collector needs to be much larger than the surface area of the p-n junction between the base and the emitter for the BJT to perform well.

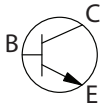
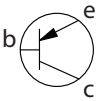
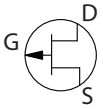
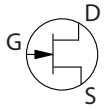
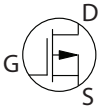
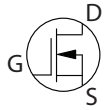
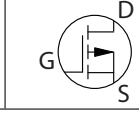
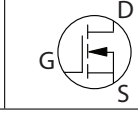
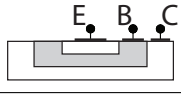
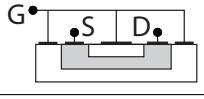
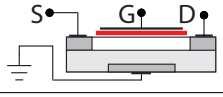
Bipolar Junction Transistor (**BJT**) (see §9.2.2)		Field-Effect Transistor (**FET**) (see §9.2.3)					
		Junction FET (**JFET**) (depletion mode)		Insulated-Gate FET (**IGFET**)			
				depletion mode		enhancement mode	
n-p-n type	p-n-p type	p-channel	n-channel	p-channel	n-channel	p-channel	n-channel
							
							

Table 9.7: Eight types of transistors, their symbols, and some key construction features. The three nodes of a BJT are denoted the **base** (B), **emitter** (E), and **collector** (C), whereas the corresponding nodes of an FET are denoted the **gate** (G), **source** (S), and **drain** (D). The most common type of IGFET is the **MOSFET**.

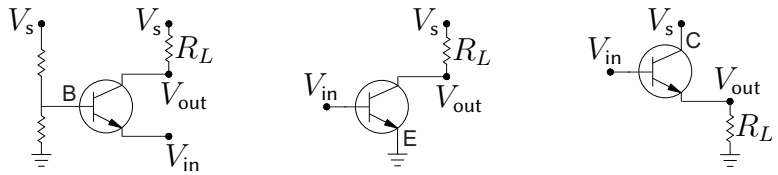


Figure 9.17: (a) **Common base** (CB), (b) **common emitter** (CE), and (c) **common collector** (CC) configurations of an n-p-n BJT. In each, the "common" terminal is held at constant voltage, and the other two terminals are interpreted as the "input" and "output", the latter of which drives the "load", denoted here R_L.

Noting that I_B is typically much smaller than I_C, the power dissipated by a BJT is nearly $P = |V_C - V_E|\, I_E$ [similarly, the power dissipated by a FET is nearly $P = |V_D - V_S|\, I_S$]. A transistor operating in the linear region is *not* power efficient; in audio applications, these diminutive devices may dissipate many watts of power in linear mode, often necessitating heat sinks. On the other hand, when used as a switch (in both saturation and cutoff modes), very little power is dissipated by a transistor (BJT or FET); further, modern transistors (MOSFETs, described below, are by far the most common today) can switch from saturation (full on) to cutoff (full off), or back, in *tens of nanoseconds*. Thus:

Guideline 9.1 *For power-efficient operation of transistors, use them as fast switches rather than amplifiers.*

As mentioned above, the **simplest model of a BJT in forward active mode** is as a current amplifier,

$$I_C = \beta_F\, I_B, \quad I_C = \alpha_F\, I_E \quad \text{with} \quad \alpha_F = \beta_F/(1+\beta_F) \quad \text{and} \quad I_E = I_C + I_B, \tag{9.10a}$$

and the **current gain** β_F (aka h_{FE}) taken as a (large) constant, with $\beta_F \approx 100$ and thus $\alpha_F \approx 0.99$ as typical values (commonly, $50 \leq \beta_F \leq 300$). The more accurate **Early model** of forward active mode models β_F as a function of the magnitude of the collector-emitter voltage $V_{CE} = |V_C - V_E|$ such that

$$\beta_F = (1 + V_{CE}/V_A)\, \beta_{F0} \tag{9.10b}$$

in (9.10a), where the constants V_A and β_{F0} are called the **Early voltage** and the **current gain at zero bias**, respectively. This model may be extended to incorporate the base-emitter voltage $V_{BE} = |V_B - V_E|$ such that

$$I_C = (1 + V_{CE}/V_A)\, (e^{V_{BE}/V_T} - 1)\, I_S \quad \Rightarrow \quad I_C \approx I_S\, e^{V_{BE}/V_T} \text{ if } V_{CE}/V_A \ll 1 \text{ and } V_{BE} \gg V_T. \tag{9.10c}$$

where the constants I_S and V_T are referred to as the **reverse saturation current** and the **thermal voltage**, respectively. Typical constant values are $I_S = 10^{-15}$ to 10^{-12} amps, $V_T = 26$ mV, and $V_A = 15$ V to 150 V. Practically speaking, if I_C is non-negligible, $V_{BE} = 0.7$ V is a good approximation for silicon BJTs.

9.2.2.1 Static characteristics of BJTs

The specifications of various transistor designs vary substantially, and are (usually) documented quite carefully in their datasheets. BJTs are benchmarked in one of three basic configurations, as shown in Figure 9.17:

- **common base** (**CB**), with the emitter driven as the (low-imedance) input, and the collector used as ouput;
- **common emitter** (**CE**), with the base driven as the (high-imedance) input, and the collector used as ouput;
- **common collector** (**CC**), with the base driven as the (high-imedance) input, and the emitter used as ouput.

High impedance inputs (that is, the CE and CC configurations) are quite valuable to isolate (often, delicate) input stages from the (often, power-hungry) output stages that they drive. The CE configuration provides both substantial voltage gain and substantial current gain, and thus the best overall power gain.

It is instructive to model and plot the "typical" characteristics of various types transistors in these basic configurations, although these characteristics vary substantially from one transistor design to the next (check the datasheets!). Some "representative" characteristics of a "typical" general-purpose n-p-n BJT are illustrated in Figure 9.18. Note that the variation of I_C with V_{CE} and I_B does *not* accurately follow (9.10a)-(9.10b) over the range of V_{CE} and I_B illustrated, and that the variation of β_F with both I_C and T_A is significant [representing β_F as a constant, or as simple linear function of V_{CE} only, is only a rough approximation].

Further, as shown in Figure 9.19, modern robust plug-in (3-pin) replacements for single transistors, with similar nominal performance characteristics, incorporate complex internal circuitry that robustly protect the transistor in the event of a host of possible overload conditions, including the exceeding of safe thermal, current, and power (current times voltage) specifications, which would likely cause a simple single transistor to fail. Due to their similar performance to simple BJTs in the vicinity of the load line (where the transistor is normally operated, see next subsection), they may often be used as simple plug-in replacements for single transistors.

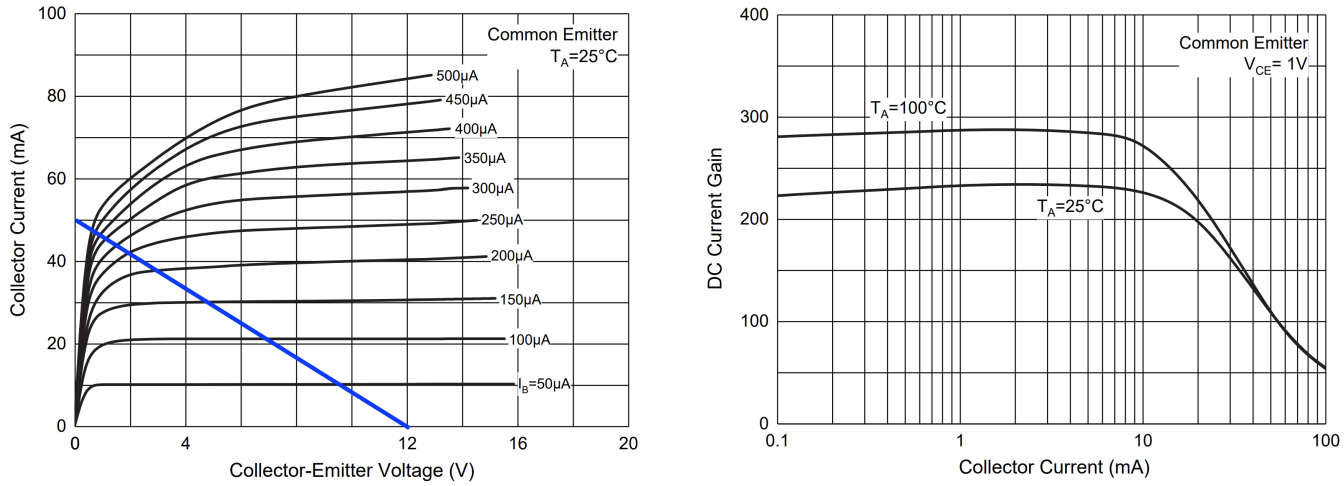


Figure 9.18: Static characteristics of a typical general-purpose n-p-n BJT, the MCC 2N3904. (left) Collector current I_C vs. collector-emitter voltage $V_{CE} = V_C - V_E$ as a function of the base current I_B. (right) DC current gain β_F vs. collector current I_C as a function of the device temperature T_A. The diagonal **load line** connects the $V_{CE,\text{max}}$ point (with $I_C = 0$) to the $I_{C,\text{max}}$ point (with $V_{CE} = 0$), as set up by the supply voltage provided to the circuit, as well as the passive components surrounding the transistor; further explanation of the importance of this line is given in §9.2.2.2 and Example 9.15.

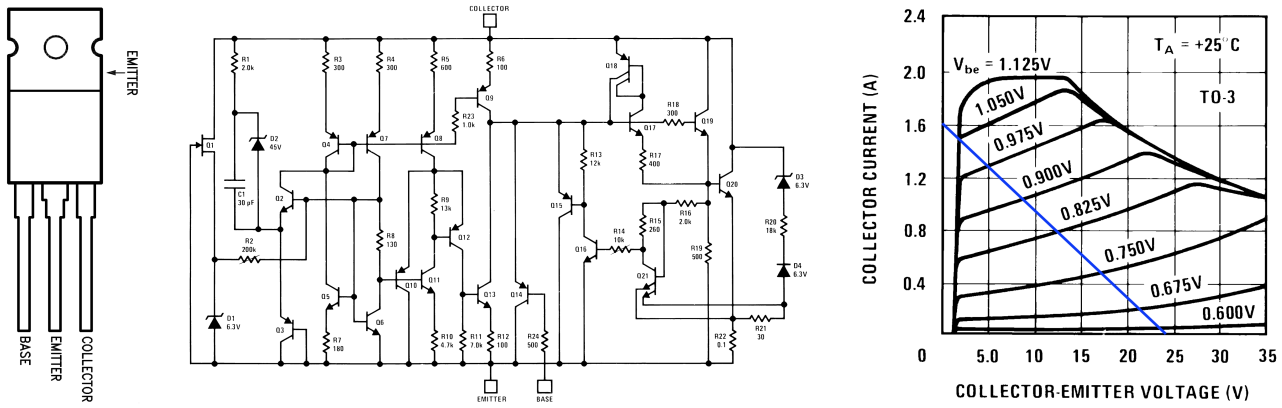


Figure 9.19: A high-reliability replacement for a single n-p-n BJT, the TI LM395. (a) standard 3-pin package, as used by many BJTs. (b) Complex internal wiring, incorporating various forms of protection circuitry. (c) Collector current I_C vs. collector-emitter voltage V_{CE} as a function of the base-emitter current V_{BE}.

9.2.2.2 BJT load line

When implementing BJTs properly into linear amplifier circuits that leverage them, as discussed in Examples 9.15-9.17 below, the gain of the overall amplifier is governed by the connectivity and passives in the circuit surrounding the transistor, and *not* by the specific I_B-to-I_C or V_{BE}-to-I_C characteristics of the transistor itself, as long as β_F is sufficiently large (recall that β_F varies significantly from one transistor to another, and depends nonlinearly on operating temperature, the current at the base, etc.).

The design process for BJT-based Class A amplifiers (see Example 9.15) generally involves drawing a diagonal **load line** (see, e.g., Figures 9.18a and 9.19c) between two extreme operating conditions:

- the maximum possible voltage V_{CE} across the transistor (usually, the supply voltage) when $I_C \approx 0$, and
- the maximum possible current I_C through the transistor when it is "saturated" (that is, when $V_{CE} \approx 0$).

[Again, these conditions are defined by the circuit around the transistor, not by the transistor itself.] The voltage biasing that sets up the nominal conditions at the gate (as illustrated in Figure 9.3c, and discussed further in Example 9.15) is then arranged to operate nominally near the center (referred to as the **Q point**) of the linear part of this V_{CE} to I_C relationship, with variations in the input (I_B or V_{BE}) around this Q point creating an approximately linear response in the current I_C, which in turn (again, due to the circuit surrounding the transistor; see e.g. Figure 9.20a) creates an approximately linear response in the output voltage V_C.

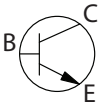
The region to the far left in the collector current I_C vs. collector-emitter voltage V_{CE} plot (see Figures 9.18a and 9.19c, noting where the several curves depicting the voltage at or current through the gate collapse into a single steep line) is sometimes called the **saturation region** or **ohmic region**. In this region, the middle of the transistor is effectively "saturated" with charge carriers, and the transistor acts effectively like a resistor, with a relatively small resistance (that is, $V_{CE} = I_C\, R_{CE(\text{on})}$), independent of the precise conditions at the base.

To the right of the saturation region is the **forward active region**, where (in the n-p-n case) $V_C > V_B > V_E$, and below that is the **cutoff region**, where (approximately) $V_C > V_E > V_B$, as discussed further in §9.2.2.

As described in §9.2.2, the three useful modes of operation of an **n-p-n** BJT (with $V_C > V_E$) are:

- **saturation** (with V_B above both V_C and V_E),
- **linear** (with V_B between V_C and V_E), and
- **cutoff** (with V_B below both V_C and V_E),

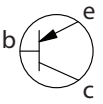
n-p-n BJT:



and the three useful modes of a **p-n-p** BJT (with $V_e > V_c$, switching to lowercase e, b, c for clarity), are:

- **cutoff** (with V_b above both V_e and V_c),
- **linear** (with V_b between V_e and V_c), and
- **saturation** (with V_b below both V_e and V_c).

p-n-p BJT:



There are a variety of ways to build an **amplifier** with such transistors for various applications, including
- the amplification of audio signals (for playing sounds on speakers, including woofers, tweeters, and mids),
- the amplification of the very-low-amplitude signals from analog sensors, like piezos and thermocouples,
- the isolation and amplification of AM or FM radio frequency (**RF**) signals, etc.

Amplification systems may also be built by *cascading* multiple amplifier stages (e.g., a **pre-amplifier** followed by a **power amplifier**). Individual amplifier stages are classified by the number of degrees that their transistors are held in linear mode (i.e., the so-called **conduction angle**) during the amplification of a sinusoidal input:

Class A amplifiers include a single transistor that is held in linear mode for 360 degrees (100% of the time).
Class B amplifiers include two transistors, each held in linear mode for almost 180 degrees (50% of the time)[18].

[18] The Class B approach significantly improves power efficiency as compared to the Class A approach, but often exhibits significant **crossover distortion** during the time in which one transistor is shifting out of cutoff mode, and the other is shifting into cutoff mode, as these processes happen somewhat gradually as the input signal changes.

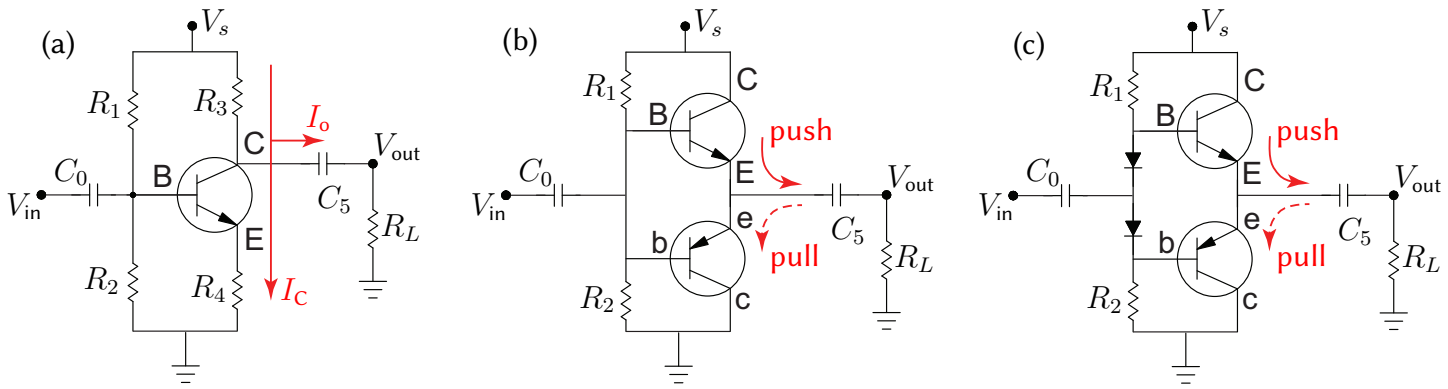


Figure 9.20: Typical embodiments of three different classes of simple BJT-based amplifiers:
(a) Class A, with a single n-p-n transistor with a $360°$ conduction angle.
(b) Class B ("push-pull"), with a matched pair of n-p-n and p-n-p transistors, each with $180°$ conduction angle.
(c) Class AB, with a matched pair of transistors each with conduction angle somewhat greater than $180°$.
Class A amplifiers actually work best as preamplifiers, and Class AB amplifiers work best as power amplifiers.

Class AB amplifiers include two transistors, each held in linear mode for slightly *more* than 180 degrees[19].
Class C amplifiers include two transistors, each held in linear mode for significantly *less* than 180 degrees[20].
Class D amplifiers are built for the amplification of binary or ternary PWM signals.

A brief introduction to representative designs in a few of these classes is given below.

Example 9.15 A representative **Class A amplifier**, built around a single n-p-n BJT, is given in Figure 9.20a. Denoting $\{V_C, V_B, V_E\}$ and $\{I_C, I_B, I_E\}$ as the voltage and current at the collector, base, and emitter of the BJT (positive current to the right and down), noting $I_4 = I_E$, and assuming[21] that the BJT is operating in linear mode 100% of the time under normal operation, the equations governing the BJT in this design are

$$I_{\rm E} = I_{\rm B} + I_{\rm C}, \quad I_{\rm C} = \alpha_{\rm F}\, I_{\rm E}, \quad V_{\rm B} - V_{\rm E} \approx V_d\,; \tag{9.11a}$$

the rest of the component and KCL equations governing this circuit are determined as in previous examples[22]:

$$I_0 = C_0\, d(V_{\rm in} - V_B)/dt, \quad V_s - V_B = I_1 R_1, \quad V_B = I_2 R_2, \quad I_0 + I_1 = I_2 + I_B, \tag{9.11b}$$

$$I_3 = I_{\rm C} + I_5, \quad V_s - V_{\rm C} = I_3 R_3, \quad V_E = I_E R_4, \quad I_5 = C_5\, d(V_C - V_{\rm out})/dt, \quad V_{\rm out} = I_5\, R_L. \tag{9.11c}$$

The full system has 12 eqns in the 12 variables $\{V_{\rm out}, V_C, V_B, V_E, I_C, I_B, I_E, I_0, I_1, I_2, I_3, I_5\}$, plus the input signal $V_{\rm in}$, the source voltage V_s, the transistor constants $\{\alpha_{\rm F}, V_d\}$, and the passives $\{C_0, R_1, R_2, R_3, R_4, R_L\}$. These 12 linear equations are easily Laplace transformed and combined.

For large β_F [and thus $\alpha_F \approx 1$] and appropriate choices for $\{R_1, R_2\}$, $I_B(t)$ is negligible in (9.11b), and the transfer function governing how $V_{\rm out}(s)$ responds to $V_{in}(s)$ essentially decouples into two cascaded parts: (i) one from $V_{in}(s)$ to $V_B(s)$ via $\{C_0, R_1, R_2\}$, (ii) one from $V_B(s)$ to $V_{\rm out}(s)$ via the transistor and $\{R_3, R_4, C_5\}$ [in the present analysis, $V_{\rm out}$ is also routed through a speaker, with characteristic resistance R_L, to ground]. This decoupling provides particular insight; we thus derive these two separate transfer functions here.

[19] The slight overlap of the "on" time of each of the two transistors when amplifying a sinusoidal signal in the Class AB approach is helpful in reducing the crossover distortion exhibited by a pure Class B approach.

[20] A Class C amplifier is often used to excite an LC oscillator that may be tuned to resonate at a particular frequency (e.g., for the modulation or demodulation of an RF transmission). The LC oscillator establishes the underlying sinusoidal nature of the signal; a Class C amplifier on its own (not exciting such an oscillator) would otherwise introduce significant distortion.

[21] Once our analysis of the circuit is established, we will tune the passives in the circuit to assure this assumption is satisfied.

[22] We (roughly) model the speaker here as a purely resistive load with resistance R_L; see Example 9.10 for further discussion.

For part (i) in this large β_F limit, it is seen that $\{C_0, R_1, R_2\}$ amount simply to a **biased first-order high-pass filter with unit gain**, exactly as in Figure 9.3c; this is seen by combining the 4 eqns in (9.11b) together with $I_B \approx 0$, in the 5 variables $\{I_0, I_1, I_2, I_B, V_B\}$, and the input V_{in}, which immediately gives

$$V_{\text{B}}(s) = \frac{R_2\, V_s}{R_1 + R_2 + R_1\, R_2\, C_0\, s} + \frac{R_1\, R_2\, C_0\, s\, V_{\text{in}}(s)}{R_1 + R_2 + R_1\, R_2\, C_0\, s} = \frac{R_2}{R_1 + R_2}\, \frac{\omega_{\text{i}}}{s + \omega_{\text{i}}}\, V_s + \frac{s}{s + \omega_{\text{i}}}\, V_{\text{in}}(s),$$

where $R_{\text{i}} = R_1 R_2/(R_1 + R_2)$ [that is, $1/R_{\text{i}} = 1/R_1 + 1/R_2$] and the corner frequency is $\omega_{\text{i}} = 1/(R_{\text{i}}\, C_0)$. The voltage divider biasing drives the steady component of $V_B(t)$ towards $V_Q = V_s\, R_2/(R_1 + R_2)$ [instead of towards zero, as done by the filter in Figure 9.3b, which effectively takes $R_1 \to \infty$ in Figure 9.3c].

For part (ii) in the large β_F limit, it is found that the transistor (assumed to be operating in the linear mode) and $\{R_3, R_4, C_5\}$ amount simply to another first-order high-pass filter with negative gain; this is seen by combining the 8 eqns in (9.11a) and (9.11c), in the 8 variables $\{I_C, I_B, I_E, V_C, V_E, I_3, I_5, V_{\text{out}}\}$, together with the input $V_B(t)$, which immediately (see code in RR.ch09) gives:

$$V_{\text{out}}(s) = \frac{C_5\, R_L\, s}{(R_3 + R_L)\, C_5\, s + 1}\, V_s - \alpha_F\, \frac{R_3}{R_4}\, \frac{R_L\, C_5\, s}{(R_3 + R_L)\, C_5\, s + 1}\, (V_B - V_d) = V_{\text{ii}}(s) + K\, \frac{s}{s + \omega_{\text{ii}}}\, V_B(s)$$

where $K = -\alpha_{\text{F}}\, (R_3/R_4)\, [R_L/(R_3 + R_L)]$, the corner frequency of the high-pass filter on $V_B(t)$ is $\omega_{\text{ii}} = 1/(R_{ii}\, C_5)$ where $R_{\text{ii}} = R_3 + R_L$, and the term $V_{\text{ii}}(s)$ is driven solely by the voltages V_s and V_d, and quickly approaches a constant. Note again that K is negative (that is, this amplifier is inverting); for audio signals, that doesn't really matter. In the case that the current through the load resistor is negligible (i.e., $R_L \to \infty$; e.g., if the speaker is removed, and replaced with a power amplifier stage with high input impedance), $K \approx -\alpha_{\text{F}}\, (R_3/R_4)$ and $\omega_{\text{ii}} \to 0$ [i.e., all frequencies are passed through part (ii)]; in this case, C_5 may be removed. Conveniently, taking R_3 as a potentiometer (potentially... one that goes all the way to 11), the voltage gain of this amplifier is adjustable, making it particularly well suited as a (highly linear) preamplifier stage.

When the input signal is time varying, the following two extreme cases are of interest:

• when V_B is near its maximum, the transistor is nearly saturated, $V_{CE} \approx 0$, and thus $I_{C,\text{max}} \approx V_s/(R_3 + R_4)$;
• when V_B is near its minimum, the transistor is nearly in cutoff, and $I_{C,\text{max}} \approx 0$.

When the input signal is constant (i.e., with the amplifier operating near its Q point, as discussed in §9.2.2.2), the transistor is near the center of its linear active region, and substantial current flows, $I_{C,\text{quiescent}} \approx I_{C,\text{max}}/2$. As a result, Class A amplifier designs, though accurately linear in behavior, are generally quite inefficient.

Consider an application with the silicon BJT in Figure 9.18, with $V_d = 0.7\,V$ and $100 < \beta_F < 285$ depending on both I_C and the ambient temperature (thus, $\alpha_F \approx 1$ in the calculations below), with an output connected to a (high-impedance) input on a power amplifier ($R_L \to \infty$), and with $V_s = 12\,\text{V}$:

• For part ii, design targets of $|K| = 10 = R_3/R_4$ and $I_{C,\text{max}} \approx 0.05\,\text{A}$ (see load line in Figure 9.18a), and thus $R_3 + R_4 = V_s/I_{C,\text{max}} \approx 240$ ohm, leads to $\{R_3, R_4\} \approx \{218.4\,\text{ohm}, 21.6\,\text{ohm}\}$, and C_5 can be removed.

• For part i, taking $I_{C,\text{quiescent}} \approx I_{C,\text{max}}/2 = 0.025\,\text{A}$ at the Q point, we have $V_E \approx I_{C,\text{quiescent}} \cdot R_4 = 0.54\,\text{V}$ and $V_B = V_E + V_d = 1.24\,\text{V}$. By Figure 9.18a, at the Q point indicated (assuming the device is at 25°C), the base current is[23] $I_B \approx 120e{-}6\,\text{A}$. At this steady ($I_{\text{in}} = I_0 = 0$) condition, we take small currents in part i by setting, say, $I_1 = 35 \cdot I_B = 0.0042\,\text{A}$; together with the KCL condition $I_0 + I_1 = I_2 + I_B$ and $V = IR$ across each resistor gives $\{R_1, R_2\} \approx \{2491, 295\}$ ohm (see code in RR.ch09). A final design target (appropos of an audio preamp) of $f_i = 1/(R_i\,C_0) \approx 10$ Hz where $R_i = R_1\,R_2/(R_1 + R_2)$ [see Example 9.2] gives $C_0 \approx 22\,mF$. △

Example 9.16 A representative **Class B amplifier**, built around a matched pair of n-p-n and p-n-p BJTs, is given in Figure 9.20b. The $\{C_5, R_1, R_2\}$ components, with $R_1 = R_2$, again provide high-pass filtering with voltage divider biasing at the input stage, the output of which is applied to the base of both the (upper) n-p-n BJT and the (lower) p-n-p BJT. Note also that the emitters of both BJTs are tied together, and that the C_6 component into R_L at the output stage again provides high-pass filtering which eliminates the DC ouput current.

With this simple "push-pull" configuration, the base voltage $V_B = V_b$ is either:

(a) above $V_E = V_e$, in which case the n-p-n BJT is in linear active mode and the p-n-p BJT is in cutoff, or
(b) below $V_E = V_e$, in which case the p-n-p BJT is in linear active mode and the n-p-n BJT is in cutoff.

Thus, in this basic Class B design, either one of these BJTs is "on", and amplifying the input in accordance with the model $I_C \approx I_S\ e^{V_{BE}/V_T}$ [see (9.10c)], or the other BJT is on; they are never both "on" simultaneously.

Looking a bit more closely [see, in particular, Figure 9.14], it actually takes the V_{BE} junction to be slightly *above* a small cut-in voltage V_d (in silicon, $V_d \approx 0.7\,\text{V}$) before the n-p-n BJT enters linear active mode, and it takes the V_{eb} junction to be slightly *below* $-V_d$ before the p-n-p BJT enters linear active mode. As a result, there is a range of (small) inputs V_{in}, referred to as a "dead zone", for which both BJTs remain in cutoff, and the output current is zero. The resulting lack of linearity in the response, which is especially pronounced for small input signals, is referred to as **crossover distortion**. As both BJTs are actually in cutoff for zero (or, small) inputs, however, Class B amplifier designs are generally much more efficient than Class A amplifier designs.

Despite the crossover distortion mentioned above, this simple "push-pull" configuration is useful in applications in which a linear response is not especially important; a typical example[24] is the output stage of a power op amp, such as the TI ALM2402-Q1 included in the Berets, as discussed in §5. △

Example 9.17 A **Class AB amplifier**, as illustrated in Figure 9.20c, is a tweak on the Class B amplifier design which adjusts the bias V_B to be about $2\,V_d \approx 1.4\,\text{V}$ above V_b. This has the effect of effectively eliminating the dead zone of the Class B amplifier mentioned previously. As the transition between cutoff and active mode of

[23] For the value of I_1 used, this value of I_B is negligible; the result thus applies even if this device is at a different temperature.

[24] Note that op amps (see §9.3) are characterized by very high gain, and are invariably used in *feedback* configurations, and thus their precise gain characteristics are actually rather unimportant.

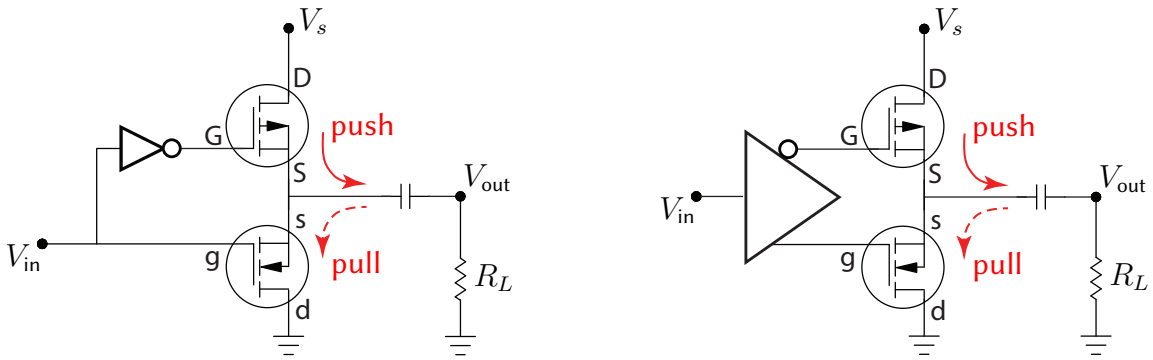


Figure 9.21: (a) A schematic representation of a Class D amplifier in which, at any given time, one transistor is "on" and one transistor is "off". Like the Class B amplifier design in Figure 9.20b, this circuit may be characterized as incorporating a "push-pull" architecture for driving the output. (b) A perhaps more accurate representation of a Class D amplifier, in which the control electronics carefully incorporate a dead time between when one transistor turns off and the other transistor turns on, in order to avoid shoot through.

the transistors is actually somewhat gradual, diodes are usually selected, carefully, with slightly larger V_d than base-emitter junctions of the two transistors, thus slightly overlapping the regions in which the two diodes are on (that is, increasing their conduction angles to somewhat more than 180 degrees), in an effort to minimize crossover distortion, at the cost of a slight reduction in efficiency. △

Example 9.18 A **Class D amplifier**, as illustrated schematically in Figure 9.21a, is a straightforward tweak on the Class B ("push-pull") amplifier design in Figure 9.20b, in which the two transistors that it incorporates are driven as fast on/off switches (often, modern MOSFETs are used, which are remarkably efficient when functioning in this capacity). Typically, binary input logic is used, with a logical PWM input signal at the gate of one of the transistors [transitioning quickly between a "logical low" ("transistor off") state and "logical high" ("transistor on") state, at an adjustable duty cycle], and the NOT of this logical signal (see §1.1.2) used at the gate of the other transistor, thus effectively turning one transistor on and the other transistor off at any given moment. Often, a very high PWM frequency is used (say, around 1 MHz), with a second-order low-pass LC filter (see Figure 9.3d) incorporated at the output (with, say a cut-off frequency of 100 kHz) to substantially attenuate the oscillations at the PWM frequency and its higher harmonics, audio signals (at frequencies from 20 Hz up to 20 kHz) can even be amplified cleanly with with approach, with remarkably high efficiency. Setting the PWM at 50% duty cycle, and low-pass filtering the output, holds the output constant at $V_s/2$; modulating the PWM duty cycle then results in very high signal fidelity at V_{out}, with very efficient current amplification.

When actually implementing the control electronics driving the transistors for such an amplifier, as illustrated perhaps more realistically in Figure 9.21b, particular care needs to be incorporated to avoid the condition in which both transistors are in the "on" state at the same time, even if just for a moment. This creates a pathway straight from power to ground through the two transistors, and results in large current spikes, called **shoot through**, that can easily break the amplifier. To avoid this condition, a certain **dead time** needs to be built in, between when one transistor turns off and the other transistor turns on. The calculation of how long this dead time needs to be in order to eliminate the possibility of shoot through should account accurately for the manufacturing tolerances of the devices to be incorporated.

Three state (ternary) logic can also be incorporated in Class D amplifiers, with the control electronics turning both transistors "off", allowing V_{out} to "float", when a "high impedance" state (that is, neither logical high nor logical low) is detected at V_{in}. △

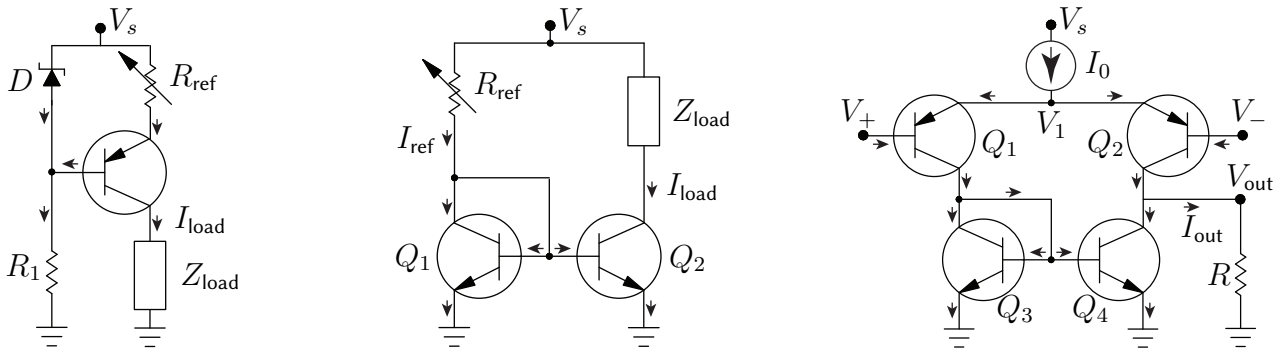


Figure 9.22: Simple transistor circuits. (a) A **current source** based on a p-n-p BJT and a Zener diode, with $I_{\text{load}} \approx (V_{\text{br}} - V_{\text{eb}})/R_{\text{ref}}$. (b) A **current mirror** based on two n-p-n BJTs, with $I_{\text{load}} \approx I_{\text{ref}} = (V_s - V_{\text{BE}})/R_{\text{ref}}$. (c) A **differential amplifier** based on four BJTs and a current source, as in (a), with $V_{\text{out}} \approx D(V_+ - V_-)$ where $D = I_{\text{S}}\, R/V_{\text{T}}$; note that the lower half of this circuit is exactly the current mirror considered in (b).

Other amplifier designs, including minor variations of these typical embodiments, are easily found online. We now present a handful of other useful transistor-based circuit designs.

Example 9.19 Current source. Consider first the circuit in Figure 9.22a. As discussed previously (see Figure 9.14), a Zener diode under a sufficiently large reverse bias has an essentially constant voltage drop across it, V_{br}, regardless of the current flowing through it; note that the resistor R_1 in this circuit limits this Zener diode current. The Zener breakdown voltage V_{br} is also applied across the series connection of the resistor R_{ref} and the emitter-base terminals of the transistor in this circuit; the voltage across resister R_{ref} is thus given by $V_{\text{br}} - V_{\text{eb}}$, where V_{eb} is the voltage drop between the emitter and base of the transistor (about 0.7V for silicon). The emitter current of the transistor is thus given by $I_{\text{e}} = (V_{\text{br}} - V_{\text{eb}})/R_{\text{ref}}$; since V_{br} and V_{eb} are approximately constant, I_{e} is approximately constant. Finally, since a BJT acts as a current amplifier with $I_{\text{b}} = \beta_{\text{F}}\, I_{\text{c}}$ where $\beta_{\text{F}} \approx 100$, it follows that $I_{\text{e}} \approx I_{\text{c}} = I_{\text{load}}$ regardless of the precise values of V_s and β_{F}, provided they are sufficiently large, and regardless of the precise values of R_1 and $|Z_{\text{load}}|$, provided they are sufficiently small. △

Example 9.20 Current mirror. Consider the circuit in Figure 9.22b. Assuming $V_{\text{CE}} \ll V_{\text{A}}$ and thus $\beta_{\text{F}} \approx \beta_{\text{F0}}$ in (9.10b), which is often a good assumption, it follows from (9.10c) that I_{C} of transistor Q_1 is related (exponentially) to V_{BE} such that $I_{\text{C}} = I_{\text{S}}\, e^{V_{\text{BE}}/V_{\text{T}}}$. Since the base-emitter voltage of the (matched) transistors Q_1 and Q_2 are precisely equal in this circuit, their collector currents are equal as well. Finally, since the base currents are negligible compared with the collector currents, it follows from KCL that $I_{\text{load}} \approx I_{\text{ref}}$. △

Example 9.21 Differential amplifier. Consider the circuit in Figure 9.22c. Let $\{V_{\text{E}_k}, V_{\text{B}_k}, V_{\text{C}_k}\}$ and $\{I_{\text{E}_k}, I_{\text{B}_k}, I_{\text{C}_k}\}$ denote the voltage and current of the emitter, base, and collector, respectively, of transistor Q_k, with positive current in the directions indicated in the figure. Due to the current mirror in the lower half of the circuit (see Example 9.20 and Figure 9.22b), $I_{\text{C}_1} \approx I_{\text{C}_4}$. Taking $V_{\text{CE}}/V_{\text{A}} \ll 1$ in (9.10c), and assuming that both $(V_1 - V_+)/V_{\text{T}} \ll 1$ and $(V_1 - V_-)/V_{\text{T}} \ll 1$, noting that $e^{\epsilon} \approx 1 + \epsilon$ for $\epsilon \ll 1$, it follows that

$$\left.\begin{aligned} I_{\text{c}_1} &= I_{\text{S}}\,(e^{(V_1 - V_+)/V_{\text{T}}} - 1) = \alpha_{\text{F}}\, I_{\text{e}_1} \\ I_{\text{c}_2} &= I_{\text{S}}\,(e^{(V_1 - V_-)/V_{\text{T}}} - 1) = \alpha_{\text{F}}\, I_{\text{e}_2} \\ I_0 &= I_{\text{e}_2} + I_{\text{e}_1} \\ I_{\text{out}} &= I_{\text{c}_2} - I_{\text{C}_4} \approx I_{\text{c}_2} - I_{\text{c}_1} \end{aligned}\right\} \Rightarrow \begin{aligned} I_0 &\approx \frac{I_{\text{S}}}{\alpha_{\text{F}}}\Big(\frac{V_1 - V_-}{V_{\text{T}}} + \frac{V_1 - V_+}{V_{\text{T}}}\Big) \;\Rightarrow\; V_1 \approx \frac{1}{2}\Big[\frac{V_{\text{T}}\,\alpha_{\text{F}}\, I_0}{I_{\text{S}}} + V_+ + V_-\Big], \\ I_{\text{out}} &\approx I_{\text{S}}\Big(\frac{V_1 - V_-}{V_{\text{T}}} - \frac{V_1 - V_+}{V_{\text{T}}}\Big) = \frac{I_{\text{S}}}{V_{\text{T}}}\Big(V_+ - V_-\Big), \\ V_{\text{out}} &= I_{\text{out}}\, R \approx D(V_+ - V_-) \quad \text{with} \quad D = I_{\text{S}}\, R/V_{\text{T}}. \end{aligned}$$

Taking V_+ and V_- as the inputs, note that V_{out} responds primarily to the **differential voltage** $(V_+ - V_-)$, while $V_1 = V_{\text{e}_1} = V_{\text{e}_2}$ "floats" up and down in response to the **common voltage** $(V_+ + V_-)$. △

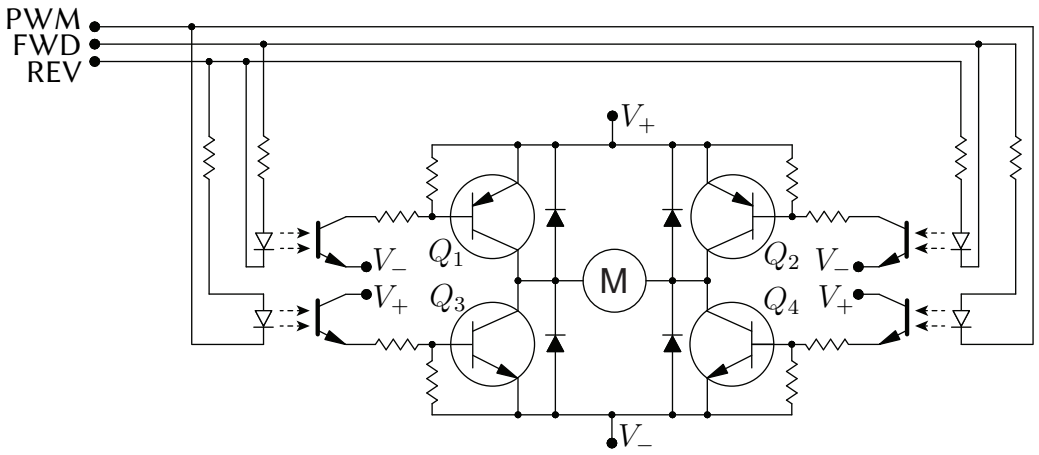


Figure 9.23: Representative H-bridge circuit for efficient bidirectional operation of a BDC motor at partial power via a PWM signal and two GPIOs. All three of these MCU output signals are electrically isolated from the power electronics via **optoisolators**, which are simply LEDs packaged in close proximity with photodiodes.

Example 9.22 H-bridges for driving Brushed DC motors. Brushed DC (BDC) motors are remarkably inexpensive and efficient for converting electrical power to mechanical (rotatary) power. They do not, however, operate effectively at low voltage, due to stiction (that is, dry friction) acting within the motor (see, e.g., the BDC motor model developed in §??), thus motivating a PWM-based control approach (see §4.2.3). They also have significant inductance, as they contain coils of wires, wrapped around ferromagnetic cores, acting as electromagnets; PWM approaches must therefore be implemented with significant caution.

A representative **H-bridge** implementing a PWM-based solution for driving BDC motors at partial power, in a bidirectional fashion, is shown in Figure 9.23. Such H-bridges implement **flyback diodes** to ensure that the voltages at the motor terminals effectively remain between $V_- - V_{\text{br}}$ and $V_+ + V_{\text{br}}$, thus preventing the generation of sparks even when driving motors with substantial inductance. The H-bridge circuit shown in Figure 9.23 operates in four distinct modes, as illustrated in Figure 9.24, based on the FWD and REV logic states:

- **Forward drive/brake** (FWD = 1, REV = 0). In this mode, Q_1 is on, Q_4 is on the same percentage of time that the PWM signal is low, and Q_2 and Q_3 are off. DC power is thus provided from left to right across the motor at a duty cycle set by the PWM. When the PWM signal is high, Q_4 is off, and the current recirculates in the top loop of the bridge through the flyback diode next to Q_2, and the motor "brakes" (see below).
- **Reverse drive/brake** (FWD = 0, REV = 1). In this mode, Q_2 is on, Q_3 is on the same percentage of time that the PWM signal is low, and Q_1 and Q_4 are off. DC power is thus provided from right to left across the motor at a duty cycle set by the PWM. When the PWM signal is high, Q_3 is off, and the current recirculates in the top loop of the bridge through the flyback diode next to Q_1, and again the motor "brakes".
- **Brake/coast** (FWD = 1, REV = 1). In this mode, Q_3 and Q_4 are on the same percentage of time that the PWM signal is low, and Q_1 and Q_2 are off. The braking mode essentially shorts together the two terminals of the motor at a duty cycle set by the PWM; this braking action effectively negates the back emf otherwise generated by the free rotation of the motor, thus causing the motor to slow down. When the PWM signal is high, all four transistors are off, and the motor "coasts" (see below).
- **Coast** (FWD = 0, REV = 0). In this mode, Q_1, Q_2, Q_3 and Q_4 are all off, regardless of the PWM signal. No extra torque is generated by the motor. If the motor momentum and load are such that a significant current is generated in one direction through the motor or the other, this current is sustained by pulling current up from V_- through a flyback diode on one side of the bridge, and pushing this current up through a flyback diode to V_+ on the other side of the bridge, thereby converting mechanical power to electrical power, storing energy back in the battery. Coast mode can thus also be considered as **regenerative braking**.

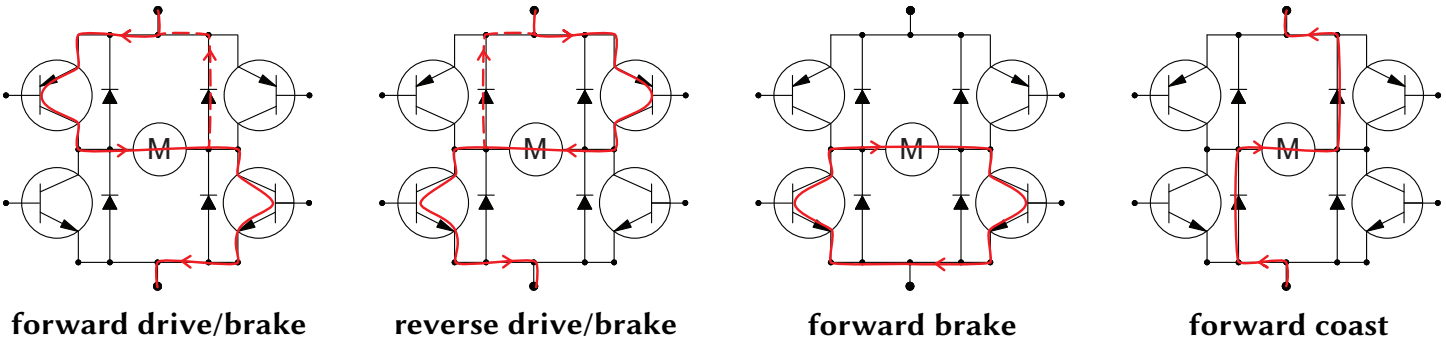


Figure 9.24: Current flow in the four modes of the H-bridge circuit illustrated in Figure 9.23. Note that the use of **high-speed** flyback diodes, which turn on quickly when put under forward bias, are essential in order to keep the voltage at the motor terminals in a limited range, thus preventing the buildup of large voltages and the generation of sparks during the PWM cycle when operating, particularly for motors with large inductance.

In practice, the use of ICs implementing an entire H-bridge circuit are often convenient, such as the Toshiba TB6612FNG, which implements the H-bridge illustrated in Figure 9.23, using MOSFETs instead of BJTs to increase efficiency.

The condition that must absolutely be avoided at all times is both transistors on the same side of an H-bridge being open at the same time, which leads to a short circuit between power and ground. The circuit of Figure 9.23 inherently prevents such a short condition from happening. Other more flexible circuit designs, with more independent control of the four transistors in the bridge, require the appropriate controller logic to be implemented to prevent a short condition. Dedicated motor driver ICs reliably implement such logic.

One feature that other H-bridge circuits (and the motor driver ICs that implement them) can facilitate is the implementation of **drive/coast** modes as an alternative to **drive/brake** modes; that is, the use of "coast" (aka "regenerative braking") during the "off" phase of the PWM when driving, which is energetically more efficient than drive/brake. Some of the more flexible dedicated motor driver ICs, like the remarkable TI DRV8912-Q1 motor drivers implemented by the Berets (see §5.3), internally generate their own PWMs, and allow the user to select between the drive/brake and the drive/coast modes of operation.

Note finally that, as BDC motors sometimes draw very substantial current, the voltage drop across the flyback diodes can be associated with significant power loss and heat generation. Advanced motor driver ICs, like the one mentioned in the previous paragraph, thus turn on the corresponding transistor (which has a much lower effective resistance, aka $R_{DS(on)}$, than a regular diode) whenever it is detected that current would otherwise flow through a flyback diode, thereby substantially reducing the associated power loss and corresponding heat generation in the device. △

9.2.3 Field Effect Transistors (FETs)

In a manner analogous to BJTs, the main flow of current in a **Field-Effect Transistor** (**FET**), between the **drain** and the **source**, is regulated by the voltage at the **gate** (relative to that at the source). The main distinction between an FET and a BJT is that the drain-source current of a FET is regulated by the *voltage* at the gate, whereas the emitter-collector current of a BJT is regulated by the *current* through the base. FETs come in two main types, Junction FETs and Insulated-Gate FETs.

A **Junction Field-Effect Transistor** (**JFET**) is a (often, essentially symmetric) transistor design in which the source and drain are connected to the two ends of a single semiconductor channel that is a either n-doped or p-doped. As indicated Table 9.7, adjacent to the channel are oppositely-doped semiconductor regions connected to the gate. A JFET operates in what is known as **depletion mode**: if the gate of the JFET is left disconnected, the (n-doped or p-doped) channel of the JFET readily conducts current from the source to the drain, or from the drain to the source. However, if a voltage is applied to the gate of the appropriate sign such that the p-n junctions along the edge of the channel are reverse-biased, a depletion zone forms in the channel which diminishes the amount of current the JFET channel can conduct between the source and the drain. Increasing the magnitude of the voltage applied to the gate increases the size of this depletion zone, which further diminishes the current that the JFET can conduct between the source and the drain, until a **pinch-off** level is reached, in which the current the JFET can conduct between the source and the drain is essentially reduced to zero.

In an **Insulated-Gate Field Effect Transistor** (**IGFET**), the gate is *electrically insulated* from the channel carrying the current between the source and the drain. Again, the semiconductor channel in an IGFET is either n-doped or p-doped. The most common type of IGFET today by far, in which the gate insulation (indicated in red in Table 9.7) is a metal oxide, is known as a **Metal Oxide Semiconductor Field-Effect Transistor** (**MOSFET**). Due to the insulation of the gate, an IGFET has a *very high input impedance*, with almost zero current flowing through the gate. This makes IGFETs particularly efficient in both logic circuits and power electronics as fast switches; however, the gate insulation of a MOSFET is generally susceptible to damage from static electricity. IGFETs come in two classes, those that work in a **depletion mode** similar to that of a JFET as described above, and those that work in an **enhancement mode**, in which the channel (the region directly below the red insulation in Table 9.7) between the source and the drain is generally nonconducting (it may even be undoped) until a sufficiently large voltage is applied between the gate and ground, which populates the channel between the source and drain with charge carriers, thus enabling current to flow [in power electronics, this mode is often the safest, as current does not flow until conditions at the gate enable it].

Though the first working transistor was demonstrated back in 1947, the technology of transistors designed for various different purposes is still evolving rapidly today. In particular, for applications in **digital logic** (CPUs, GPUs, DSPs, etc.), designed for fast power-efficient numerical computation, transistors continue to be shrunk in size, packed more densely together, and reduced in operating voltage (thus improving power efficiency). Such ICs are developed with photolithography fabrication techniques leveraging short wavelength **DUV** (deep ultraviolet, ∼200 nm) and **EUV** (extreme ultraviolet, 13.5 nm) light sources. Using so-called "3 nm" processes, densities of well over 200 MTr/mm^2 (million transistors per mm^2) are now possible in large-scale chip fabrication. Transistor counts in modern high-end microprocessors and GPUs (graphics processing units) are in the tens of billions. In contrast, microcontrollers (MCUs) incorporate *much* more streamlined CPUs (for example, an ARM Cortex M0 has about 12k gates, and thus about 72k transistors); the majority of the transistors on an MCU are associated with cache and memory (as a datapoint, a 64 KiB cache has about 3.1M transistors).

For applications in **power electronics**, transistor technology is also still evolving quickly. Four transistor metrics of interest in high-power applications are particularly important:

• The **effective resistance** of a MOSFET when operating in the "on" state, aka $R_{DS(on)}$. In addition to wasting less power (and thus being able to run on a given battery charge for a bit longer), even more significant in many power protection (reverse-voltage, over-voltage, etc) applications is that reduced values of $R_{DS(on)}$ on the power

MOSFET imply much less waste heat generated under normal operating conditions that must be dissipated (on the PCB, and/or with a heat sink), thereby facilitating a higher density integration of power components on the PCB. MOSFETs with values of $R_{DS(on)}$ in the neighborhood of 1 mΩ (‼) are now common.

• The **time delay** from the on state to the off state of the transistor when the gate is triggered, aka $t_{d(off)}$. This measure characterizes how quickly a power protection MOSFET can turn the power off to a system when a fault condition occurs. MOSFETs with values of $t_{d(off)}$ in the neighborhood of 20 to 40 ns (‼) are now common.

• The **rise time** t_r and **fall time** t_f associated with repeated fast switching of the transistor between the on and off states. Many MOSFETs are commonly used for switching power to a device (e.g., an LED) on and off thousands of times a second, an approach referred to as **pulse width modulation** (**PWM**; see §4.2.3). Averaged over several cycles (e.g., by the human eye), this makes the device look as if it is running at partial power (proportional to the fraction of time that the transistor is in the on state, referred to as the **duty cycle** of the PWM signal). PWM-based driving strategies will be used in multiple circuits discussed in the remainder of this chapter. As discussed previously (see Guideline 9.1), a transistor is generally quite efficient in both the on state and the off state; the time spent switching between these two states is where most of the inefficiencies lie. Thus, at a given switching frequency, the faster the switching time, the more efficient this PWM approach is. MOSFETs with values of t_r and t_f in the neighborhood of 10 to 20 ns (‼) are now common.

Examples of modern high-performance MOSFETs include the TI CSD18510Q5B and the CSD88584Q5DC, as used for power protection and BLDC motor control, respectively, in the Beret family of boards (see §5).

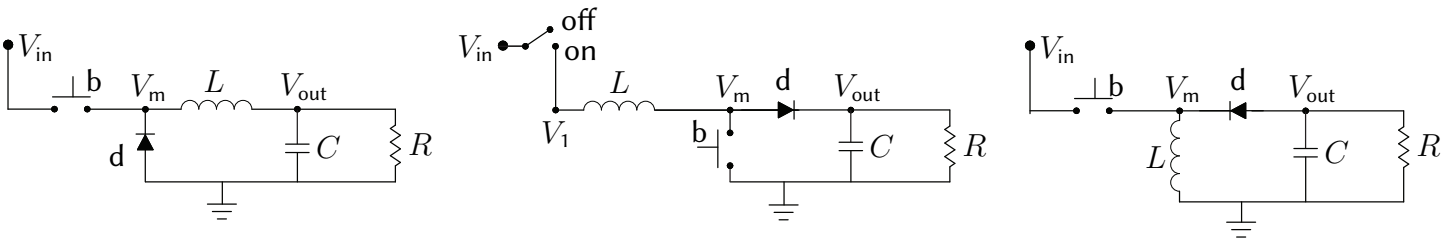


Figure 9.25: Essential components of (left) a **buck** converter, as considered in Example 9.23, (center) a **boost** converter, as considered in Example 9.24, and (right) a **buck-boost** converter, as considered in Example 9.25.

9.2.4 DC-to-DC voltage conversion

Example 9.23 Buck converters. A buck converters is used for DC-to-DC voltage conversion, to step from one voltage (e.g., from a battery) *down* to a well-regulated lower voltage, which is a common problem in electromechanical systems. The essential components of a buck converter are illustrated in Figure 9.25a.

Assume a square wave at V_m, formed by repeated pressing of the button[25] b, with current flowing from V_{in} (through the button) when it is pressed, and current flowing from ground (through the diode) when it is not. Defining $\omega_c = 1/\sqrt{L\,C}$, the output V_{out} when the button is pressed may be determined as follows:

$$V_m - V_{out} = L\,\frac{dI_L}{dt}, \quad I_C = C\,\frac{dV_{out}}{dt}, \quad V_{out} = I_R\,R, \quad I_L = I_C + I_R \quad \Rightarrow \quad \frac{V_{out}(s)}{V_m(s)} = \frac{\omega_c^2}{s^2 + s/(R\,C) + \omega_c^2}.$$

Thus, regardless of the precise value of the load R, if L and C are selected to be large enough that $\omega^2 \gg \omega_c^2$, where ω is the frequency of square wave at V_m, then the fundamental and higher harmonics of this square wave will be significantly damped by the unit-gain second-order low-pass LC filter (see Example 9.2) in the buck converter, leaving at V_{out} only the average value of V_m, which is simply V_{in} times the duty cycle D (where $0 \le D \le 1$) of the square wave at V_m, plus a small essentially sinusoidal ripple at the PWM frequency.

In implementation, rather than running at a fixed duty cycle, *feedback* can be implemented to identify and implement the duty cycle D required to achieve a desired value of V_{out}. This may be achieved, e.g., by implementing a TI TPS56637, which is compact (less than 9 mm^2), easy to hook up (SMT, 10 pins), and inexpensive (less than \$1.50). The (simple) external wiring and (complex) internal circuitry diagrams for the TPS56637 (which generates the required PWM signal with a few op amps) are illustrated in Figure 9.26; note that the button b is replaced by a PWM-actuated MOSFET-driven class D amplifier (shown near the SW pin in the internal circuitry diagram), as discussed further in Example 9.32. The output $V_{out}(t)$ may then be hooked to a load of a few hundred ohms (or more), and this circuit will drive the tens of mA (or less) necessary to drive the load.

Example 9.24 Boost converters. A boost converter is also used for DC-to-DC voltage conversion, but to step from a given V_{in} *up* to a higher output voltage V_{out}. Its essential components are illustrated in Figure 9.25b.

Startup. For the purpose of analysis, assume the circuit is initialized with the switch at left in the **off** position, and with $V_1 = V_m = V_{out} = 0$. At time $t = 0$, the switch at left is moved to the **on** position, thus setting $V_1 = V_{in}$ (the voltage of the source), and the button b is left open as shown, so the current through the diode is I_L.

The eqns for the R, C, and L components, and the KCL eqn at the V_{out} node, may be written

$$V_{out}(t) = R\,I_R(t), \quad I_C(t) = C\,d[V_{out}(t)]/dt, \quad V_1(t) - V_m(t) = L\,d[I_L(t)]dt, \quad I_L(t) = I_C(t) + I_R(t). \quad (9.12)$$

[25] In application (see Figure 9.26), the button is replaced by a PWM signal driving a Class D amplifier (see Example 9.32).

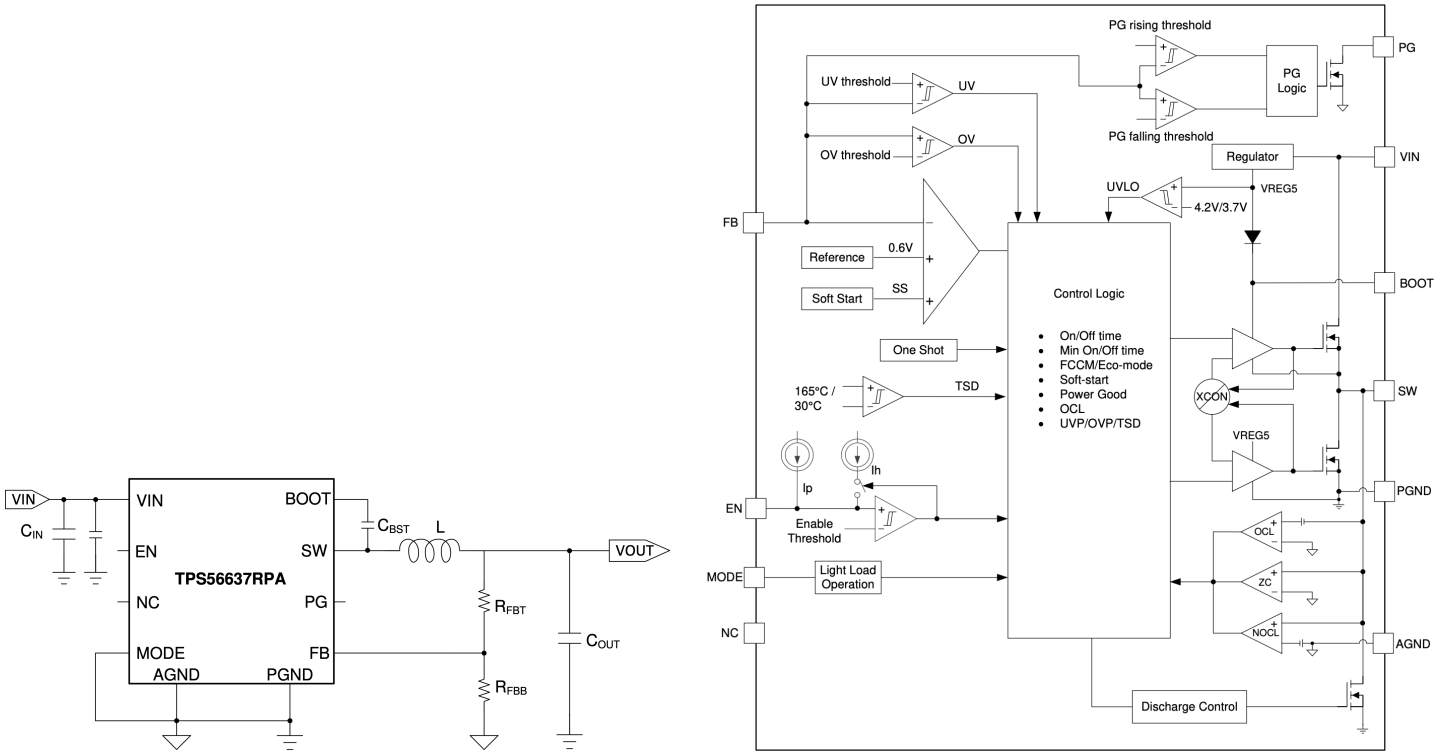


Figure 9.26: The (left) simple external wiring and (right) complex internal circuitry of the TI TPS56637 buck converter IC. In the external wiring diagram, the discrete components marked L and C_{OUT} correspond to the L and C components in Figure 9.25a, and the discrete resistors marked R_{FBT} and R_{FBB} form a voltage divider (see Example 9.1) that generate $V_{FB} = V_{OUT} \cdot R_{FBB}/(R_{FBT} + R_{FBB})$ at the FB pin on the IC, which is compared to an (internally-generated, with a zener diode) $V_{FB,\,ref} = 0.6\,\text{V}$ reference voltage within the IC, in order to increase or decrease the duty cycle of the PWM signal at the SW pin, as appropriate, in order to reach a desired $V_{OUT,\,target}$. [Thus, e.g., if $V_{OUT,\,target} = 5\,\text{V}$, one could select, say, $R_{FBT} = 73.25\,\text{kohm}$ and $R_{FBB} = 10\,\text{kohm}$.]

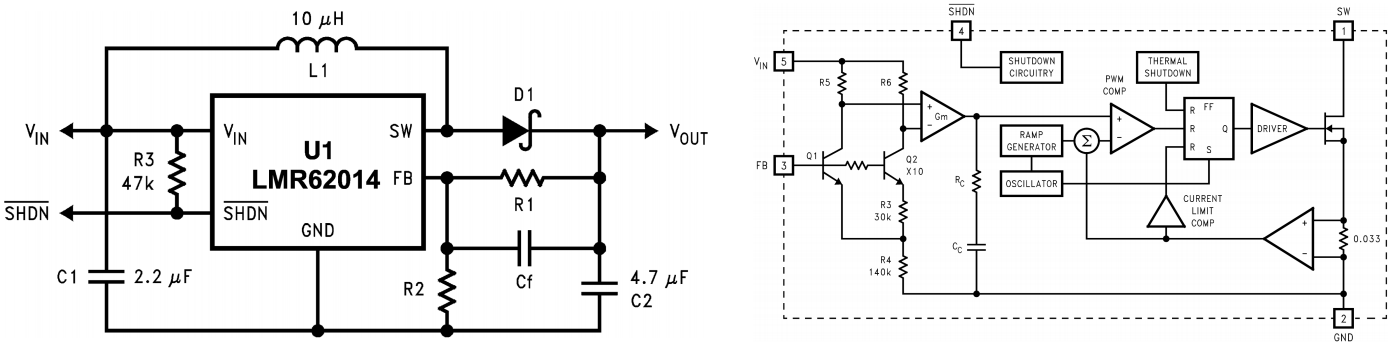


Figure 9.27: The (left) simple external wiring and (right) complex internal circuitry of the TI LMR62014 boost converter IC. In the external wiring diagram, the discrete components marked {L1, C2, D1} correspond to the $\{L, C, d\}$ components in Figure 9.25b, and the discrete resistors marked R1 and R2 form a voltage divider (see Example 9.1) that generate $V_{FB} = V_{OUT} \cdot R2/(R1 + R2)$ at the FB pin on the IC, which is compared to an (internally-generated, with a zener diode) $V_{FB,\,ref} = 1.23\,\text{V}$ reference voltage within the IC, in order to increase or decrease the duty cycle of the PWM signal at the SW pin, as appropriate, in order to reach a desired $V_{OUT,\,target}$. [Thus, e.g., if $V_{OUT,\,target} = 12\,\text{V}$, one could select, say, $R1 = 117\,\text{kohm}$ and $R2 = 13.3\,\text{kohm}$.]

Assume the voltage drop of the diode d, under forward bias, is V_d (with $V_d \ll V_s$), so that the moment that current begins to flow, at $t = 0$, V_m jumps up to V_d. Thereafter, the eqn for the voltage across the diode, while it remains under forward bias, is simply $V_\text{m} - V_\text{out} = V_d$. Noting Table 8.1, we thus have

$$V_m(t) - V_\text{out}(t) = \begin{cases} 0 & t < 0 \\ V_d & t \geq 0 \end{cases} \quad \Rightarrow \quad V_m(s) - V_\text{out}(s) = V_d/s. \tag{9.13a}$$

Noting similarly that $V_1(t) = 0$ for $t < 0$ and $V_1(t) = V_\text{in}$ for $t \geq 0$, we have $V_1(s) = V_\text{in}/s$.

In the development that follows (which will account for the periodic pressing and releasing of the button), we will focus specifically on the time evolution of both $V_\text{out}(t)$ and $I_L(t)$. In this startup phase, the initial values of both of these variables is zero. However, in order to better handle "Phase B" of the analysis below, we will consider even from the beginning here the possibility of nonzero initial values of both of these variables, denoting the initial value of V_out as V_out^B, and the initial value of $I_L(t)$ as I_L^B. With this, the Laplace transform of the component and KCL equations in (9.18) may be written

$$V_\text{out}(s) = R\, I_R(s), \;\; I_C(s) = C\,[sV_\text{out}(s) - V_\text{out}^B], \;\; V_\text{in}/s - V_\text{m}(s) = L\,[sI_L(s) - I_L^B], \tag{9.13b}$$

$$I_L(s) = I_C(s) + I_R(s). \tag{9.13c}$$

In (9.13a)-(9.13c), we have 5 eqns in the 5 variables $\{I_L(s), I_C(s), I_R(s), V_\text{m}(s), V_\text{out}(s)\}$. These 5 eqns may be combined to determine stand-alone algebraic eqns for $V_\text{out}(s)$ and $I_L(s)$ [see code in RR.ch09], which gives:

$$V_\text{out}(s) = \frac{b_2\, s^2 + b_1\, s + b_0}{s\,[s^2 + a_1 s + a_0]}, \quad I_L(s) = \frac{c_2\, s^2 + c_1\, s + c_0}{s\,[s^2 + a_1 s + a_0]}, \quad \text{where} \quad a_1 = 1/(C\,R), \quad a_0 = 1/(L\,C) \tag{9.14}$$

$$b_2 = V_\text{out}^B, \quad b_1 = I_L^B/C, \qquad\qquad b_0 = (V_\text{in} - V_d)/(L\,C),$$

$$c_2 = I_L^B, \quad c_1 = (V_\text{in} - V_d - V_\text{out}^B)/L + I_L^B/(C\,R), \quad c_0 = (V_\text{in} - V_d)/(L\,C\,R).$$

Noting the $e^{-\sigma t}\cos(\omega_d t)$ and $e^{-\sigma t}\sin(\omega_d t)$ entries in Table 8.1, setting $\sigma = a_1/2$ and $\omega_d = \sqrt{a_0 - a_1^2/4}$, gives

$$V_\text{out}(s) = \frac{b_2\, s^2 + b_1\, s + b_0}{s\,[s^2 + a_1 s + a_0]} = \frac{b_2\, s^2 + b_1\, s + b_0}{s\,[(s+\sigma)^2 + \omega_d^2]} = B_2\,\frac{1}{s} + B_1\frac{(s+\sigma)}{(s+\sigma)^2 + \omega_d^2} + B_0\frac{\omega_d}{(s+\sigma)^2 + \omega_d^2}. \tag{9.15a}$$

$$I_L(s) = \frac{c_2\, s^2 + c_1\, s + c_0}{s\,[s^2 + a_1 s + a_0]} = \frac{c_2\, s^2 + c_1\, s + c_0}{s\,[(s+\sigma)^2 + \omega_d^2]} = C_2\,\frac{1}{s} + C_1\frac{(s+\sigma)}{(s+\sigma)^2 + \omega_d^2} + C_0\frac{\omega_d}{(s+\sigma)^2 + \omega_d^2}. \tag{9.15b}$$

This may be solved for $\{B_2, B_1, B_0\}$ and $\{C_2, C_1, C_0\}$ by forming a common denominator and setting like powers of s in the numerator as equal, which gives

$$B_2 = \frac{b_0}{\sigma^2 + \omega_d^2}, \;\; B_1 = b_2 - B_2, \;\; B_0 = \frac{b_1 - b_2\sigma}{\omega_d} - B_2\frac{\sigma}{\omega_d}, \;\; C_2 = \frac{c_0}{\sigma^2 + \omega_d^2}, \;\; C_1 = c_2 - C_2, \;\; C_0 = \frac{c_1 - c_2\sigma}{\omega_d} - C_2\frac{\sigma}{\omega_d},$$

and thus, for $t \geq 0$,

$$V_\text{out}(t) = B_2 + B_1\, e^{-\sigma t}\cos(\omega_d\, t) + B_0\, e^{-\sigma t}\sin(\omega_d\, t), \quad I_L(t) = C_2 + C_1\, e^{-\sigma t}\cos(\omega_d\, t) + C_0\, e^{-\sigma t}\sin(\omega_d\, t), \tag{9.16}$$

where the constants $\{\omega_d, \sigma, B_2, B_1, B_0, C_2, C_1, C_0\}$ depend on $\{L, C, R, V_\text{in}, V_d, V_\text{out}^B, I_L^B\}$ via the equations above.

It follows from (9.16) that $V_\text{out}(t = 0) = B_2 + B_1 = b_2 = V_\text{out}^B$ and $I_L(t = 0) = C_2 + C_1 = c_2 = I_L^B$, as specified. Thus, simplifying (9.16) by taking $V_\text{out}^B = I_L^B = 0$ gives $V_\text{out}(t = 0) = I_L(t = 0) = 0$, as expected. Both $V_\text{out}(t)$ and $I_L(t)$ are oscillating decaying sinusoids with frequency $\omega_d = \sqrt{1/(L\,C) - 1/(4\,C^2\,R^2)}$ and damping $\sigma = 1/(2\,C\,R)$, eventually approaching $V_\text{out}(t) \to B_2 = V_\text{in} - V_d$ and $I_L(t) \to C_2 = (V_\text{in} - V_d)/R$.

If the diode is removed (thus setting $V_d = 0$ in our equations), the above expressions would apply for all $t \geq 0$. With the diode present, this solution is only valid until $I_L(t)$ falls to zero, after which time the diode shuts off. However, taking $V_{\text{out}}^B = I_L^B = 0$ and finite positive values for $\{L, C, R\}$, $I_L(t) > 0$ for all $t \geq 0$, so the above solution is valid for all $t \geq 0$ in the setting described as long as the button b is not pressed.

Phase A. At time $t = t_A$, the button b is pressed **closed** (leaving the switch at left in the **on** position); the moment this button is pressed, V_m jumps down to 0 (that is, to GND). Thus, by pressing this button, the diode is put under reverse bias, effectively isolating the portion of the circuit to the left of the diode from that to its right. These two portions of the circuit, now electrically isolated, are thus analyzed separately below.

The current $I_L(t)$ for $t_A \leq t \leq t_B$ is governed by a single ODE, with an initial value at $t = t_A$ of I_L^A:

$$dI_L(t)/dt = V_{\text{in}}/L \quad \text{with} \quad I_L(t_A) = I_L^A \quad \Rightarrow \quad \boxed{I_L(t) = V_{\text{in}}(t - t_A)/L + I_L^A \quad \text{for} \;\; t_A \leq t \leq t_B.}$$

The voltage $V_{\text{out}}(t)$ for $t_A \leq t \leq t_B$ is also governed by a single ODE, with an initial value at $t = t_A$ of V_{out}^A:

$$[1/(RC) + d/dt]\, V_{\text{out}}(t) = 0 \quad \text{with} \quad V_{\text{out}}(t_A) = V_{\text{out}}^A \quad \Rightarrow \quad \boxed{V_{\text{out}}(t) = V_{\text{out}}^A\, e^{-(t-t_A)/(RC)} \quad \text{for} \;\; t_A \leq t \leq t_B.}$$

Phase B. At time $t = t_B$, the button b is released to **open** (leaving the switch at left in the **on** position); the moment the button is released, the current from the inductor is again rerouted through the diode, and V_m jumps back up to again satisfy the equation $V_{\text{m}} - V_{\text{out}} = V_d$. The voltage and current of this circuit are thus precisely as given in (9.16) in the vicinity of $t' = t - t_B = 0$, with nonzero initial values, at $t' = 0$ (that is, at $t = t_B$), of I_L^B and V_{out}^B, determined by evaluating the boxed expressions above for $I_L(t)$ and $V_{\text{out}}(t)$ at $t = t_B$:

$$\boxed{\begin{aligned} V_{\text{out}}(t) &= B_2 + B_1\, e^{-\sigma(t-t_B)} \cos[\omega_d\,(t - t_B)] + B_0\, e^{-\sigma(t-t_B)} \sin[\omega_d\,(t - t_B)] \quad \text{for} \;\; t_B \leq t \leq t_C \\ I_L(t) &= C_2 + C_1\, e^{-\sigma(t-t_B)} \cos[\omega_d\,(t - t_B)] + C_0\, e^{-\sigma(t-t_B)} \sin[\omega_d\,(t - t_B)] \quad \text{for} \;\; t_B \leq t \leq t_C. \end{aligned}}$$

Periodic oscillation. At time $t = t_C$, the button b is again pressed **closed** (again, leaving the switch at left in the **on** position), thus re-entering Phase A, with nonzero initial values for I_L^A and V_{out}^A, determined by evaluating the expressions above for $V_{\text{out}}(t)$ and $I_L(t)$ at $t = t_C$. Replacing the button with a MOSFET connected to GND, and excited at its gate with a PWM signal (generated by a microcontroller, like that on the Berets in §5), this process repeats periodically, at a constant (very high) frequency f and duty cycle D where $0 < D < 1$, for some t_A and corresponding $t_B = t_A + D/f$ and $t_C = t_A + 1/f$. After repeating many times, a periodic behavior of the circuit settles in over each period $t_A \leq t \leq t_C$ (that is, at the same frequency f as this PWM excitation).

The periodic condition that this system converges to may be found by setting the values of $I_L(t)$ and $V_{\text{out}}(t)$ at $t = t_A$ equal to the values of $I_L(t)$ and $V_{\text{out}}(t)$ at $t = t_C$ in the above analysis. Evaluating the first two boxed equations at time $t = t_B$, setting $V_{\text{out}}(t_B) = V_{\text{out}}^B$ and $I_L(t_B) = I_L^B$, and the second two boxed equations at time $t = t_C$, setting $V_{\text{out}}(t_C) = V_{\text{out}}^A$ and $I_L(t_C) = I_L^A$, gives four conditions for this periodic behavior, which may then be solved to find the $\{V_{\text{out}}^A, V_{\text{out}}^B, I_L^A, I_L^B\}$ which simultaneously solve these four equations:

$$V_{\text{out}}^B = V_{\text{out}}^A\, e^{-(t_B - t_A)/(RC)}, \tag{9.17a}$$

$$I_L^B = V_{\text{in}}(t_B - t_A)/L + I_L^A, \tag{9.17b}$$

$$V_{\text{out}}^A = B_2 + B_1\, e^{-\sigma(t_C - t_B)} \cos[\omega_d\,(t_C - t_B)] + B_0\, e^{-\sigma(t_C - t_B)} \sin[\omega_d\,(t_C - t_B)], \tag{9.17c}$$

$$I_L^A = C_2 + C_1\, e^{-\sigma(t_C - t_B)} \cos[\omega_d\,(t_C - t_B)] + C_0\, e^{-\sigma(t_C - t_B)} \sin[\omega_d\,(t_C - t_B)], \tag{9.17d}$$

where

$$\omega_d = \sqrt{1/(LC) - 1/(4\,C^2\,R^2)}, \quad \sigma = 1/(2\,C\,R), \quad t_B - t_A = D/f, \quad t_C - t_B = (1 - D)/f = \bar{D}/f,$$

$$B_2 = V_{\text{in}} - V_d, \qquad B_1 = V_{\text{out}}^B - (V_{\text{in}} - V_d), \qquad B_0 = [I_L^B/C - V_{\text{out}}^B \sigma]/\omega_d - B_2\sigma/\omega_d,$$

$$C_2 = (V_{\text{in}} - V_d)/R, \quad C_1 = I_L^B - (V_{\text{in}} - V_d)/R, \quad C_0 = [(V_{\text{in}} - V_d - V_{\text{out}}^B)/L + I_L^B(\sigma - 1/(C\,R))]/\omega_d - C_2.$$

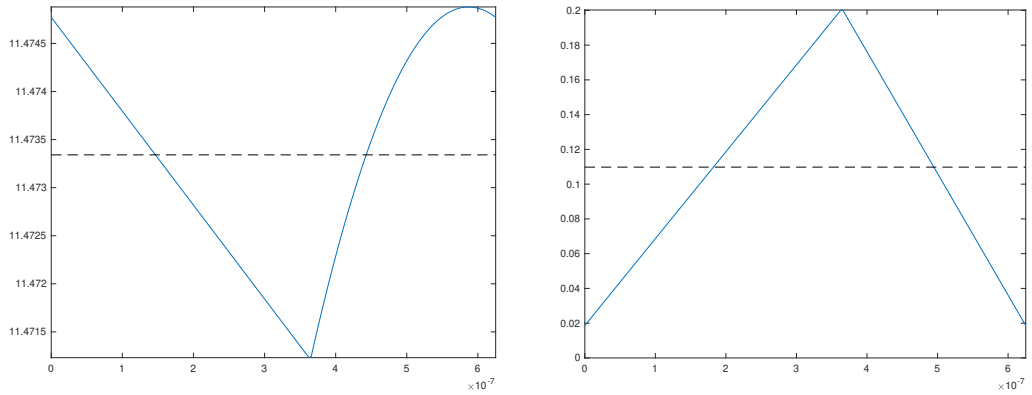


Figure 9.28: Representative curves for (left) $V_{\text{out}}(t)$ and (right) $I_L(t)$ over $t_A \le t \le t_C$ for a boost converter.

The equations above are easily typed directly into the computer and solved (see code in RR.ch09), to give $\{V_{\text{out}}^A, V_{\text{out}}^B, I_L^A, I_L^B\}$ as complicated functions of $\{V_{\text{in}}, V_d, L, C, R, f, D\}$.

Given the values of $\{I_L^A, V_{\text{out}}^A, I_L^B, V_{\text{out}}^B, I_L^C, V_{\text{out}}^C\}$ characterizing the periodic oscillations (with $I_L^A = I_L^C$ and $V_{\text{out}}^A = V_{\text{out}}^C$) as determined above, taking $V_{\text{in}} = 5\,\text{V}$, $V_d = 0.5\,\text{V}$, $L = 10\,\mu\text{H}$, $C = 4.7\,\mu\text{F}$, $R = 250$ ohms, $f = 1.6\,\text{MHz}$, and $D = 7/12$, and noting the four boxed equations above for $V_{\text{out}}(t)$ and $I_L(t)$ over both Phase A ($t_A \le t < t_B$) and Phase B ($t_B \le t < t_C$), the corresponding $V_{\text{out}}(t)$ and $I_R(t)$ are plotted over the entire interval $t_A \le t \le t_C$ in Figure 9.28. Note that V_{out} is a decaying exponential on (t_A, t_B), and a decaying sinusoid on (t_B, t_C); I_L is linear on (t_A, t_B), and a decaying sinusoid on (t_B, t_C). When excited at a high frequency f, most of these curves appear to be nearly linear over each phase. Overall, there are relatively small fluctuations in $V_{\text{out}}(t)$, and relatively large fluctuations in $I_L(t)$. Note also that $I_L^{\text{mean}} \approx V_{\text{out}}^{\text{mean}}/[R\,(1-D)]$. By trial and error, it is found that a slightly adjusted value of $D \approx 0.60086$ gives $V_{\text{out}}^{\text{mean}} \approx 12V$.

Implementation. Rather than running at a fixed duty cycle, feedback can again be implemented to identify the duty cycle D required to more precisely achieve a desired value of $V_{\text{out}}^{\text{mean}}$. This is especially important when the value of V_{in} is not accurately known (e.g., if it comes from the output of a LiPo, which ranges from 3.0 to 4.2 V per cell). This may be achieved, e.g., by implementing a TI LMR62014, which is compact (less than 9 mm^2), easy to hook up (SMT, 5 pins), and inexpensive ($0.27). The (simple) external wiring and (complicated) internal circuitry diagrams for the LMR62014 (which generates the required PWM signal with a few op amps) are illustrated in Figure 9.27; note the button b is replaced by a PWM-actuated MOSFET illustrated near the SW pin of the internal circuitry diagram (Figure 9.27b). $V_{\text{out}}(t)$ may then be hooked to a load of a few hundred ohms (or more), and this circuit will drive the tens of mA (or less) necessary to drive the load.

Note that the $I_L(t)$ plot given in Figure 9.28 also appears on page 12 of the LMR62014 datasheet.

Example 9.25 Buck-boost converters.

A buck-boost converter can step a DC input voltage V_{in} either up *or* down, as necessary, to generate a desired DC output voltage V_{out}. Its essential components are illustrated in Figure 9.25c. Note that what we refer to in this discussion as the "button" is replaced in the implementation by a MOSFET, acting as a high-speed "switch" driven by a PWM signal at a specified (relatively high) frequency ω and duty cycle D, with $0 \le D < 1$.

The principle of operation of a buck-boost converter is similar to that of the boost converter (Example 9.24).

During Phase A, the "button" is closed (i.e., the MOSFET is "on"), and thus the current travels from the top of the battery (which generates V_{in}), through the closed button, down through the inductor to ground (i.e., back to the battery). During this phase, the current through the inductor, I_{in} increases linearly.

During Phase B, the "button" is open (i.e., the MOSFET is "off"). Note that the current through the inductor is continuous. This current has to go somewhere during Phase B; since the battery is disconnected from the inductor (by the button) during this phase, this current flows back up through the capacitor the load resistor,

and through the diode back to the top of the inductor. Defining the voltage at the bottom of the battery, inductor, capacitor, and load resistor in Figure 9.25c as ground, it is thus seen that the voltage at V_{out} must be *negative*.

The voltage drop across the capacitor (and, thus, V_{out}) is continuous. Since the frequency of the PWM excitation of the button is relatively high, the magnitude of the voltage fluctuation at V_{out} is relatively small. Thus:

- during Phase A, the diode is "off" (as $V_m = V_{\text{in}}$, and $V_{\text{out}} < 0$), and
- during Phase B, the diode is "on" (with $V_m = V_{\text{out}} - V_d$, where V_d is the cut-in voltage of the diode).

Note that V_m jumps when switching from Phase A and Phase B, and again when switching back to Phase A.

Again, the full analysis of the buck-boost converter is quite similar to Example 9.24, which we follow closely below. We focus specifically on the periodic condition that the system quickly settles into.

Phase A. At time $t = t_A$, the button is pressed **closed**, and the diode is off, effectively isolating the portion of the circuit to the left of the diode from that to its right. The current $I_L(t)$ for $t_A \leq t \leq t_B$ is governed by a single ODE, with an initial value at $t = t_A$ of I_L^A:

$$dI_L(t)/dt = V_{\text{in}}/L \quad \text{with} \quad I_L(t_A) = I_L^A \quad \Rightarrow \quad \boxed{I_L(t) = V_{\text{in}}(t - t_A)/L + I_L^A \quad \text{for} \;\; t_A \leq t \leq t_B.}$$

The voltage $V_{\text{out}}(t)$ for $t_A \leq t \leq t_B$ is also governed by a single ODE, with an initial value at $t = t_A$ of $V_{\text{out}}^A < 0$:

$$[1/(RC) + d/dt]\, V_{\text{out}}(t) = 0 \quad \text{with} \quad V_{\text{out}}(t_A) = V_{\text{out}}^A \quad \Rightarrow \quad \boxed{V_{\text{out}}(t) = V_{\text{out}}^A\, e^{-(t-t_A)/(RC)} \quad \text{for} \;\; t_A \leq t \leq t_B.}$$

Phase B. At time $t = t_B$, the button is **open**.

The eqns for the R, C, and L components, and the KCL eqn at the V_{out} node, may be written

$$V_{\text{out}}(t) = R\, I_R(t), \quad I_C(t) = C\, d[V_{\text{out}}(t)]/dt, \quad V_{\text{m}}(t) = L\, d[I_L(t)]dt, \quad I_L(t) + I_C(t) + I_R(t) = 0. \tag{9.18}$$

9.2.5 BDC and BLDC motor control

move some motor control stuff to here...

9.2.6 Implementing digital logic using CMOS: Inverters, NOR, and NAND

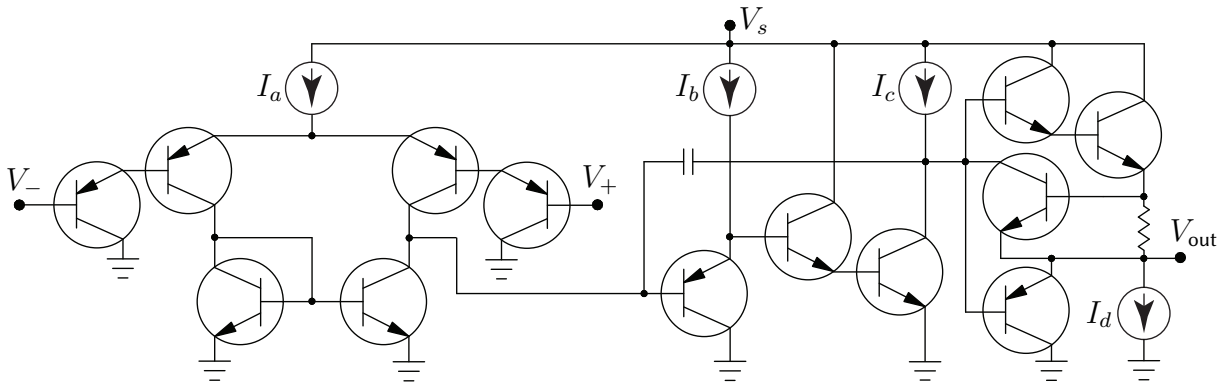


Figure 9.29: Internal construction of an LM324 op amp, with $I_a = I_b = 6\,\mu$A, $I_c = 100\,\mu$A, $I_d = 50\,\mu$A. The first stage of the op amp is the differential amplifier considered in Example 9.21 (see Figure 9.22c), with **darlington transistors** (that is, a cascade of two transistors, interconnected as shown) used on each of the inputs to increase the gain, and the output resistor replaced by the op amp's second stage. The rest of the circuit amplifies the output from the first stage, effectively implements an RC first-order low-pass filter (note the resistor and the capacitor) to suppress very-high-frequency noise, and provides high current driving capability with low output impedance as well as short circuit protection. The LM324 quad op amp implements four such circuits together in a single, robust, and convenient 14-pin **dual in-line package** (**DIP**).

9.3 Operational amplifiers

An **operational amplifier** (a.k.a. **op amp**) is an **active** (powered) integrated circuit with two inputs, $V_+(t)$ and $V_-(t)$, and one output, $V_{\text{out}}(t)$, that functions as a differential amplifier with output

$$V_{\text{out}}(t) = \begin{cases} V_{s+} & \text{if } V_{s+} < V_o(t) \\ V_o(t) & \text{if } V_{s-} < V_o(t) < V_{s+} \\ V_{s-} & \text{if } V_o(t) < V_{s-} \end{cases} \quad \text{with} \quad V_o(t) \approx A\,[V_+(t) - V_-(t)], \tag{9.19a}$$

where the gain A is *very* large (indeed, it is often approximated as $A \to \infty$), and two additional properties:

a) **very high input impedance** (that is, the input terminals of the op amp draw negligible current), and
b) **very low output impedance** (that is, the output voltage of the op amp is set by the input voltages as specified above, essentially independent of the attached load).

The internal construction of an op amp is a fairly involved arrangement of transistors and other circuit elements, as typified by[26] the LM324 op amp illustrated[27] in Figure 9.29. A more accurate *dynamic* model of $V_o(t)$ in the (typical) LM324 op amp, which takes into account the fact that the magnitude of its frequency response rolls off at a couple hundred kilohertz [cf. (9.19a)], may be written in transfer-function form as

$$V_o(s) = G(s)\,[V_+(s) - V_-(s)] \quad \text{with} \quad G(s) = A\,\frac{a}{s+a}, \tag{9.19b}$$

where $A \approx 10^5$ and $a \approx 10^6$. Note that the low-pass-filter nature of an op amp is usually neglected [see (9.19a)]; that is, the corner frequency a is so large that the transfer function $G(s)$ of the op amp is usually approximated as a pure gain (and, further, the gain A of an op amp is so large that it is often considered to be essentially infinite when modeling the behavior of an op amp circuit). However, the more precise model of an op amp

[26] Note that there are many such op amp designs that lead to the same essential properties.
[27] Note that, in Figure 9.29, V_{s+} is denoted V_s, and V_{s-} is denoted by ground. Both conventions are common.

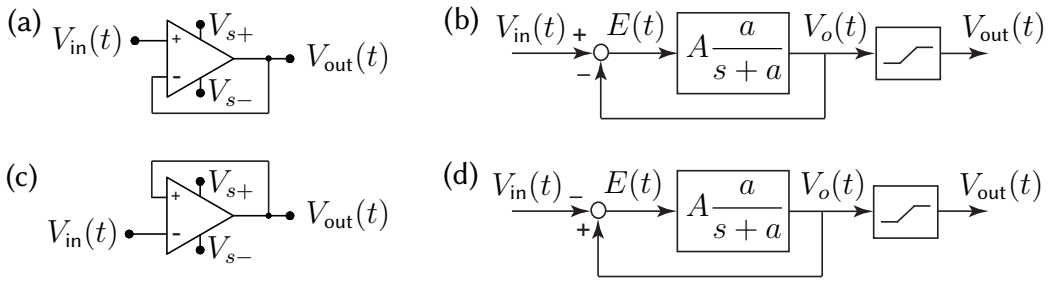


Figure 9.30: Two simple op amp circuits: (a) an op amp wired with negative feedback and (b) its corresponding block diagram, and (c) an op amp wired with positive feedback and (d) its corresponding block diagram. As op amps are **active** devices, their connections to V_{s+} & V_{s-} are often shown explicitly in the op amp symbol, as in (a) and (c); these connections are suppressed for notational simplicity in most of the remainder of this text.

given in (9.19b) is the best starting point to understand op amp behavior, as it explains why an op amp with feedback is either stable or unstable (note that both modes have their uses), depending on which input terminal the feedback is connected to, as shown below.

Consider first the simple op amp circuit in Figure 9.30a, with **negative feedback**, and its corresponding block diagram in Figure 9.30b, with an input-output transfer function of

$$\left.\begin{array}{l} V_o(s) = G(s)\,E(s) \\ E(s) = V_{\text{in}}(s) - V_o(s) \end{array}\right\} \quad \Rightarrow \quad H(s) = \frac{V_o(s)}{V_{\text{in}}(s)} = \frac{G(s)}{1+G(s)} = \frac{aA}{s+a+aA} \approx \frac{aA}{s+aA}.$$

The gain of this first-order low-pass filter is nearly unity over a very wide range of frequencies; note the fast stable pole at $s \approx -aA$. With large A, this circuit behaves as a **voltage follower** or **buffer**, with $V_{\text{out}}(t) \approx V_{\text{in}}(t)$. This active circuit is useful because, due to its high input impedance and low output impedance, it **isolates** the circuits hooked to its input and output terminals; that is, it draws negligible current from the circuit connected to its input terminal, and maintains $V_{\text{out}}(t) \approx V_{\text{in}}(t)$ while providing as much current as required (within limits) by the circuit connected to its output terminal, thus allowing filters to be constructed and analyzed as independent stages then cascaded together, effectively relaxing the restrictive assumptions of Example 9.2.

Now consider the op amp circuit in Figure 9.30c, with **positive feedback**, and its corresponding block diagram in Figure 9.30d, with an input-output transfer function of

$$\left.\begin{array}{l} V_o(s) = G(s)\,E(s) \\ E(s) = V_o(s) - V_{\text{in}}(s) \end{array}\right\} \quad \Rightarrow \quad H(s) = \frac{V_o(s)}{V_{\text{in}}(s)} = \frac{-G(s)}{1-G(s)} = \frac{-aA}{s+a-aA} \approx \frac{-aA}{s-aA}.$$

Due to the (fast) unstable pole at $s = aA$, *the equilibrium* $V_o(t) \approx V_{in}(t)$ *is unstable*, and is thus, in practice, never realized. Instead, $V_{\text{out}}(t)$ is driven to one of the limiting values of the op amp, V_{s+} or V_{s-}, and stays there; which limit it goes to depends on the initial values of $V_o(t)$ and $V_{\text{in}}(t)$ when the op amp is turned on.

9.3.1 Design and analysis of a few useful op amp circuits

We now show via several examples how the arrangement of transistors in an op amp is convenient in a variety of practical situations. Note that *almost all useful op amps circuits implement feedback*, with Example 9.26 being a notable exception. Further, *almost all useful op amps circuits implementing feedback use the (stable)* **negative feedback** *configuration* discussed above, with Examples 9.29, **??**, and 9.32 being notable exceptions.

Example 9.26 Voltage comparator. When implemented without feedback, a bare op amp (9.19a) in the large gain limit $A \to \infty$ functions simply as a **voltage comparator**:

$$V_{\text{out}}(t) = \begin{cases} V_{s+} & \text{if } V_+(t) > V_-(t), \\ V_{s-} & \text{if } V_+(t) < V_-(t). \end{cases}$$

Example 9.27 Inverting and noninverting amplifiers. Implementing (stabilizing) feedback to the inverting input of the op amp, an **inverting amplifier** may be implemented as shown in Figure 9.31a, in which

$$V_{\text{in}}(t) - V_-(t) = I_{\text{in}}(t)\, R/M, \quad V_-(t) - V_{\text{out}}(t) = I_R(t)\, R, \quad I_{\text{in}}(t) = I_R(t);$$

applying (9.19b) thus leads to

$$V_{\text{out}}(s) = \frac{a\,A}{s+a}\,[0 - V_-(s)] \quad \Rightarrow \quad \frac{V_{\text{out}}(s)}{V_{\text{in}}(s)} = \frac{-M\,a\,A}{(M+1)s + [a\,A + a(M+1)]} \quad \underset{A\to\infty}{\Longrightarrow} \quad V_{\text{out}}(t) \approx -M\,V_{\text{in}}(t).$$

Similarly, a **noninverting amplifier** may be implemented as shown in Figure 9.31b, in which

$$V_-(t) = I_O(t)\, R/f, \quad V_-(t) - V_{\text{out}}(t) = I_R(t)\, R, \quad I_O(t) = -I_R(t);$$

applying (9.19b) leads to

$$V_{\text{out}}(s) = \frac{a\,A}{s+a}\,[V_{\text{in}}(s) - V_-(s)] \quad \Rightarrow \quad \frac{V_{\text{out}}(s)}{V_{\text{in}}(s)} = \frac{a\,A}{s + a + a\,A/(1+f)} \quad \underset{A\to\infty}{\Longrightarrow} \quad V_{\text{out}}(t) \approx (1+f)\,V_{\text{in}}(t).$$

As illustrated by both of these examples, the (stabilizing) feedback to the inverting input of the op amp leads, in the $A \to \infty$ limit, to the condition that $V_+ = V_-$; note in both cases the very fast stable poles. It often simplifies the analysis of a stable op amp circuit to simply apply the condition $V_+ = V_-$ at the outset; if you have doubts whether or not the circuit considered is stable, implement (9.19b) instead, as done above.

Example 9.28 A general op amp circuit for adding and subtracting. Appropriate combination of the inverting and noninverting amplifiers of Example 9.27 leads to an op amp circuit such that

$$V_{\text{out}}(t) = \sum_{j=1}^{n} m_j\, v_j - \sum_{j=1}^{N} M_j\, V_j, \tag{9.20}$$

that is, to an op amp circuit that can perform an arbitrary linear combination of $n + N$ inputs, with n positive coefficients m_j and N negative coefficients $(-M_j)$. Defining $f = \sum m_j - \sum M_j - 1$, we will consider three cases: $f < 0$, $f = 0$, and $f > 0$. In the sample circuit we will consider, we take $n = N = 3$; the modifications required to handle a different numbers of inputs are trivial. Most op amp circuits used for adding and subtracting, as found online, are special cases of the general circuit presented here.

The circuit required in the $f < 0$ case is illustrated in Figure 9.31c. For notational clarity, in this example only, we take the voltages, currents, and resistances in the upper half of the circuit as uppercase, and the voltages, currents, and resistances in the lower half of the circuit as lowercase. Ohm's law and KCL then give

$$\begin{array}{lllll} V_1 - V_- = I_1\, R/M_1, & V_2 - V_- = I_2\, R/M_2, & V_3 - V_- = I_3\, R/M_3, & V_- - V_{\text{out}} = I_R\, R, & I_1 + I_2 + I_3 = I_R, \\ v_a - v_+ = i_a\, r/m_a, & v_b - v_+ = i_b\, r/m_b, & v_c - v_+ = i_c\, r/m_c, & v_+ = i_o\, r/|f|, & i_a + i_b + i_c = i_o. \end{array}$$

Since negative (stable) feedback is used, assuming $A \to \infty$, we take $V_- = v_+$; noting that $f = \sum m_j - \sum M_j - 1$ and solving then leads immediately to (9.20).

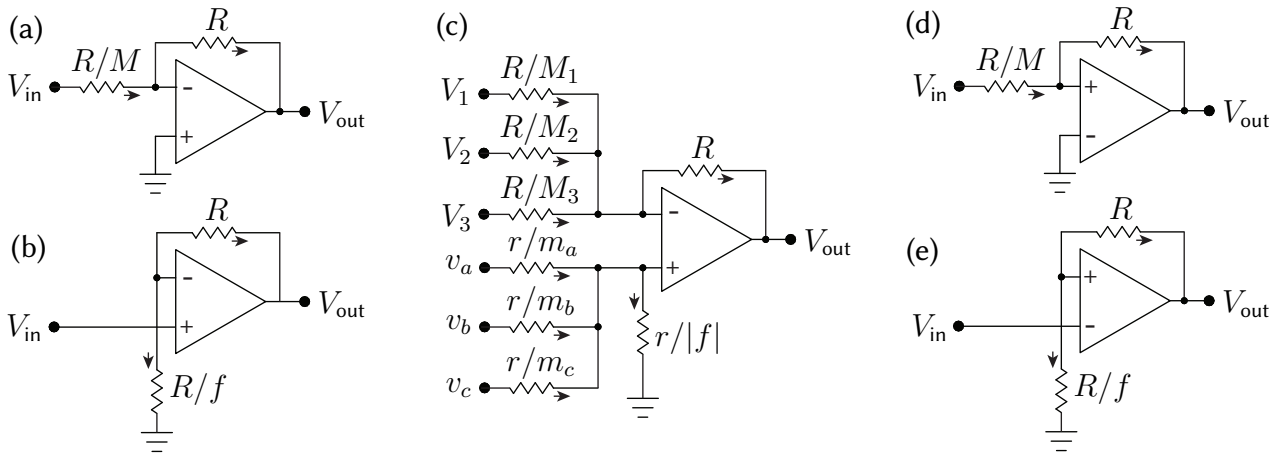


Figure 9.31: Some useful op amp circuits. (a) Inverting amplifier. (b) Noninverting amplifier. (c) A general adder/subtractor [note: the ground connection shown is for $f < 0$; if $f = 0$, this connection to ground is removed; if $f > 0$, the connection to ground is attached to V_- instead of V_+, through a resistance R/f]. (d) Noninverting Schmitt trigger. (e) Inverting Schmitt trigger. Note that (d) and (e) are hysteretic.

As $f \to 0$, the resistance of the connection between the noninverting input of the op amp and ground in Figure 9.31c goes to infinity. In the $f = 0$ case, this connection may thus be eliminated entirely; removing the equation $v_+ = i_o\, r/|f|$ from the above set of equations, taking $i_o = 0$, and solving leads again to (9.20).

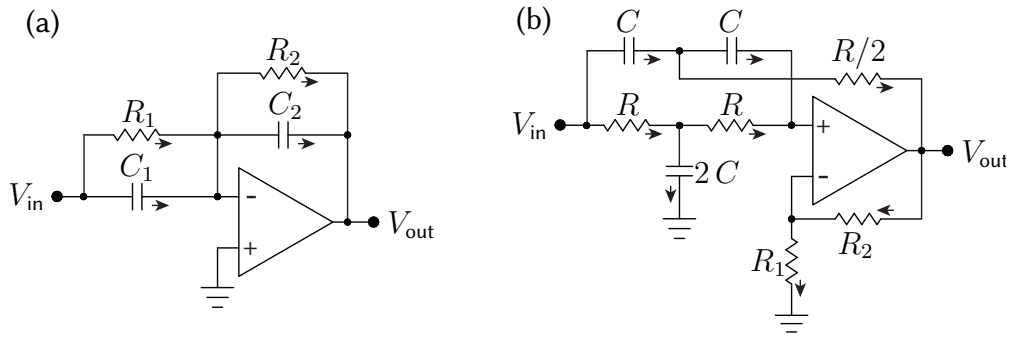
Finally, in the $f > 0$ case, we replace the connection between the noninverting input of the op amp and ground with a connection between the inverting input of the op amp and ground, with resistance R/f. In this case, the equation $v_+ = i_o\, r/|f|$ in the above set of equations is replaced by $V_- = I_o\, R/f$, and the two KCL relations are now $I_1 + I_2 + I_3 = I_o + I_R$ and $i_a + i_b + i_c = 0$; solving again leads to (9.20).

In all three cases, $f < 0$, $f = 0$, and $f > 0$, the resulting relation between the voltages, (9.20), is in fact *independent* of both R and r, which are typically selected so that all resistors used in the circuit are between 1 kΩ and 100 kΩ. Note that, in the case that $n = 0$ (see, e.g., Figure 9.31a), we may take $r = 0$, wiring the noninverting input of the op amp directly to ground. In the case that $n = 1$ and $f \geq 0$ (see, e.g., Figure 9.31b), we may also take $r = 0$, wiring the noninverting input of the op amp directly to v_a.

Example 9.29 Schmitt triggers. We now consider two **hysteretic** circuits that are simply the inverting and noninverting amplifiers of Example 9.27 with the inputs to the op amp swapped from the stable (negative-feedback) configuration to the unstable (positive-feedback) configuration. Assuming $V_{s+} = V_s$ and $V_{s-} = -V_s$,

- in the unstable circuit illustrated in Figure 9.31d, called a **noninverting Schmitt trigger**,
  - if $V_{\text{out}} = +V_s$, then it will stay there until V_{in} passes below $-V_s/M$ (that is, until V_+ passes below V_-), after which the output will switch to $V_{\text{out}} = -V_s$, whereas
  - if $V_{\text{out}} = -V_s$, then it will stay there until V_{in} passes above V_s/M (that is, until V_+ passes above V_-), after which the output will switch to $V_{\text{out}} = +V_s$;
- in the unstable circuit illustrated in Figure 9.31e, called a **inverting Schmitt trigger**,
  - if $V_{\text{out}} = +V_s$, then it will stay there until V_{in} passes above $V_s/(1+f)$ (that is, until V_- passes above V_+), after which the output will switch to $V_{\text{out}} = -V_s$, whereas
  - if $V_{\text{out}} = -V_s$, then it will stay there until V_{in} passes below $-V_s/(1+f)$ (that is, until V_- passes below V_+), after which the output will switch to $V_{\text{out}} = +V_s$.

A primary application of Schmitt triggers is **switch debouncing**: once a remotely-operated Schmitt trigger, acting as a switch, is flipped one way, it takes a large change in the input to flip the Schmitt trigger the other way, thus preventing "chatter" of the switch due to noise over the communication channel.



s

Figure 9.32: Some *dynamic* op amp circuits: (a) the **inverting first-order filter** $F(s) = -K(s+z)/(s+p)$ of Example 9.30, which may be simplified in of several different useful ways, and (b) the **notch filter** $F_{\text{notch}}(s) = K(s^2+\omega_o^2)/(s^2+\omega_o\, s/Q+\omega_o^2)$ of Example 9.31. .

Example 9.30 A general-purpose inverting first-order filter. The circuit illustrated in Figure 9.32a is a remarkably flexible general-purpose inverting first-order filter design with transfer function

$$F(s) = \frac{V_{\text{out}}(s)}{V_{\text{in}}(s)} = -\frac{C_1}{C_2}\,\frac{s+1/(R_1C_1)}{s+1/(R_2C_2)} = -\frac{R_2}{R_1}\,\frac{1+R_1C_1s}{1+R_2C_2s} = -R_2C_1\,\frac{s+1/(R_1C_1)}{1+R_2C_2s} = -\frac{1}{R_1C_2}\,\frac{1+R_1C_1s}{s+1/(R_2C_2)}.$$

That is, $F(s) = -K_0(s+z)/(s+p)$, where $K_0 = C_1/C_2$, $z = 1/(R_1C_1)$, and $p = 1/(R_2C_2)$; we also define $K_1 = R_2/R_1$, $K_2 = R_2C_1$, and $K_3 = 1/(R_1C_2)$. If the op amp is ideal, the circuit design in Figure 9.32a is actually nine circuits in one, reducing[28] in the appropriate limits to *all* of the inverting first-order filters:

- taking $R_1C_1 > R_2C_2$, it is an inverting **lead filter** with $F(s) = -K_0(s+z)/(s+p)$ where $z < p$;
- taking $R_2C_2 > R_1C_1$, it is an inverting **lag filter** with $F(s) = -K_0(s+z)/(s+p)$ where $p < z$;
- removing C_1, it is an inverting **first-order low-pass filter** with $F(s) = -K_3/(s+p)$;
- removing R_1, it is an inverting **first-order high-pass filter** with $F(s) = -K_0\, s/(s+p)$;
- removing[29] R_2, it is an inverting **PI filter** with $F(s) = -K_0\,(s+z)/s$;
- removing[29] R_2 and C_1, it is an inverting **pure integrator** $F(s) = -K_3/s$;
- removing C_2, it is an inverting **PD filter**[30] with $F(s) = -K_2\,(s+z)$;
- removing C_2 and R_1, it is an inverting **pure differentiator**[30] $F(s) = -K_2\, s$;
- removing C_1 and C_2, it is an inverting **amplifier** $F(s) = -K_1$.

The development of a corresponding general-purpose *noninverting* first-order filter is considered in Exercise 9.6. To build a second-order filter that incorporates both a first-order lag filter at low frequencies (to reduce steady-state error) and a first-order lead filter at high frequencies (to improve damping and reduce overshoot), creating what is called a **lead/lag filter** (see Figure 10.3.2b), one may simply cascade together the lead and lag filters described above as necessary. Note that a **PID filter** is simply a special case of a lead/lag filter with

- the roll-off of the integral action of the lag filter taken all the way down to $\omega \to 0$, and
- the roll-off of the derivative action of the lead filter taken all the way up to $\omega \to \infty$.

To build a PID filter, one could simply cascade together the PI and PD filters described above. However, note that lead/lag filters are strongly preferred over PID filters for the reasons discussed in §10.3.1: that is, the roll-off of the low-frequency gain and the high-frequency gain mentioned above almost never need to be taken all the way

[28] Removing a capacitor corresponds to taking its capacitance $C \to 0$, and removing a resistor corresponds to taking its resistance $R \to \infty$; in both cases, by (9.2), the current goes to zero through the (removed) component regardless of the applied voltage.

[29] Alternatively, R_2 may be replaced by a switch, allowing the integrator to be reset whenever desired.

[30] PD filters and pure differentiators must never be used in practice, because they amplify high-frequency noise without bound, which is a significant problem. *Lead filters* and *first-order high-pass filters* (a.k.a. **dirty differentiators**) should be used instead.

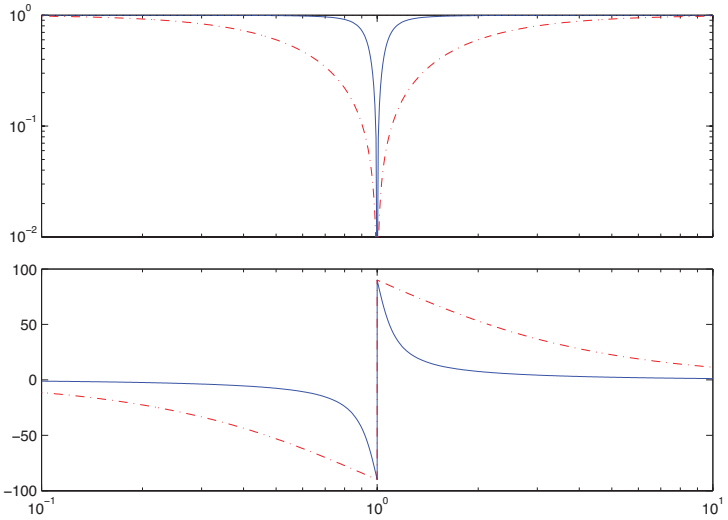


Figure 9.33: Bode plot of the **notch filter** of Example 9.31, rescaled such that $K = 1$, taking $\omega_o = 1$, (dot-dashed) $Q = 0.5$, and (solid) $Q = 5$. Note the "notch" shape of the magnitude part of the Bode plot.

to zero and infinity respectively, and doing such generally causes significant problems (specifically, *integrator windup* and the *amplification of high-frequency noise*) in the closed-loop setting.

Example 9.31 Notch filter. The circuit illustrated in Figure 9.32a, called an **active twin-T filter**, is a convenient op amp circuit implementation of the **notch** transfer function

$$F_{\text{notch}}(s) = \frac{V_{\text{out}}(s)}{V_{\text{in}}(s)} = K \frac{s^2 + \omega_o^2}{s^2 + (\omega_o/Q)s + \omega_o^2} \tag{9.21}$$

where[31] $K = 1 + R_2/R_1$, $Q = 1/[2(2-K)]$, and $\omega_o = 1/(RC)$. Note that the zeros of the notch are pure imaginary, and the so-called "**quality**" of the notch is given by $Q = 1/(2\zeta)$, where ζ is the damping of its poles; thus, $Q = 0.5$ (that is, $K = 1$, or $R_2 = 0$ and $R_1 \to \infty$) corresponds to two identical real poles, and $Q > 0.5$ corresponds to a pair of complex-conjugate poles with damping which decreases as Q is increased.

The Bode plot of $F_{\text{notch}}(s)$ for $\omega_o = 1$ and two different values of Q is illustrated in Figure 9.33; note that the gain of the notch filter is zero at ω_o, and that the width of the range of frequencies significantly affected by the notch decreases with increasing Q. When using a notch to eliminate, e.g., a 50 or 60 Hz "buzz" (that is, noise with a very narrow power spectrum) in a signal, a high Q value is used to minimize the impact of the notch on the signal of interest outside of the range of frequencies corrupted by the buzz. However, when using a notch in a feedback control setting to "knock out" the oscillatory dynamics of a plant (see §10.3.2), one certainly does *not* want to introduce lightly damped poles with the notch, and values of Q in the range $0.5 \leq Q \leq 0.707$ are preferred[32]. Note finally that an active twin-T implementation of a notch filter may be cascaded with a doubled lead filter to move the poles resulting from the notch even further into the LHP.

[31] Note that Q and K can not be set independently in this particular circuit; this usually does not create much of issue, however, because a notch filter is usually cascaded with other op amp circuits that can be used to set the gain to the desired value.

[32] That is, when implementing a notch filter to stabilize, e.g., a Ford automobile, achieving high quality is *not* necessarily job one.

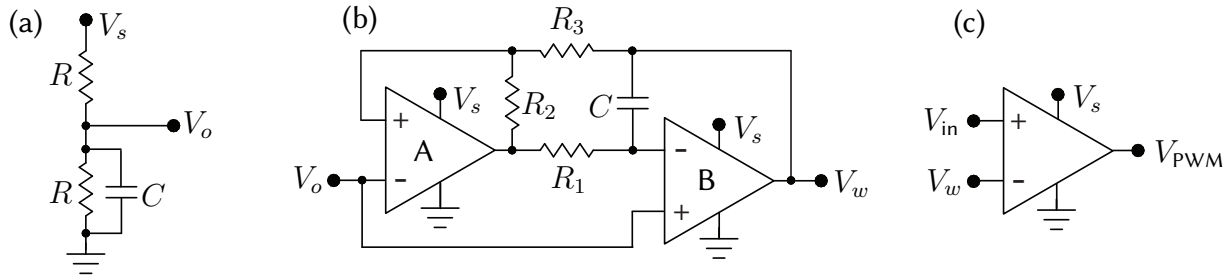


Figure 9.34: A PWM generation circuit with duty cycle V_{in}/V_s and frequency $1/(4\,R_1\,C)$ Hz, formed by the cascade of (a) a passive **stabilized voltage divider**, (b) a **triangle-wave generator**, and (c) a **comparator**.

Example 9.32 Efficient control of high-power loads via pulse width modulation. Given a power supply (e.g., a battery) and a resistive load (e.g., a light bulb), we now consider how to run the load at partial power. An inefficient solution to this problem is simply to put a **variable resistor** (a.k.a. **rheostat**) in series with the load, thus reducing both the current through the load and the voltage across the load, thereby reducing the power consumed by the load. Unfortunately, the variable resistor used in such a setting itself consumes a lot of power that is rejected as waste heat, thereby wiping out any potential senergy savings that might otherwise be realized.

A more efficient solution is to *cycle the voltage applied to the load on and off very quickly*; the percentage of the time the switch is on, called the **duty cycle**, then regulates the (time-averaged) percentage of full power at which the load will operate. This solution, referred to as **pulse width modulation** (**PWM**; see §4.2.3), is facilitated by the fact that transistors are quite efficient when operated as fast switches (see Guideline 9.1).

A good way to produce a PWM signal, which operates at a constant frequency $\omega = 1/(4\,R_3\,C)$ Hz, is given by cascading together the three stages shown in Figure 9.34. The first stage (Figure 9.34a) is a passive voltage divider with $V_o = V_s/2$, with a capacitor added to stabilize the output voltage in case the (small) load attached to its output fluctuates; values of $R \sim 10$ kΩ and $C \sim 100$ nF are typical. The second stage (Figure 9.34b) generates a triangle wave V_w between 0 and V_s and operating at a frequency of $\omega = 1/(4\,R_1\,C)$ Hz, as discussed below. The third stage (Figure 9.34c) then compares the triangle wave V_w with V_{in}, outputting V_s whenever V_{in} is greater than V_w, and outputting 0 whenever V_{in} is less than V_w, thus resulting in a duty cycle of V_{in}/V_s.

To analyze the operation of the triangle-wave generator of Figure 9.34b, denote the inputs and outputs of op amp A (configured in an unstable configuration with positive feedback) as $\{V_{\text{A},+}, V_{\text{A},-} = V_o, V_{\text{A,out}}\}$, and denote the inputs and outputs of op amp B (configured in a stable configuration with negative feedback) as $\{V_{\text{B},+} = V_o, V_{\text{B},-}, V_{\text{B,out}} = V_w\}$. Note that, since op amp B is wired with (stabilizing) negative feedback, $V_{\text{B,out}}$ adjusts so that $V_{\text{B},-} = V_{\text{B},+} = V_o = V_s/2$ at all times. Note also that resistors R_2 and R_3 form another voltage divider so that, taking[33] $R_2 \approx R_3$, it follows that $V_{\text{A},+} = (V_{\text{A,out}} + V_{\text{B,out}})/2$ at all times. As in the Schmitt trigger considered previously, there are two states to consider:

(a) $V_{\text{A,out}} = 0$. Assuming the capacitor is initially charged such that $V_{\text{B,out}} = 0$, current flows from the output of op amp B through C and R_1 to the output of op amp A, charging the capacitor until $V_{\text{B,out}} = V_s$ and thus $V_{\text{A},+}$ just[33] exceeds $V_{\text{A},-} = V_s/2$, and op amp A flips to state (b).

(b) $V_{\text{A,out}} = V_s$. Assuming the capacitor is initially charged such that $V_{\text{B,out}} = V_s$, current flows from the output of op amp A through R_1 and C to the output of op amp B, charging the capacitor until $V_{\text{B,out}} = 0$ and thus $V_{\text{A},+}$ falls just[33] below $V_{\text{A},-} = V_s/2$, and op amp A flips back to state (a).

The period of this oscillation is constant, and may be calculated by determining how long it takes the capacitor of the relaxation oscillator to gather sufficient charge to flip the switch in each state. For example, taking $V_{\text{A,out}} = 0$

[33] Note that R_3 should actually be chosen to be just slightly smaller than R_2, so that $V_{\text{A},+} = (0.5 - \epsilon)V_{\text{A,out}} + (0.5 + \epsilon)V_{\text{B,out}}$, and the states indeed flip as described. This can be achieved by selecting two resistors of the same rated resistance and, say, 5% variance, then carefully measuring the resistance of the two and and putting the one with smaller resistance in the R_3 location.

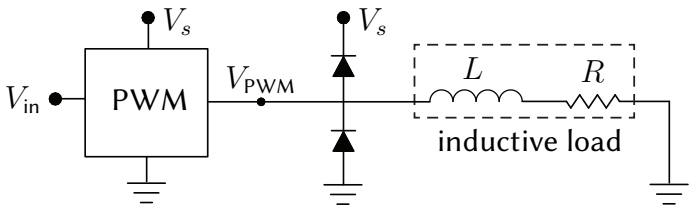


Figure 9.35: Application of a PWM circuit, formed by cascading the three stages of Figure 9.34, to an inductive load, incorporating a pair of protective **flyback diodes** to prevent voltage spikes from forming at V_{PWM} (and, possibly, an arc between exposed wires, and/or damage to one of the op amps) when the PWM circuit acts to quickly turn the power on and off to an inductive load.

and $V_{\text{B,out}}(0) = 0$, the dynamics of state (a) are governed by

$$C\frac{d}{dt}(V_{\text{B,out}} - V_{\text{B,}-}) = \frac{V_{\text{B,}-} - V_{\text{A,out}}}{R_1} \quad \Rightarrow \quad R_1\, C\, \frac{d}{dt}V_{\text{B,out}} = V_{\text{B,}-} - V_{\text{A,out}} \quad \Rightarrow$$
$$V_{\text{B,out}}(t) = \frac{1}{R_1\, C}(V_{\text{B,}-} - V_{\text{A,out}})t, \quad V_{\text{B,out}}(T/2) = V_s \quad \Rightarrow \quad T = 4\, R_1\, C.$$

Example 9.33 Efficient power control of inductive loads via pulse width modulation. The PWM strategy for driving loads at partial power, described above, is highly efficient and remarkably inexpensive to implement with modern electronics. If applied to a load with inductance, however, a problem is encountered. The PWM effectively acts as a switch, quickly turning on and off the power to (and, thus, the current through) the attached load. If the load contains an inductor governed by $V = L\ dI/dt$ [see (9.2c)], rapid changes in the current through the inductor would tend to induce large voltage spikes at V_{PWM}. As introduced in Figure 9.23, the diodes used in the circuit in Figure 9.35, called **flyback** (a.k.a. **snubber**, **freewheeling**, or **suppressor**) diodes, ensure (by providing a current path from ground, or to power, when necessary) that V_{PWM} does not exceed the range $-V_{\text{d}}$ to $V_s + V_d$, where V_{d} is the (small) cut-in voltage of the diode. *A good PWM circuit should always have such a flyback diodes incorporated, just in case the load attached to the PWM circuit has inductive elements.*

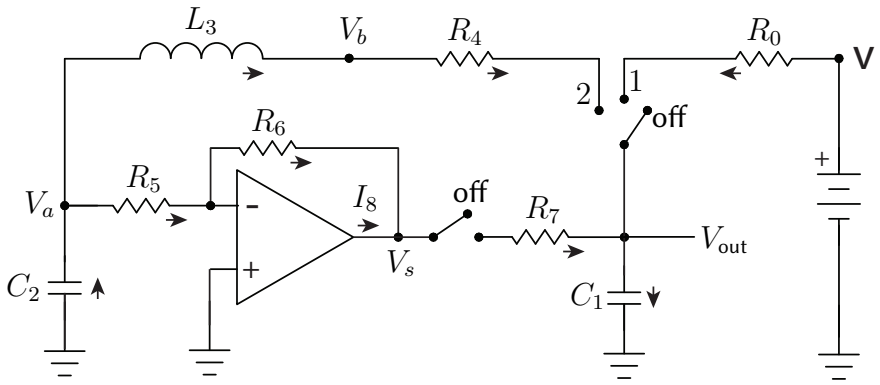


Figure 9.36: The **Colpitts oscillator** considered in Example 9.34.

Example 9.34 Colpitts oscillator. A Colpitts oscillator is an LC tank oscillator (see Example 9.12) with a simple transistor or op amp circuit implemented in order to offset the losses associated with the inevitable resistance of the real components in the circuit, thereby exhibiting sustained oscillations. We will consider here the case with an op amp implemented, as illustrated in Figure 9.36 (cf. Figure 9.12), initialized with both switches in the **off** position, current equal to zero everywhere, and all capacitors fully discharged.

Startup. Startup of the Colpitts oscillator, which occurs when we turn the switch from off to **position 1** (leaving the 2-way switch at **off**), is the same as the startup of the LC tank oscillator discussed in Example 9.12.

Decaying oscillations. Starting from the steady values of $V_{\text{out}}(t) = V_s$ and $I_1(t) = 0$, moving the 3-way switch from position 1 to **position 2** (leaving the 2-way switch at **off**) initiates sinusoidal oscillations that gradually decay due to the inevitable parasitic resistance of the components in the circuit. Analysis of these oscillations is the same as the analysis of the decaying oscillations of the LC tank oscillator discussed in Example 9.12.

Sustained oscillations. Some time after starting the oscillations discussed above, we turn the 2-way switch from off to **on** (leaving the 3-way switch at **position 2**), thereby attaching the op amp to the LC tank oscillator. Selecting $\{R_5, R_6, R_7\}$ appropriately, the net effect of hooking in this op amp is to provide just enough energy back into the oscillations, phased appropriately, to make up for the energy lost in R_4.

To derive the equations governing this circuit, we extend the analysis of the LC tank in Example 9.12, modifying the statements of KCL as appropriate (include I_8, the current coming out of the op amp), incorporating the component equations for $\{R_5, R_6, R_7\}$, and applying the relation $V_- = V_+ = 0$ to account for the behavior of the op amp, which is implemented in the stable (negative feedback) configuration. We assume that, when the 2-way switch is turned on [taken here as $t = 0$], $V_{\text{out}}(t=0) = V_{\text{out}}^0$, $V_a(t=0) = V_a^0$, and $I_3 = I_3^0$. The Laplace transforms of $\{V'_{\text{out}}(t), V'_a(t), I'_3(t)\}$ are thus $(s\,V_{\text{out}}(s) - V_{\text{out}}^0)$, $(s\,V_a(s) - V_a^0)$, and $(s\,I_3(s) - I_3^0)$, and our 12 governing eqns (5 KCLs and 7 components) are:

$$I_2(s) = I_3(s) + I_5(s), \quad I_3(s) = I_4(s), \quad I_4(s) + I_7(s) = I_1(s), \quad I_5(s) = I_6(s), \quad I_6(s) + I_8(s) = I_7(s),$$
$$I_1(s) = C_1[s\,V_{\text{out}}(s) - V_{\text{out}}^0], \quad I_2(s) = -C_2\,[s\,V_a(s) - V_a^0], \quad V_a(s) - V_b(s) = L_3\,[s\,I_3(s) - I_3^0],$$
$$V_b(s) - V_{\text{out}}(s) = R_4\,I_4(s), \quad V_a(s) = R_5\,I_5(s), \quad -V_s(s) = R_6\,I_6(s), \quad V_s - V_{\text{out}}(s) = R_7\,I_7.$$

Thus, 12 linear eqns in 12 variables $\{I_1(s), I_2(s), I_3(s), I_4(s), I_5(s), I_6(s), I_7(s), V_a(s), V_b(s), V_{\text{out}}(s)\}$, the first 11 of which are to be eliminated, as easily solved via Matlab (see code in RR.ch09), thus giving:

$$V_{\text{out}}(s) = \frac{b_2 s^2 + b_1 s + b_0}{s^3 + a_2 s^2 + a_1 s + a_0} = \frac{b_2 s^2 + b_1 s + b_0}{(s + a_3)[s^2 + a_4 s + a_5]} \quad \text{where} \tag{9.22}$$

$$a_2 = \frac{C_1\,C_2\,R_4\,R_5\,R_7 + C_2\,L_3\,R_5 + C_1\,L_3\,R_7}{C_1\,C_2\,L_3\,R_5\,R_7}, \quad a_1 = \frac{L_3 + C_2\,R_4\,R_5 + C_1\,R_4\,R_7 + C_1\,R_5\,R_7 + C_2\,R_5\,R_7}{C_1\,C_2\,L_3\,R_5\,R_7}, \quad a_0 = \frac{R_4 + R_5 + R_6 + R_7}{C_1\,C_2\,L_3\,R_5\,R_7}.$$

The values of $\{a_2, a_1, a_0\}$, all of which are seen to be positive, determine the behavior of the solution. The values of $\{b_2, b_1, b_0\}$ just determine the coefficients of the solution components that are present, based on the precise values of $\{V_{\text{out}}^0, V_a^0, I_3^0\}$, and are not considered further here.

Factoring the denominator of (9.22), a_3 may be determined via Cardano's formula: taking $q = (3\,a_1 - a_2^2)/3$, $r = (2\,a_2^3 - 9\,a_2\,a_1 + 27\,a_0)/27$, and $d = r^2/4 + q^3/27$, it follows that $a_3 = a_2/3 - \sqrt[3]{-r/2 + \sqrt{d}} - \sqrt[3]{-r/2 - \sqrt{d}}$, where $\sqrt[3]{x}$ denotes the principal (that is, the real) 3rd root of the real number x. It then follows immediately that $a_4 = a_2 - a_3$ and $a_5 = a_0/a_3$. The variables $\{a_3, a_4, a_5\}$ are determined from $\{C_1, C_2, L_3, R_4, R_5, R_6, R_7\}$ by the Barkhausen condition (see code in RR.ch09). It turns out that $a_3 > 0$ and $a_5 > 0$ [that is, the first term in the denominator of $V_{\text{out}}(s)$ is stable, and the second term is oscillatory] if all seven of these parameters are positive. The oscillatory term has:

- *positive* damping (thus, damped oscillations) if $a_4 > 0$, and
- *negative* damping (thus, sustained oscillations) if $a_4 < 0$ (this is known as the **Barkhausen criterion**).

If $a_4 < 0$, the oscillations grow until the output of the op amp reaches the minimum and maximum values of the power supply itself, after which the clipping of the op amp output [see (9.19)] reduces its effective gain slightly, and a periodic, essentially sinusoidal oscillation is established.

When designing a Colpitts oscillator of the form illustrated in Figure 9.36, one first selects $\{C_1, C_2, L_3\}$ such that the LC tank part of the circuit (see Example 9.12) oscillates at the desired frequency, $\omega_d = 1/\sqrt{L_3 C}$ where $1/C = 1/C_1 + 1/C_2$. Often a balanced configuration is chosen, with $C_1 = C_2$. In this design, a tradeoff is made, keeping both the capacitors and the inductor sufficiently small to minimize the overall size, cost, and parasitic resistance (modeled as R_4 in this analysis) of the components selected. Intermediate values of R_5 and R_7 are then assigned, to keep the current demands on the op amp relatively small. Finally, one then tunes R_6 in order to make a_4 slightly negative, thus sustaining the oscillations; for example, taking $C_1 = C_2 = 1\,\mu\text{F}$, $L_3 = 1\,\text{mH}$, $R_4 = 1\,\Omega$, $R_5 = 1\,\text{k}\Omega$, and $R_7 = 100\,\Omega$, it is found that $R_6 \approx 1325\,\Omega$ achieves this goal. Large negative values of Ω are not desired, as they put proportionally more work on the op amp (which costs power) and less on the passive components of the LC tank.

Note that, if R_7 is taken as zero, it is not possible to satisfy the Barkhausen criterion, and thus sustain oscillations with an op-amp reinforced Colpitts oscillator circuit. This is despite several (explicit or implied) false claims to the contrary online (as of 2021, do a Google image search yourself), and even a couple of well-meaning textbooks; Jakas & Llopis (2007) discuss this confusion in detail.

9.3.2 Constructing binary logic gates

9.3.3 Digital storage elements

9.4 Signal transmission

This section still under construction.

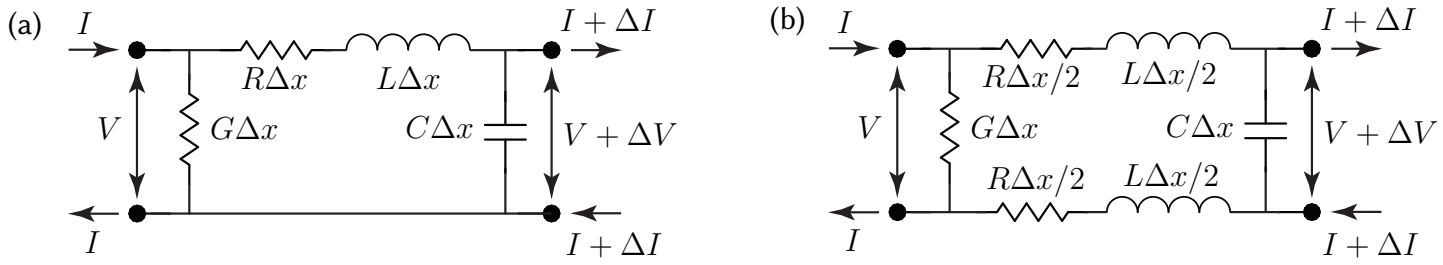


Figure 9.37: Two equivalent models of a section, of length Δx, of a pair of transmission wires.

9.4.1 Telegrapher's equation and characteristic impedance

Consider the models for the electrical characteristics of a pair of transmission wires illustrated in Figure 9.37. Note that, in this section (§9.4.1) only, we follow the dominant convention in the literature, and indicate by $\{R, L, G, C\}$ the resistance *per unit length*[34] (along the pair of wires), the inductance *per unit length* (along the pair of wires), the conductance[35] *per unit length* (between the two wires), and the capacitance *per unit length* (between the two wires). These quantities might more precisely be denoted as $\{dR/dx, dL/dx, dG/dx, dC/dx\}$, respectively, but that would be notationally too heavy in the derivation that follows.

The equations that govern the electrical signals that propagate down the pair of wires in this model (either Figure 9.37a or, equivalently, Figure 9.37b) may be written simply by expressing, over a short length of wire Δx, the change in voltage between the two wires, ΔV, and the change in current flowing within the wires, ΔI, using the basic "lumped parameter" component equations (9.2a)-(9.2c), which easily gives

$$\Delta V = -(R\,\Delta x)I - (L\,\Delta x)\frac{dI}{dt}, \qquad \Delta I = -(G\,\Delta x)V - (C\,\Delta x)\frac{dV}{dt}. \tag{9.23a}$$

Dividing both equations by Δx, then taking the limit as $\Delta x \to 0$, thus gives the telgrapher's equations

$$\frac{dV}{dx} = -\Big(R + L\frac{d}{dt}\Big)I, \qquad \frac{dI}{dx} = -\Big(G + C\frac{d}{dt}\Big)V. \tag{9.23b}$$

Taking d/dx of the first equation and inserting the second gives a scalar second-order form of this PDE,

$$\frac{d^2V}{dx^2} = \Big(R + L\frac{d}{dt}\Big)\Big(G + C\frac{d}{dt}\Big)V. \tag{9.24}$$

Following a **separation of variables** (SOV) solution approach, we seek solution modes of the **separable** form

$$V^m(x,t) = X^m(x)\,T^m(t). \tag{9.25}$$

Inserting the assumed form of the solution (9.25), aka the **solution ansatz**, into the PDE (9.24), and isolating the spatial dependence on one side of the equation and the time dependence on the other side, gives

$$X''\,T = R\,G\,X\,T + (R\,C + L\,G)X\,T' + L\,C\,X\,T'' \quad \Rightarrow \quad \frac{X''}{X} = \frac{R\,G\,T + (R\,C + L\,G)T' + L\,C\,T''}{T} = -k^2,$$

[34] The resistance per unit length of a wire (measured in ohms per meter, Ω/m) can be found by dividing the **resistivity** ρ of its conductive material (measured in ohm meters, Ω m) by its cross-sectional (measured in square meters, m^2).

[35] Conductance is simply the reciprocal of the resistance, and is sometimes a useful characteristic to use in settings (like, between to wires in a transmission line system) where the resistance between two points is either very large or essentially infinite. The **siemens** (**S**) is the unit of electric conductance in SI; a siemens (also referred to as a mho) is thus equal to the reciprocal of an ohm, $S = 1/\Omega$.

where k is a constant (that is, neither a function of x, nor a function of t). Note that we have dropped the $()^m$ superscripts, and the (x) and (t) dependencies, for notational clarity. We thus have

$$X'' + k^2 X = 0 \quad \Rightarrow \quad X(x) = X_+ e^{\mathrm{i}kx} + X_- e^{-\mathrm{i}kx}, \tag{9.26a}$$

$$a\,T'' + b\,T' + c\,T = 0 \quad \Rightarrow \quad T(t) = T_+ e^{\mathrm{i}\omega_+ t} + T_- e^{\mathrm{i}\omega_- x}, \tag{9.26b}$$

where $a = L\,C$, $b = R\,C + L\,G$, $c = R\,G + k^2$, and $-a\,\omega^2 + \mathrm{i}\,b\,\omega + c = 0$, and thus

$$L\,C\,\omega^2 - \mathrm{i}\,(R\,C + L\,G)\,\omega - R\,G = k^2 = \omega^2\,L\,C\Big(1 - \frac{\mathrm{i}\,(R\,C + L\,G)}{\omega\,L\,C} + \frac{\mathrm{i}^2 R\,G}{\omega^2\,L\,C}\Big),$$

and therefore

$$k = \omega\,\sqrt{L\,C}\,\sqrt{\Big(1 - \frac{\mathrm{i}\,R}{\omega\,L}\Big)\Big(1 - \frac{\mathrm{i}\,G}{\omega\,C}\Big)} \quad \Leftrightarrow \quad \mathrm{i}\,k = \sqrt{(\mathrm{i}\,\omega\,L + R)(\mathrm{i}\,\omega\,C + G)}; \tag{9.27}$$

this is known as a **dispersion relation** of the transmission line system. Note that,

$$\text{if } \frac{R}{\omega\,L} \ll 1 \text{ and } \frac{G}{\omega\,C} \ll 1, \text{ it follows that } k \approx \omega\,\sqrt{L\,C}; \tag{9.28}$$

this common case, with low resistance per unit length, R, along the wire, and low conductance (the inverse of resistance) per unit length, G, from one wire to the other, corresponds to modes [see (9.25) and (9.26)] of the general form $V^m(x,t) = Ae^{\mathrm{i}(kx+\omega t)} = Ae^{\mathrm{i}k(x+ct)}$, where the wave speed $c = \omega/k = 1/\sqrt{L\,C}$ is real (and, less than the speed of light, which is the speed that electrons themselves move). In other words, this case is characterized by waves that propagate without distortion in shape[36]. If one or both of the terms at left in (9.28) is not small, then the dispersion relation is not a simple real, linear relationship between k and ω, and the wave suffers some distortion (in shape) and attenuation (reduction in magnitude) as it travels down the wire.

We now put the voltage and corresponding current components of the solution back together. By (9.25), (9.26), and (9.23b), one component of the voltage and current modal solutions [other modal components can be written with different signs on k and ω] may be written

$$V^m(x,t) = V_o\,e^{\mathrm{i}kx}e^{\mathrm{i}\omega t}, \quad I^m(x,t) = -I_o\,e^{\mathrm{i}kx}e^{\mathrm{i}\omega t}, \tag{9.29}$$

where, by both equations given in (9.23b) and the expression at right in (9.27), we may write

$$\left.\begin{aligned} \mathrm{i}\,k\,V_o = (\mathrm{i}\,\omega\,L + R)I_o \quad &\Rightarrow \quad \frac{V_o}{I_o} = \frac{(\mathrm{i}\,\omega\,L + R)}{\mathrm{i}\,k} \\ \mathrm{i}\,k\,I_o = (\mathrm{i}\,\omega\,C + G)V_o \quad &\Rightarrow \quad \frac{V_o}{I_o} = \frac{\mathrm{i}\,k}{(\mathrm{i}\,\omega\,C + G)} \end{aligned}\right\} \quad \Rightarrow \quad \frac{V_o}{I_o} = \sqrt{\frac{\mathrm{i}\,\omega\,L + R}{\mathrm{i}\,\omega\,C + G}} = Z_o; \tag{9.30}$$

note that the **characteristic impedance** of the system, Z_o, is measured in ohms. Note also that,

$$\text{if } \frac{R}{\omega\,L} \ll 1 \text{ and } \frac{G}{\omega\,C} \ll 1, \text{ it follows that } Z_o = \frac{V_o}{I_o} \approx \sqrt{\frac{L}{C}}. \tag{9.31}$$

Again, this case, with low resistance per unit length, R, along the wires, and low conductance (the inverse of resistance) per unit length, G, from one wire to the other, is quite common. Note that, in this case, the characteristic impedance Z_o (measured in ohms) arises primarily due to the L and C terms in the model [see (9.23)] of the transmission line system.

Typical values (for Cat 5e ethernet cable; other cables are similar) are $L = 525$ nH/m and $C = 52$ pF/m, and thus $Z_o = \sqrt{L/C} = \sqrt{(525 \cdot 10^{-9})/(52 \cdot 10^{-12})} = 100$ ohm and $c = 1/\sqrt{L\,C} = 1.9 \cdot 10^8$ m/s. Note that this wave propagation speed c is about 2/3 of the speed of light in a vacuum, which is $2.99792 \cdot 10^8$ m/s.

[36] How much of each mode is present in the overall solution, which may be formed as a superposition of modes of the form given in (9.25), is a function of the boundary conditions and initial conditions on the system, which we do not explore further here.

Example 9.35 Termination, impedance matching, and signal distortion. Consider the transmission line model illustrated in Figure 9.37a, with the voltage of the lower wire defined as ground, and with a specified $\{R, L, G, C\}$ (see §9.4.1). We now model the voltage $V(x,t)$ and current $I(x,t)$ propagating along the upper wire by discretizing the linear PDE (9.23b), rearranged into the form

$$\frac{d}{dt}\begin{pmatrix} V \\ I \end{pmatrix} = -\begin{pmatrix} G/C & (1/C)\,d/dx \\ (1/L)\,d/dx & R/L \end{pmatrix}\begin{pmatrix} V \\ I \end{pmatrix}, \tag{9.32}$$

starting from the initial condition that the wire starts out with zero voltage and zero current everywhere, that is, $V(x, t=0) = I(x, t=0) = 0$. We will apply a boundary condition on the voltage at the left end of the wire, $V(x=0,t)$, that transitions fairly smoothly from $V = 0$ to $V = 2$ after $t = 0$, according to

$$V(x=0,t) = \begin{cases} 1 - \cos(\pi t/t_1) & 0 \le t \le t_1, \\ 2 & t_1 \le t, \end{cases} \tag{9.33}$$

where $t_1 = 10^{-8}$ s. We will initially, in Case A below, consider a simple case with a zero boundary condition on the current at the right end of the wire, $I(x=X,t)$. [We will change this in Example 9.35.]

We first prepare to numerically simulate both wave components, $V(x,t)$ and $I(x,t)$, as they propagate down the transmission line for $0 < x < X$ and $t > 0$. We do this by discretizing the PDE + BCs with simple numerical methods (see §9), as described below, using sufficiently small grid spacing in time and space (that is, small Δt and Δx) in the numerical discretization such that reducing Δt and Δx further does not substantially change the numerical result. Other numerical methods would certainly be more efficient and better behaved for larger Δx and Δt, but the simple approach implemented here proves to be adequate.

We start by discretizing $V(x,t)$ on a spatial grid with $x_i = i\Delta x$, denoting $V_i(t) = V(x_i, t)$. Noting that (9.32) is characterized by *first-order derivatives* in space, we discretize I on a grid that is *staggered* with respect to the grid used to discretize V, denoting $I_{i+0.5}(t) = I(x_{i+0.5}, t)$ where $x_{i+0.5} = (i + 0.5)\Delta x$ for integer i.

The grid is defined by taking $\Delta x = X/(N + 0.5)$, so the rightmost gridpoint, at $x = X$, is the gridpoint corresponding to $I_{N+1/2}(t)$; this will make implementing the BC $I(x=X,t)$ particularly easy. Note that, using this grid, there are N integer gridpoints on the interior, at which $V_1(t)$ to $V_N(t)$ are defined, and there are N fractional gridpoints on the interior, at which $I_{0.5}(t)$ to $I_{N-0.5}(t)$ are defined.

We apply the BC on $V_0(t)$ as a forcing term on the RHS, and note that the BC $I(x = X, t) = 0$ may be enforced simply by dropping this term from the discretized equations. We discretize the PDE in (9.32) with second-order central discretizations of the dI/dx and dV/dx terms; the staggered grid implemented makes this approach work particularly well. Defining $d = 1/\Delta x$, this is accomplished with a linear system of the general form

$$d\mathbf{x}/dt = A\mathbf{x} + BV_0(t) \tag{9.34}$$

$$\mathbf{x}(t) = \begin{pmatrix} I_{0.5}(t) \\ V_1(t) \\ I_{1.5}(t) \\ V_2(t) \\ \vdots \\ I_{N-0.5}(t) \\ V_N(t) \end{pmatrix}, \quad A = \begin{pmatrix} -R/L & -d/L & & & & & & 0 \\ d/C & -G/C & -d/C & & & & & \\ & d/L & -R/L & -d/L & & & & \\ & & d/C & -G/C & -d/C & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & d/L & -R/L & -d/L & \\ 0 & & & & & d/C & -G/C & \end{pmatrix}, \quad B = \begin{pmatrix} d/L \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}.$$

Denoting $h = \Delta t$, $t_n = hn$, $\mathbf{x}_n = \mathbf{x}(t_n)$, and $V_{0,n} = V_0(t_n)$, a CN method is used to propagate (9.34):

$$\frac{\mathbf{x}_n - \mathbf{x}_{n-1}}{h} = \frac{A}{2}\Big[\mathbf{x}_n + \mathbf{x}_{n-1}\Big] + B\,V_{0,n-0.5} \quad \Rightarrow \quad \Big[I - \frac{A\,h}{2}\Big]\mathbf{x}_n = \Big[I + \frac{A\,h}{2}\Big]\mathbf{x}_{n-1} + B\,h\,V_{0,n-0.5} = \mathbf{r}_{n-0.5}.$$

The system $[I - A\,h/2]\,\mathbf{x}_n = \mathbf{r}_{n-0.5}$ may be solved efficiently at each timestep using a reduced form of Gaussian elimination that accounts for the fact that A is tridiagonal and, for sufficiently small h, diagonally dominant. A simple code that performs the simulation described above, but quite *inefficiently* (but, good enough for the present purposes, if you have a fast computer, and choose the number of gridpoints used, N, to be sufficiently small), is given in Algorithm **??**; this code simply calculates `x=D\r` at each timestep, where $D = I - A\,h/2$ and $\mathbf{r} = [I + A\,h/2]\mathbf{x} + B\,hV_{0,n-0.5}$. Egregiously, the code stores D as a full matrix, which indeed is *not at all* efficient. The implementation of a *computationally efficient* numerical method for this simulation, leveraging the Thomas algorithm, is discussed in Example **??**.

The simulations listed below take parameter values typical values for Cat 5e ethernet cable ($R = G \approx 0$, $L = 525 \cdot 10^{-9}$ H/m, and $C = 52 \cdot 10^{-12}$ F/m, and thus $c = 1.9 \cdot 10^8$ m/s and $Z_0 = 100\,\Omega$; see the last paragraph of §9.4.1). We will consider a wire length of $X = 10$ m, and thus expect the wave traveling along the wire to begin to reach the rightmost boundary at $T = X/c = 5.22 \cdot 10^{-8}$ s. We will use a timestep of $h = \Delta t = 4 \cdot 10^{-11}$ s, and a spatial grid spacing with $N = 200$, and thus $\Delta x = X/(N + 0.5) = 0.049875$ m.

Case A: No termination. In this case, the boundary condition at the right edge of the domain is taken as $I(x = L, t) = I_{N+1/2}(t) = 0$ [that is, no significant current into the downstream component, which accurately models a connection to the input of an op amp]. In the representation (9.34), this term is simply set to zero, so the last component of the $\mathbf{x}$ vector is $V_N(t)$, and the dependence of the evolution equation on $I_{N+1/2}(t)$ is simply dropped.

The resulting wave propagation is illustrated in Figure 9.38. Note that the wave is reflected back and forth across the domain indefinitely, causing spurious echos in the communication channel (akin to talking over a phone line or PA system in the presence of undamped feedback).

Case B: Termination with a resistor. In this example, the boundary conditions at the right edge of the domain is changed to model the effect of a single resistor, with resistance Z_0, installed between the two wires at $x = X$. We do this by taking $I_{N+1/2}(t) = (1/Z_0)V_N(t)$.

Again taking $R = G = 0$, $L = 525 \cdot 10^{-9}$, and $C = 52 \cdot 10^{-12}$, the resulting wave propagation is illustrated in Figure 9.39. Note that the wave reflections have been eliminated. Problem solved. △

Example 9.36 Wire junction. In the... △

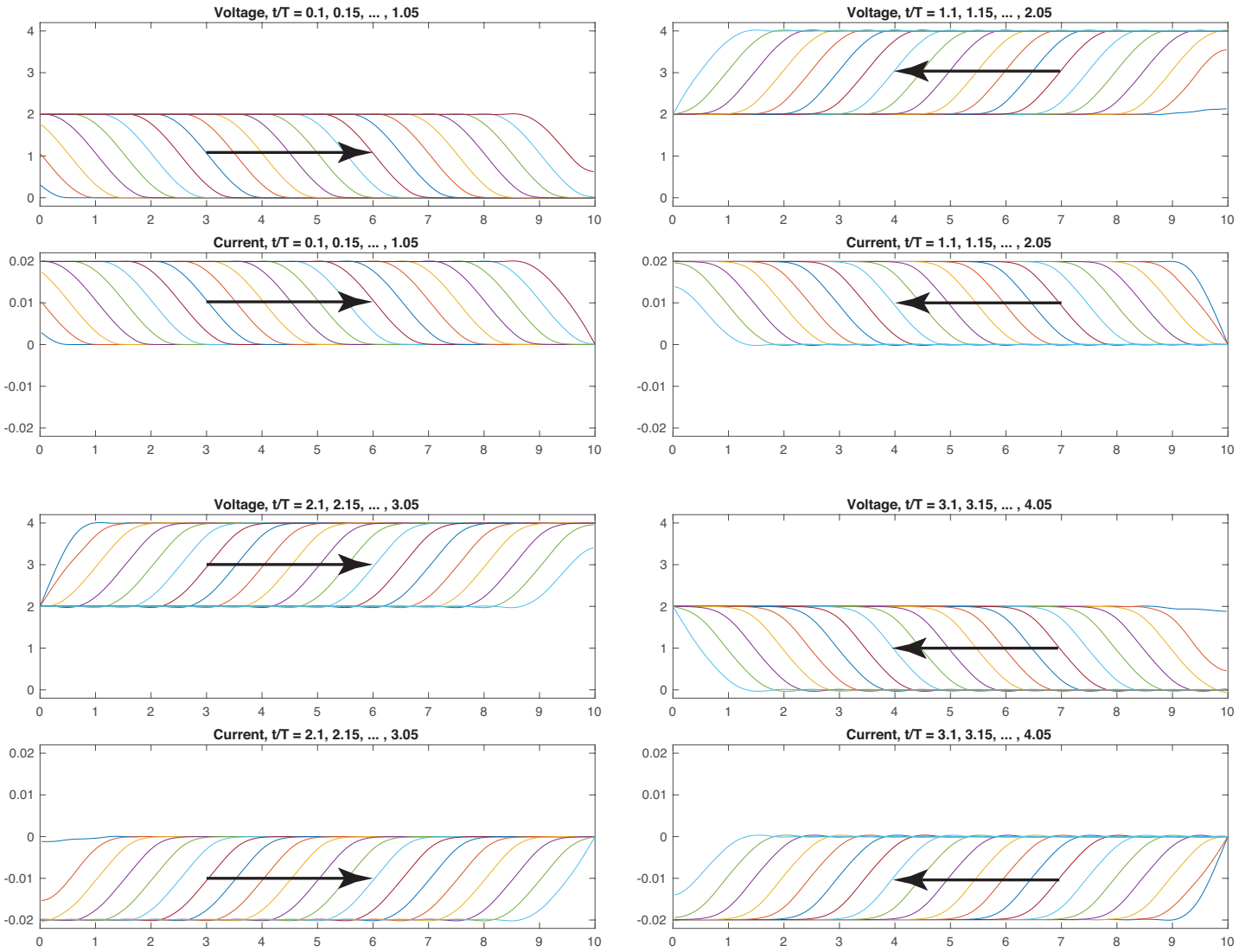


Figure 9.38: Transmission of a signal down a wire as in Case A of Example 9.35, with **no termination**. Note that, if $R = G = 0$, the signal reflects back and forth over the wire indefinitely.

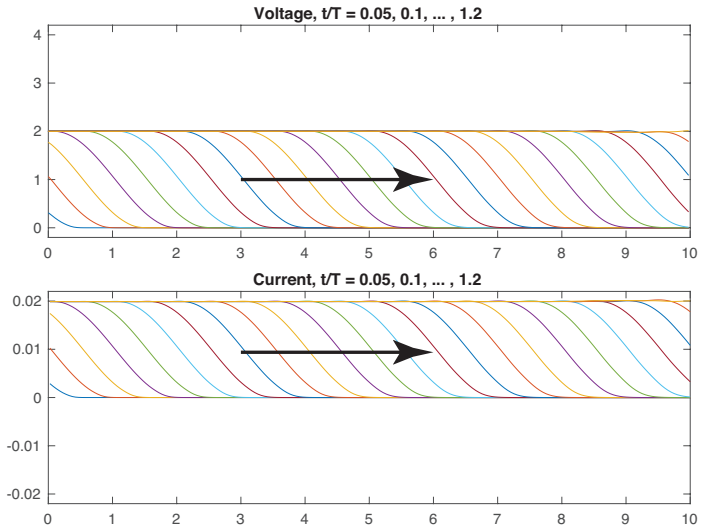


Figure 9.39: Transmission of a signal as in Example 9.35, implementing a **terminating** resistor of Z_0 ohms. The signal reflection is eliminated; after $t/T = 1.2$, $V(x,t) \approx 2$ and $I(x,t) \approx 0.02$ along the entire wire.

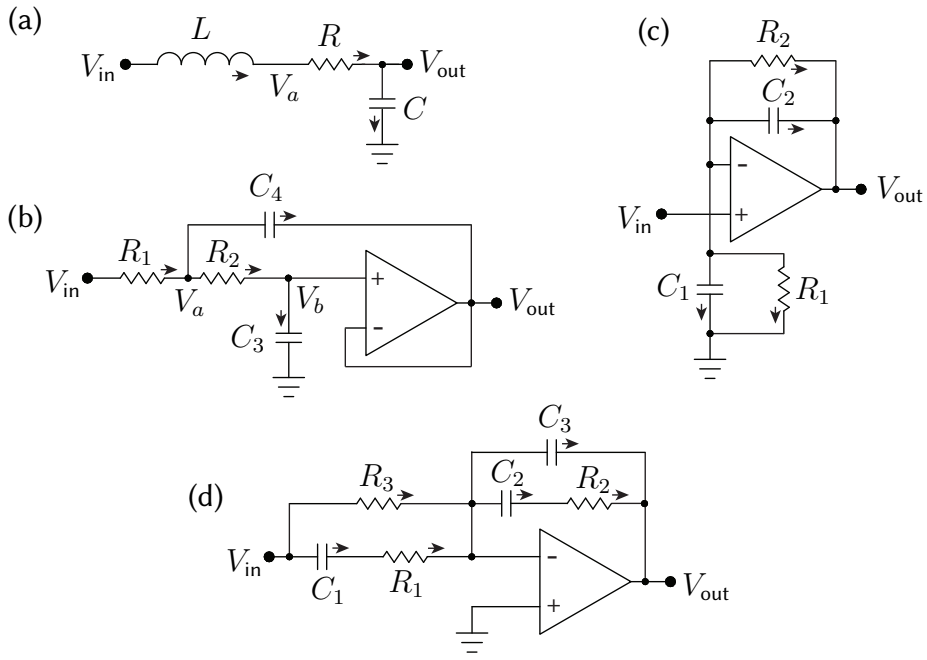


Figure 9.40: Some dynamic filters considered in the exercises. (a) A passive second-order low-pass filter. (b) An active second-order low-pass filter. (c) A general-purpose noninverting first-order filter. (d) A single op-amp implementation of a PID filter combined with two first-order low-pass filters.

Exercises

[*Note: Put all numerical codes you write to solve these exercises, with appropriate (succinct, clear) comments to make them readable, in a private repository in your account on Github, share this repository with the course instructors, and provide links to these codes in the pdfs of your homework writeups that you submit for the class.*]

Exercise 9.1 Following an analogous derivation as that in Example 9.4, replace the six resistors in Figure 9.5a-b with six capacitors $\{C_1, C_2, C_3, C_a, C_b, C_c\}$, and write a code `Wye_Delta_Capacitors.m` that quantifies $\{C_a, C_b, C_c\}$ in terms of $\{C_1, C_2, C_3\}$, and vice versa, so that the two circuits are equivalent. Then, replacing the five resistors in Figure 9.6d with five capacitors $\{C_1, C_2, C_3, C_4, C_5\}$ and applying this result, compute the equivalent capacitance C of this network. *Note: Given the simple path to solution in this problem using symbolic manipulation, the engineering student is encouraged to to crank through such tedious algebraic manipulations symbolically whenever possible from now on!*

Exercise 9.2 Following an analogous derivation as that in Example 9.6, given one inductor L_2 of a precisely known inductance, write a code `Wheatstone_Inductors.m` that quantifies how a Wheatstone bridge may be used to measure the inductance of an unknown inductor L_5.

Exercise 9.3 Following an analogous derivation as that in Example 9.8, compute the power provided or absorbed by the voltage and current sources of Figure 9.8 without making the assumption that $R_1 = I_L = 0$.

Exercise 9.4 Much useful information can be gleaned from device datasheets. This exercise considers just three examples for the TI LMR62014 discussed in Example 9.24:

• By the figure on page 2 of this datasheet, what values of R_1 and R_2 are recommended for the voltage divider to implement feedback (into the device FB pin) that boosts the output to $V_{out}^{mean} \approx 12\,V$?

• By the figure on page 1 of the datasheet, what power efficiency is anticipated for $V_{\text{in}} = 5\,\text{V}$, $V_{\text{out}}^{\text{mean}} = 12\,\text{V}$, and a load of $R = 250$ ohms?
• By page 9 of the datasheet: what is the purpose of the input capacitor C1 in the TI LMR62014 circuit?

Exercise 9.5 (a) Compute the transfer function $V_{\text{out}}(s)/V_{\text{in}}(s)$ of the passive circuit shown in Figure 9.40a, making the same two assumptions as in Example 9.2. Assuming $L\,C = 1$ and $R\,C = 0.5$, plot its Bode plot. What is the corner frequency and damping of this filter?
(b) Compute the transfer function $V_{\text{out}}(s)/V_{\text{in}}(s)$ of the active circuit shown in Figure 9.40b, again making the same two assumptions as in Example 9.2. Assuming $R_1 = 10\,\text{k}\Omega$ and that the op amp is ideal with amplification $A \to \infty$, what values of $\{R_2, C_1, C_2\}$ result in the same frequency response as the passive filter considered in part (a)? Assuming all of the components are readily available (they are!), what advantages does a circuit of the type considered in part (b) have over the circuit considered in part (a)?
(c) Note that a fourth-order low-pass filter may be constructed simply by cascading together two second-order active low-pass filters of the type considered in part (b). Following the development in §8.5.1, design an active fourth-order low-pass Butterworth filter and an active fourth-order low-pass Bessel filter, both with $\omega_c = 100$ Hz. Specify the resistor and capacitor values used in each design.
(d) In the active circuit considered in part (b), replace the resistors with capacitors and the capacitors with resistors. Compute the corresponding transfer function and discuss.

Exercise 9.6 Example 9.30 developed a flexible general-purpose *inverting* first-order filter. Sketch the Bode plots of the nine filters that the circuit in Example 9.30 reduces to in the nine special cases enumerated. Then, develop a similarly flexible *noninverting* first-order filter, which is a bit more involved. Start by performing a careful analysis of the circuit in Figure 9.40c, and describe which of the nine cases itemized in the inverting case (Example 9.30) are realizable with this noninverting circuit. Identify precisely what limitations (if any) are present in each case. Reviewing the three special cases considered in Example 9.28, describe precisely how the limitations of this circuit may be circumvented by reconnecting it appropriately.

Exercise 9.7 The circuit in Figure 9.40d (cf. Figure 9.32a) has a transfer function of

$$\frac{V_{\text{out}}(s)}{V_{\text{in}}(s)} = -K\frac{(s/z_1+1)(s/z_2+1)}{s}\cdot\frac{1}{(s/p_1+1)(s/p_2+1)},$$

and thus may be interpreted as an inverting PID filter combined with two first-order low-pass filters. Determine how each of the variables $\{K, z_1, z_2, p_1, p_2\}$ depend on $\{R_1, R_2, R_3, C_1, C_2, C_3\}$ (show your work, or provide the symbolic code you write to solve the problem for you). Then, solve for $\{R_1, R_2, C_1, C_2, C_3\}$ in terms of $\{K, z_1, z_2, p_1, p_2, R_3\}$.

Exercise 9.8 Replace $\{C_1, C_2, L_3\}$ with $\{L_1, L_2, C_3\}$, respectively, in Example 9.34 (that is, replace the capacitors with inductors, and replace the inductor with a capacitor, thereby forming a **Hartley oscillator**), and repeat the example in its entirety, following closely the same analysis.

Chapter 10

Classical Control

10.1 Closing the loop: an introduction to feedback control design

As illustrated in Figure 10.1, the problem of **feedback control design** amounts to the design a **controller** [denoted $D(s)$ or $D(z)$] that coördinates the **control input**(s) $\mathbf{u}$ of the **plant**[1] [denoted $G(s)$ or $G(z)$] with the **measurement**(s) $\mathbf{y}$ of the plant in such a way as to deliberately *change* the dynamics otherwise exhibited by the system, optimizing some balance of the closed-loop system's **performance** (that is, the ability of the closed-loop system to track a desired **reference input** $\mathbf{r}$ with sufficient accuracy) and its **robustness** (that is, the insensitivity of the closed-loop system response to **state disturbances** $\mathbf{w}$, **measurement noise** $\mathbf{v}$, and **modeling errors** Δ in the plant itself). The performance and robustness measures of interest, as well as the balance between these two generally competing objectives, must be defined carefully in any given application. The large variety of possible systems that one might consider (ODE, PDE, DAE, linear, nonlinear, etc.), the wide range of performance and robustness measures of possible interest, the numerous balances between these measures that one might attempt to achieve, and various practical restrictions on actuator authority (saturation and bandwidth limits) and limitations in the controller design (sample time, decentralized communication architecture, computational complexity, and varying degrees and types of uncertainty in the plant itself) give rise to a rich variety of possible control strategies that have been and will continue to be developed. The brief introduction to the feedback control problem presented in this chapter, which surveys some of the key issues and foundational ideas, intends to serve as a prologue to a more in-depth study of this rich field.

A typical **performance specification** for a CT or DT LTI system is the prescription of the minimum **rise time** and **settling time**, and the maximum **overshoot** and **steady-state error**, of the closed-loop system's step response (see Figure 8.3): that is, the response $\mathbf{y}$ to a step reference input $\mathbf{r}$ to the system depicted in Figure 10.1. A typical **robustness specification** for such a system is the minimization of the response of the system to external **state disturbances** $\mathbf{w}$ and **measurement noise** $\mathbf{v}$ that might otherwise disrupt the system. A balance between these simple performance and robustness specifications on the closed-loop behavior of a SISO LTI system forms a starting point for the study of the feedback control problem to be presented in this chapter. As a rule of thumb, intentionally a bit loosely stated, the following is a good starting point:

Guideline 10.1 *Apply just enough control feedback to approximately achieve the performance specification.*

Pushing a system harder than this with control excitation generally degrades robustness to a host of possible unmodeled effects, as described in §10.1.1. Time delays are especially dangerous (see Examples 10.2 and 10.11).

[1] The ubiquitous use of the name **plant** for the system to be controlled is historical, reflecting the initial applications of linear systems theory to chemical process control.

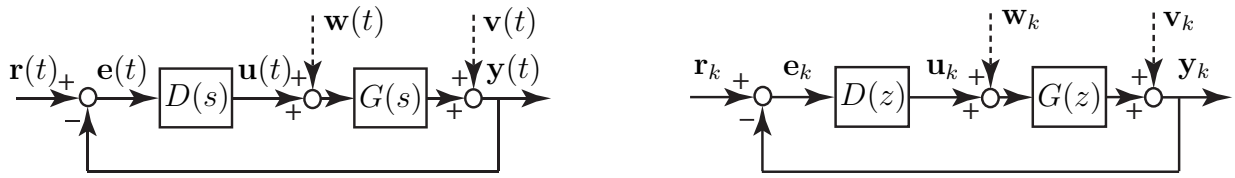


Figure 10.1: **Closed-loop** (a.k.a. **feedback**) connections of (left) a **continuous-time** (**CT**) LTI plant $G(s) = Y(s)/U(s)$ and controller $D(s) = U(s)/E(s)$, and (right) a **discrete-time** (**DT**) LTI plant $G(z) = Y(z)/U(z)$ and controller $D(z) = U(z)/E(z)$. In such **block diagrams**, for clarity, we denote the signals in the time domain and the systems in the transformed domain (that is, in **transfer function form**). The sign convention chosen at the leftmost summing junction is such that the **error signal** $\mathbf{e} = \mathbf{r} - \mathbf{y}$. For the (practical) closed-loop connection of a DT controller with a CT plant, see Figures 10.34 and 10.35.

In the CT SISO case, the Laplace transform of the output, $Y(s)$, is related to the Laplace transform of the input, $R(s)$, in the absence of state disturbances $\mathbf{w}$ and measurement noise $\mathbf{v}$ as follows:

$$Y(s) = G(s)\,U(s) = G(s)\,D(s)\,E(s) = G(s)\,D(s)\,[R(s) - Y(s)] \;\Rightarrow\; T(s) \triangleq \frac{Y(s)}{R(s)} = \frac{G(s)\,D(s)}{1 + G(s)\,D(s)} \tag{10.1a}$$

The matrix $T(s)$ [in some texts denoted $H(s)$] is often referred to as "the" **closed-loop transfer function** of the system, and may be analyzed in order to ensure the desired performance specifications, such as those mentioned above. Note that, if $G(s)$ and $D(s)$ are rational functions of s, then we may write, e.g.,

$$G(s) = \frac{b(s)}{a(s)}, \quad D(s) = \frac{y(s)}{x(s)} \quad \Rightarrow \quad T(s) = \frac{b(s)\,y(s)}{a(s)\,x(s) + b(s)\,y(s)} = \frac{g(s)}{f(s)}, \tag{10.1b}$$

where $\{a(s), b(s), x(s), y(s), g(s), f(s)\}$ are polynomials. That is, $T(s)$ is a rational function of s as well. Thus, once simplified appropriately, the closed-loop transfer function $T(s)$ may be analyzed with the various techniques described in §8.2. It is, therefore, the roots of the *denominator* of $T(s)$ [that is, the **poles of the closed-loop transfer function**] which dictate the *nature* (that is, the speed of oscillation and the rate of exponential growth or decay) of each component of the response. The roots of the *numerator* of $T(s)$ [that is, the **zeros of the closed-loop transfer function**] dictate only the *coefficients* of each of these components. Via similar derivations, the following additional transfer functions may also be identified:

$$\frac{E(s)}{R(s)} = \frac{1}{1 + G(s)\,D(s)} = S(s), \quad \frac{U(s)}{R(s)} = \frac{D(s)}{1 + G(s)\,D(s)} = S_u(s), \quad \frac{Y(s)}{V(s)} = \frac{1}{1 + G(s)\,D(s)} = S(s),$$
$$\frac{Y(s)}{W(s)} = \frac{G(s)}{1 + G(s)\,D(s)} = S_i(s), \quad \frac{U(s)}{V(s)} = \frac{-D(s)}{1 + G(s)\,D(s)} = -S_u(s), \quad \frac{U(s)}{W(s)} = \frac{-G(s)\,D(s)}{1 + G(s)\,D(s)} = -T(s).$$

These transfer functions may be analyzed in order to ensure the desired robustness specifications. As a mnemonic, the closed-loop transfer function of any feedback loop is given by

$$\textbf{transfer function} = \frac{\textbf{forward gain}}{1 - (\textbf{loop gain})}. \tag{10.2}$$

Note also that the poles of the closed-loop transfer function $T(s)$ in (10.1) are given by the values of s for which

$$1 + G(s)\,D(s) = 0 \qquad \Leftrightarrow \qquad f(s) = a(s)\,x(s) + b(s)\,y(s) = 0. \tag{10.3}$$

Once the transfer function of a CT controller, $D(s) = U(s)/E(s)$, is designed, it is easy to compute the inverse Laplace transform of $D(s)$ to find the **differential equation** [relating $\mathbf{u}(t)$ to $\mathbf{e}(t)$] that the controller must

obey. It is also straightforward (see §9) to build an **electric circuit** that obeys this differential equation with an inexpensive arrangement of resistors, capacitors, and operational amplifiers.

In the DT case, the closed-loop transfer functions are derived in an *identical* fashion as in the CT case, with the role of z replacing that of s. Thus, the closed-loop transfer function considered to analyze performance is

$$T(z) \triangleq \frac{Y(z)}{R(z)} = \frac{G(z)\,D(z)}{1+G(z)\,D(z)} = \frac{b(z)\,y(z)}{a(z)\,x(z)+b(z)\,y(z)} = \frac{g(z)}{f(z)},$$

and the closed-loop transfer functions considered to characterize robustness again include all those identified above, with z replacing s. As in the CT case, once simplified appropriately, these closed-loop transfer functions are rational functions of z, and thus may be analyzed with the techniques described in §8.3. Furthermore, once the transfer function of a DT controller, $D(z)$, is designed, it is easy to compute the inverse Z transform of $D(z)$ to determine the **difference equation** [relating $\mathbf{u}_k$ to $\mathbf{e}_k$] that the corresponding controller must obey. It is also straightforward to implement this difference equation on an inexpensive **microcontroller** (see §1.5).

10.1.1 Fundamental limitations†

The four **sensitivities** derived above are closely related [i.e., you can't change them independently via $D(s)$]:

$$\begin{aligned} S(s) &= \frac{1}{1+G(s)\,D(s)} = \frac{a(s)\,x(s)}{a(s)\,x(s)+b(s)\,y(s)}, & S_u(s) &= \frac{D(s)}{1+G(s)\,D(s)} = \frac{a(s)\,y(s)}{a(s)\,x(s)+b(s)\,y(s)}, \\ T(s) &= \frac{G(s)\,D(s)}{1+G(s)\,D(s)} = \frac{b(s)\,y(s)}{a(s)\,x(s)+b(s)\,y(s)}, & S_i(s) &= \frac{G(s)}{1+G(s)\,D(s)} = \frac{b(s)\,x(s)}{a(s)\,x(s)+b(s)\,y(s)}; \end{aligned} \tag{10.4}$$

$S(s)$ is called the **sensitivity**, $T(s)$ is called (in this setting) the **complementary sensitivity**, $S_u(s)$ is called the **control sensitivity**, and $S_i(s)$ is called the **output sensitivity**. Note that, in SISO systems,

- $Y(s)/R(s) = -U(s)/W(s)$; thus, the response of the control $U(\mathrm{i}\omega)$ to disturbances $W(\mathrm{i}\omega)$ is suppressed only at those frequencies ω for which the closed-loop tracking is poor (thus motivating Guideline 10.1).
- $S_u(s) = D(s)/[1+G(s)\,D(s)] = T(s)/G(s)$; thus, at frequencies characterized by good tracking ($T(\mathrm{i}\omega) \approx 1$) but low plant gain ($|G(\mathrm{i}\omega)| \ll 1$), large control gains $D(\mathrm{i}\omega) \gg 1$ are needed.
- $S(s) = 1 - T(s)$; thus, the sensitivity $Y(\mathrm{i}\omega)/V(\mathrm{i}\omega)$ is suppressed only at those frequencies ω for which the complementary sensitivity $U(\mathrm{i}\omega)/W(\mathrm{i}\omega)$ is not.

The sensitivities defined in (10.4) are related such that $S_i(s) = S(s)\,G(s)$ and $T(s) = S_u(s)\,G(s)$. Thus:

Fact 10.1 (Internal stability of closed-loop SISO systems) *A closed-loop SISO system [see (10.1a) - (10.4)] is said to be* **internally stable** *if the sensitivities $\{T(s), S(s), S_u(s), S_i(s)\}$ are all stable; in such systems,*

(a) the poles of $G(s)$ appear either as zeros of $S(s)$ or (if they are in the LHP) possibly as poles of $S_i(s)$;
(b) the zeros of $G(s)$ appear either as zeros of $S_i(s)$ or (if they are in the LHP) possibly as poles of $S(s)$;
(c) the poles of $G(s)$ appear either as zeros of $S_u(s)$ or (if they are in the LHP) possibly as poles of $T(s)$;
(d) the zeros of $G(s)$ appear either as zeros of $T(s)$ or (if they are in the LHP) possibly as poles of $S_u(s)$.

If a root of $a(s)$ is also a root of $y(s)$, it is a root of $a(s)\,x(s)+b(s)\,y(s)$; internal stability thus requires:

(e) only poles of $G(s)$ that are in the LHP can appear as zeros of $D(s)$.

If a root of $b(s)$ is also a root of $x(s)$, it is a root of $a(s)\,x(s)+b(s)\,y(s)$; internal stability thus requires:

(f) only zeros of $G(s)$ that are in the LHP can appear as poles of $D(s)$.

In other words, internal stability requires that all pole/zero cancellations between $G(s)$ and $D(s)$ be in the LHP.

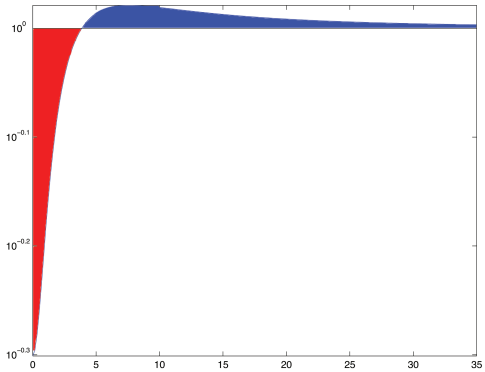


Figure 10.2: Semilogy plot of the log of the sensitivity $S(s) = 1/[1 + L(s)]$ for $L(s) = K/[(s+1)(s+10)]$ with $K = 10$, illustrating Bode's Integral Theorem: that is, the (red) region below the $\ln|S(\mathrm{i}\omega)| = 0$ line has the same area as the (blue) region above it, and thus $\int_0^\infty \ln|S(\mathrm{i}\omega)|\, d\omega = 0$, independent of K.

Taking a perturbed plant $G_\Delta(s) = G(s)[1 + \Delta(s)]$ with (multiplicative) modeling errors $\Delta(s)$, and defining

$$\delta(s) = 1/[1 + T(s)\,\Delta(s)], \tag{10.5a}$$

it follows from their definitions in (10.4) that the sensitivities of the perturbed plant $G_\Delta(s)$ are given by

$$\begin{aligned} S_\Delta(s) &= S(s)\,\delta(s), & S_{u\Delta}(s) &= S_u(s)\,\delta(s), \\ T_\Delta(s) &= T(s)\,[1 + \Delta(s)]\,\delta(s) & S_{i\Delta}(s) &= S_i(s)\,[1 + \Delta(s)]\,\delta(s); \end{aligned} \tag{10.5b}$$

Noting (10.5), it is seen that good tracking [$T(\mathrm{i}\,s\omega) \approx 1$] implies $O(1)$ susceptibility of all four of the sensitivity functions to destabilizing multiplicative modeling errors, risking instability. Modeling errors $\Delta(\mathrm{i}\omega)$ generally increase in magnitude with rising frequency ω; thus, to decrease the risk of closed-loop instability due to such modeling errors, the closed-loop bandwidth ω_{BW} [that is, the frequency ω above which $T(\mathrm{i}\omega)$ drops off] should be made as low as possible, again motivating Guideline 10.1.

Another class of fundamental limitations arises by integrating the log of the sensitivity over all frequencies. An example limitation of this class is given in Fact 10.2 below [recall from §8 that a stable CT transfer function has all of its poles in the LHP, that the **relative degree** n_r of a transfer function is the degree of the polynomial in its denominator minus the degree of the polynomial in its numerator, and that a CT transfer function is said to be **proper** if $n_r \geq 0$, and **strictly proper** if $n_r > 0$]:

Fact 10.2 (Bode's Integral Theorem) *Consider a stable strictly proper open-loop system $L(s) = G(s)\,D(s)$ with relative degree $n_r > 0$ and (by Fact 8.5) $\kappa = \lim_{t\to 0^+} f(t) = \lim_{s\to\infty} s\,L(s)$ as the initial value of the impulse response of $L(s)$; note that κ is finite if $n_r = 1$, and zero if $n_r > 1$. It follows that*

$$\int_0^\infty \ln|S(i\omega)|\, d\omega = -\kappa\,\pi/2.$$

Proof of Bode's Integral Theorem, which involves contour integration, is given in Example B.1 of *NR*. As shown in Figure 10.2, for a given value of κ [if $n_r > 1$, then $\kappa = 0$, independent of $D(s)$], Bode's integral theorem may be understood geometrically as the waterbed effect: if the magnitude of the sensitivity $S(\mathrm{i}\omega)$ is reduced over some frequencies, it is increased over other frequencies in such a way that its integral over all frequencies is $-\kappa\,\pi/2$, independent of how you adjust the controller $D(s)$, thus illustrating another fundamental tradeoff. Attempts to reduce the magnitude of the sensitivity $S(\mathrm{i}\omega) = Y(\mathrm{i}\omega)/V(\mathrm{i}\omega)$ constitute, in a sense, a **zero-sum game**: if this sensitivity is reduced (improved) at some frequencies ω, it will be increased (worsened) at other frequencies. This yet again, in a somewhat different sense, motivates Guideline 10.1: we should attempt to reduce the system's sensitivity to measurement noise $V(\mathrm{i}\omega)$ only at those frequencies for which the effect of this noise is otherwise expected to be pronounced.

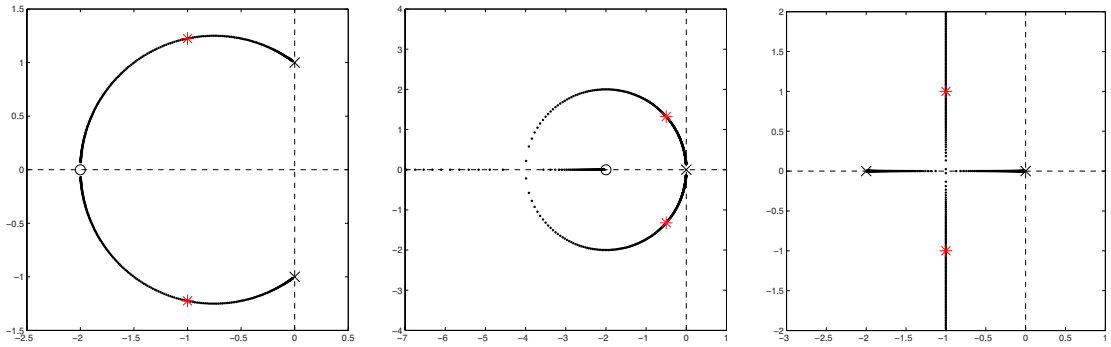


Figure 10.3: (dots) Root loci of three systems with proportional feedback $[D(s) = K]$ applied, with $\times$ marking the open-loop poles, $\circ$ marking the open-loop zeros, and $*$ marking the closed-loop poles for $K = 1$. Systems considered are: (left) $G(s) = (s+2)^2/(s^2+1)$, (center) $G(s) = (s+2)/s^2$, and (right) $G(s) = 2/(s^2+2s)$.

10.2 Primary analysis tools used in classical control design

We now consider the four essential tools of classical feedback control design: the **root locus** (§10.2.1), the **Bode plot** (§10.2.2), the **Nyquist plot** (§10.2.3), and the **closed-loop Bode plot** (§10.2.4); these four tools may be used in a deliberate fashion[2] along with the **step response** (see Example 8.1) for effective CT control design. To introduce these tools, we consider first (in §10.2) the closing of feedback loops around various simple CT plants with a **proportional control** strategy which sets the control input proportional to the error signal [that is, $u(t) = Ke(t)$, and thus $D(s) = K$]. In §10.2.5, we discuss how these tools may be extended to DT systems. In §10.3, we show how these tools may be used to tune more sophisticated control designs which more precisely target the dynamics of interest in the system under consideration.

10.2.1 The root locus with respect to K

Recall from (10.3) that, if $L(s) = G(s)\,D(s) = K\,b(s)/a(s)$, then the poles of the closed-loop transfer function $T(s)$ are given by[3] $a(s) + Kb(s) = 0 \;\Leftrightarrow\; a(s)/K + b(s) = 0$. Thus,

- for small K, the poles of $T(s)$ are near the *poles* of $L(s)$ (i.e., the values of s with $a(s) = 0$),
- for large K, the poles of $T(s)$ are near the *zeros* of $L(s)$ (i.e., the values of s with $b(s) = 0$), and
- for intermediate K, the poles are in-between, moving continuously as K is increased (see §B.2.5).

A plot reflecting the movement of the closed-loop poles of a system as a parameter in the controller (in this case, K) is varied (in this case, from zero to infinity) is known as a **root locus**. A root locus with respect to the gain K is by far the most common type of root locus encountered, and is often referred to as "the" root locus of the system; root loci with respect to other controller parameters are considered in the Exercises.

Root loci with respect to K of some simple plants $G(s)$ with proportional control $D(s) = K$ applied are illustrated in Figures 10.3 and 10.4. If $L(s) = G(s)\,D(s)$ is strictly proper, with n poles and m finite zeros where $n > m$, it may be argued (see Figure 10.4) that the "missing zeros" that $(n-m)$ branches of the locus approach for large K are at the "point at infinity" in the **extended complex plane**, a notion which is most clearly understood when the extended complex plane is mapped onto the **Riemann sphere** (see Figure B.2).

[2] To ensure the controller excites the system as little as possible while meeting the control objectives (see Guideline 10.1), one is highly discouraged from applying the **random control design** (**RCD**) strategies facilitated by root locus **graphical user interfaces** (**GUIs**), which almost encourage one to plunk a controller pole here and zero there until, perhaps accidentally, stability is achieved.

[3] Thus, e.g., if $a(s) = s^2 + 1$ and $b(s) = (s+2)^2$ (see Figure 10.3a), then $(s^2+1) + K(s+2)^2 = 0$, and the closed-loop poles are located at $s = [-4K \pm \sqrt{(4K)^2 - 4(K+1)(4K+1)}]/[2(K+1)]$. For higher-order systems, the locations of the poles as a function of K must generally be found numerically.

Root loci are valuable for understanding parametric dependencies during feedback control design. Though easily plotted using a computer, it is sometimes useful to know how to quickly sketch a root locus with respect to the overall gain K by hand in order to anticipate how the closed-loop poles of a system change when the controller is modified, or (as discussed in §10.3 through §10.4) to understand how to modify a controller to change a root locus in a desired manner, thereby moving the poles of a closed-loop system into the region suggested by the approximate design guides illustrated for CT in Figure 8.4, and for DT in Figure 8.6. Some simple rules for sketching root loci by hand are thus outlined below.

Drawing root loci of CT and DT systems numerically

Like the code for drawing a Bode plot, programming a computer to draw a root locus of a given closed-loop system $T(s)$ or $T(z)$ is trivial, and is easily done in a few lines of code (see): simply loop over several values of K, from small to large, compute the roots of the denominator of $T(s)$ or $T(z)$ for each value of K, then plot these roots in the complex plane. [Note that the code for drawing root loci of CT and DT systems is identical, it is only the goal of where you want the closed-loop poles to wind up that changes.]

Sketching root loci with respect to K by hand

To proceed, assume $L(s) = G(s)\,D(s)$ is a transfer function with m complex zeros z_k and n complex poles p_k, where $m \leq n$ (see §8.2.3.1), such that

$$L(s) = G(s)\,D(s) = K\,\frac{b(s)}{a(s)} = K\,\frac{(s-z_1)(s-z_2)\cdots(s-z_m)}{(s-p_1)(s-p_2)\cdots(s-p_n)}.$$

We also define the multiplicity of the k'th pole as q_k and the multiplicity of the k'th zero as r_k.

The **180° root locus with respect to** K [i.e., noting (10.3), the locus of all points s such that $1 + L(s) = 0$ when $K > 0$] may be sketched using the following handy rules (Evans 1950):

1. Mark the n poles p_i (the roots of $a(s) = 0$) with an $\times$ and the m zeros z_i (the roots of $b(s) = 0$) with an $\circ$.

2. Draw the locus on the real axis to the left of an odd number of real poles plus zeros counted from the right.

3. The branches of the locus **depart** (start) from the open-loop poles and **arrive** (end) at the open-loop zeros. If $n > m$, then $n - m$ branches extend to infinity, approaching $n - m$ asymptotes centered at $\alpha = \frac{\sum p_i - \sum z_i}{n-m}$ and departing at angles $\theta_\ell = \frac{180^\circ + (\ell-1)360^\circ}{n-m}$ for $\ell = 1, \ldots, n-m$ (see Figures 10.4d-f for $n - m = 1$, 2, and 3).

4. For any open-loop pole p_k of multiplicity q_k, or zero z_k of multiplicity r_k, define ψ_i as the angle from the i'th pole to this point, and ϕ_i as the angle from the i'th zero to this point. Then one may calculate the

   a. **departure angles** $\psi_{\text{dep}}^{k,\ell}$ from the open-loop pole p_k as $\psi_{\text{dep}}^{k,\ell} = \frac{\sum \phi_i - \sum_{i \neq k} \psi_i + 180^\circ + (\ell-1)360^\circ}{q_k}$ for $\ell = 1, \ldots, q_k$.

   b. **arrival angles** $\phi_{\text{arr}}^{k,\ell}$ to the open-loop zero z_k as $\phi_{\text{arr}}^{k,\ell} = \frac{\sum_{i \neq k} \phi_i - \sum \psi_i + 180^\circ + (\ell-1)360^\circ}{r_k}$ for $\ell = 1, \ldots, r_k$.

5. In the case that two (resp., three) branches of the locus touch at a point, the angle between the branches of the locus at this junction is 90° (resp., 60°), and the branches into and out of the junction alternate.

The **0° root locus with respect to** K [i.e.,, the locus of all points s such that $1 + L(s) = 0$ with $K < 0$] may be sketched by modifying the above rules as follows:

2. Draw the locus on the real axis to the left of an *even* number of real poles plus zeros counted from the right.

3. The asymptotes depart at angles $\theta_\ell = \frac{(\ell-1)360^\circ}{n-m}$ for $\ell = 1, \ldots, n-m$.

4a. The departure angles from the open-loop pole p_k are $\psi_{\text{dep}}^{k,\ell} = \frac{\sum \phi_i - \sum_{i \neq k} \psi_i + (\ell-1)360^\circ}{q_k}$ for $\ell = 1, \ldots, q_k$.

4b. The arrival angles to the open-loop zero z_k are $\phi_{\text{arr}}^{k,\ell} = \frac{\sum_{i \neq k} \phi_i - \sum \psi_i + (\ell-1)360^\circ}{r_k}$ for $\ell = 1, \ldots, r_k$.

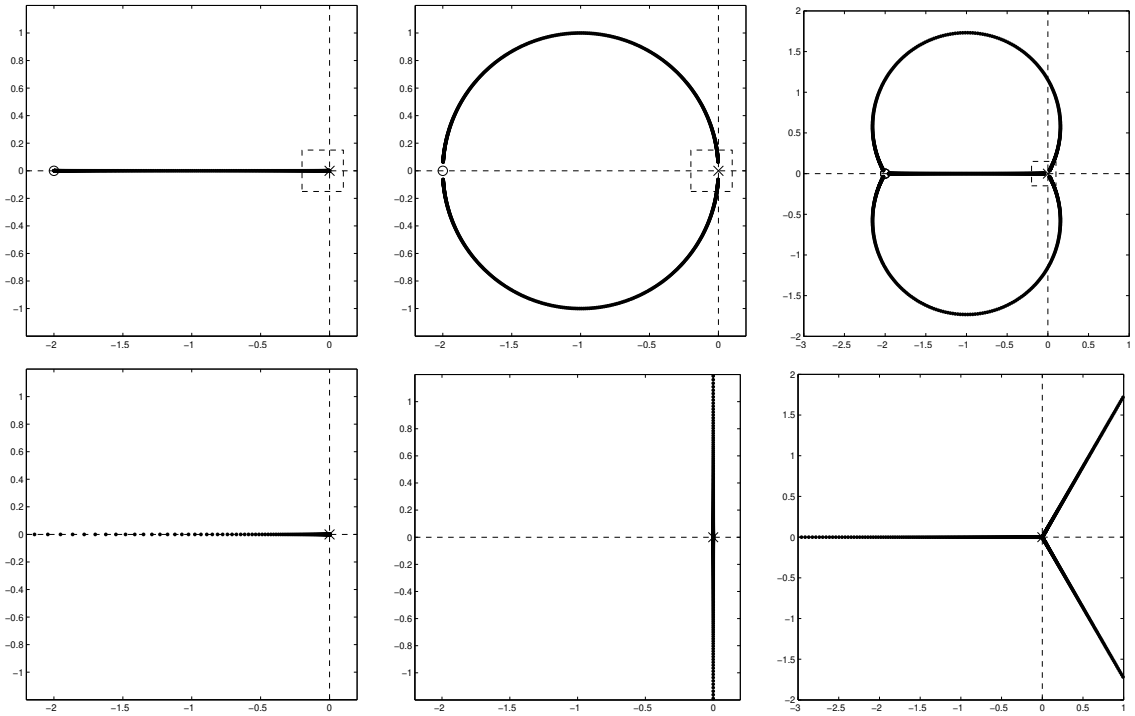


Figure 10.4: Root loci of (left) $L(s) = G(s)D(s) = K(s+c)/(cs)$, (center) $L(s) = K(s+c)^2/(cs)^2$, and (right) $L(s) = K(s+c)^3/(cs)^3$, taking (top) $c = 2$ and (bottom) the limit as $c \to \infty$. As c is increased, the location of the zero(s), towards which the closed-loop pole(s) converge for large K, moves off to the left towards infinity, and the root locus in the vicinity of the origin is dominated by the pattern outlined by the small dashed box. Thus, in the limit as $c \to \infty$, the three root loci approach those of $L(s) = K/s$, $L(s) = K/s^2$, and $L(s) = K/s^3$, respectively. This explanation helps one to visualize that the zeros at infinity are, in a sense, *all at the same place*—a place which one may refer to as the north pole if the s plane is conformally mapped onto a Riemann sphere, with the origin of the s plane mapped to the south pole (see Figure B.2).

Note that, for any K, the closed-loop poles are characterized by $1 + K\, b(s)/a(s) = 0$, and thus

$$K = -a(s)/b(s); \tag{10.6}$$

this useful formula gives the value of K corresponding to any given point s on the locus[4]. Further, in the vicinity of any **breakaway point** of the root locus (i.e., any point where multiple branches of the locus connect to the real axis), the value of K given by (10.6) reaches a local maximum or local minimum with respect to the (real) value of s as you move to the left or right on the axis. Considering $-\infty < K < \infty$, the set of all breakaway points of the 180° *and* 0° root loci on the real axis are given by setting the derivative of (10.6) equal to zero:

$$\frac{dK}{ds} = 0 \quad \Rightarrow \quad -\frac{a'(s)}{b(s)} + \frac{a(s)\, b'(s)}{[b(s)]^2} = 0 \quad \Rightarrow \quad a'(s)\, b(s) - a(s)\, b'(s) = 0; \tag{10.7a}$$

the breakaway points are given by taking the roots of the polynomial at right in (10.7a). Dividing this formula by $[a(s)\, b(s)]$ and taking $b(s) = (s - z_1) \cdots (s - z_m)$ and $a(s) = (s - p_1) \cdots (s - p_n)$ gives the alternative form

$$\textstyle\sum_{i=1}^{n} 1/(s - p_i) - \sum_{i=1}^{m} 1/(s - z_i) = 0. \tag{10.7b}$$

With a little practice, one can usually sketch a root locus quite accurately and quickly by hand by following the above several rules. Note that the connection of the various branches of a root locus sometimes change rather

[4] Note that K must be real; if a point s on the locus is only known approximately, application of (10.6) might result in a complex value of K; taking the real part of this value results in a nearby point on the locus.

rapidly (but smoothly! see §B.2.5) as the open loop poles and zeros are moved, as seen in Figure 10.6. Such **rapid reconnections** of the root locus are somewhat difficult to anticipate when sketching the locus by hand; computation of the breakaway points is often helpful. Note, however, that these reconnections of the locus in fact have little impact on the closed-loop behavior of the system; the two systems depicted in Figure 10.6 have essentially identical step responses for all values of K.

10.2.1.1 The full and simplified Routh and Bistritz tests†

The **Routh test**, developed in §B.2.6, is a procedure for counting how many roots of a polynomial $p(s)$ are in the LHP, on the imaginary axis, and in the RHP [often referred to as the **inertia** of $p(s)$], without requiring the computation of the roots themselves, which can be computationally expensive.

The **Bistritz test**, developed in §B.2.7, is a procedure for counting how many roots of a polynomial $p(z)$ are inside the unit circle, on the unit circle, and outside the unit circle [often referred to as the **stationarity** of $p(z)$], without requiring the computation of the roots themselves.

When applying the Routh or Bistritz test to the polynomial in the denominator of a CT or DT (open-loop or closed-loop) transfer function, to determine whether or not this transfer function is stable, the **simplified Routh test** or **simplified Bistritz test** may be used; these simplified tests are presented in §B.2.8 and §B.2.9. A useful feature of these simplified tests is that one can easily carry one or more *symbolic variables* in a control design formulation, such as the controller gain K, all the way through the analysis, thereby determining necessary and sufficient conditions on these variables for closed-loop stability.

10.2.1.2 Approximate pole-zero cancellations in the LHP

Consider a plant and controller given by

$$G(s) = \frac{1}{s(s+0.95)}, \quad D(s) = 10\,\frac{s+1}{s+4} \quad \Rightarrow \quad T(s) = \frac{G(s)\,D(s)}{1+G(s)\,D(s)} = \frac{10(s+1)}{(s+1.0219)(s-p_+)(s-p_-)}$$

where $p_\pm = -1.9641 \pm 2.4348\,\mathrm{i} \triangleq -\sigma \pm \mathrm{i}\,\omega_d$. Note the controller zero close to the plant pole in the LHP near $s=-1$; as a result, the closed-loop system has both a pole and a zero near $s=-1$. Via partial fraction expansion and inverse Laplace transform (see Example 8.1), the step response of this closed-loop system is

$$Y(s) = T(s)R(s) = \frac{T(s)}{s} = \frac{-1.0314s - 4.0194}{(s+\sigma)^2+\omega_d^2} + \frac{0.0314}{s+1.0219} + \frac{1}{s}$$
$$\Rightarrow \quad y(t) = -1.0314\,e^{-\sigma t}\cos(\omega_d t) - 0.8188\,e^{-\sigma t}\sin(\omega_d t) + 0.0314\,e^{-1.0219\,t} + 1.$$

Note the third term of $y(t)$, which is the result of the closed-loop pole near $s=-1$ arising from the inexact pole/zero cancellation in between the plant and the controller. This contribution to the step response is a decaying exponential with, due to the proximity of this pole and zero, a very small coefficient. This is essentially negligible; had we simply cancelled the plant pole and the nearby controller zero during the analysis,

$$G(s)\,D(s) \approx \frac{10}{s(s+4)} \quad \Rightarrow \quad T(s) \approx \frac{10}{s^2+4s+10} = \frac{10}{(s-\overline{p}_+)(s-\overline{p}_-)}$$

where $\overline{p}_\pm = -2 \pm 2.4495\,\mathrm{i} \triangleq -\overline{\sigma} \pm \mathrm{i}\,\overline{\omega}_d$, the step response computed would have been

$$Y(s) \approx \frac{T(s)}{s} = \frac{-1s-4}{(s+\overline{\sigma})^2+\overline{\omega}_d^2} + \frac{1}{s} \quad \Rightarrow \quad y(t) \approx -1\,e^{-\overline{\sigma}\,t}\cos(\overline{\omega}_d t) - 0.8165\,e^{-\overline{\sigma}\,t}\sin(\overline{\omega}_d t) + 1,$$

which is essentially identical. Note, however, that had the approximate pole/zero cancellation taken place in the RHP, then the step response $y(t)$ would have had a component with a *growing* exponential and a very small coefficient; in addition to this approach failing to provide internal stability [see parts (e) and (f) of Fact 10.1], this exponentially-growing component would eventually dominate the system response. All attempts to cancel a plant pole (or zero) with a controller zero (or pole) must ultimately be considered as approximate to some degree. The conclusion to be drawn is thus consistent with that of Fact 10.1: *approximate pole/zero cancellations arising during controller design may simply be neglected in the LHP, but must never be attempted in the RHP.* The design of a controller to achieve an approximate pole/zero cancellation on or near the imaginary axis, called a **notch filter**, is delicate but doable, and is considered at length in §10.3.2.

10.2.2 The Bode plot, revisited

As seen in §8.4, an (open-loop) Bode plot summarizes the gain and phase of the response of the system $G(s)\,D(s)$, in *open loop*, to sinusoidal inputs $u(t)$. As seen in (10.3), the poles of the *closed-loop* transfer function $T(s)$ are given by the roots of the equation $1 + G(s)\,D(s) = 0$ [i.e., $G(s)\,D(s) = -1$] and, the motion of these roots as a parameter of the controller (usually, its gain) is varied is often plotted in a root locus.

A somewhat subtle connection between the open-loop and closed-loop problems makes the Bode plot especially useful for feedback control design. If for some s on the imaginary axis [that is, $s = \mathrm{i}\omega_r$ for some **resonant frequency** ω_r] the open-loop system $G(s)\,D(s)$ simultaneously has a gain of 1 and a phase of $-180°$ [that is, $G(\mathrm{i}\omega_r)D(\mathrm{i}\omega_r)$ has the critical value of -1], then the closed-loop transfer function $T(s) = G(s)\,D(s)/[1 + G(s)\,D(s)]$ has a pole on the imaginary axis, and is on the verge of instability, meaning that:

- if the system is given an impulse input, it will oscillate at the resonant frequency ω_r without decaying,
- if the system is excited sinusoidally at frequency ω_r, the response will grow without bound, and
- any tiny unmodeled error in either the plant or the controller could lead to closed-loop instability.

We might thus label the imaginary axis in the s-plane as an **axis of evil**; it is imperative to check any stable closed-loop system (that is, with its poles in the LHP) to ensure that its poles are in some sense "far" from being on this axis. Two valuable measures that may be read directly off the Bode plot (see, e.g., Figure 10.7) accomplish exactly this: the **phase margin** (**PM**) quantifies the amount that the phase of the open-loop system $G(\mathrm{i}\omega)D(\mathrm{i}\omega)$ is away from $-180°$ at the frequency ω_g for which the open-loop system gain equals 1, whereas the **gain margin** (**GM**) quantifies the factor by which the gain of the open-loop system $G(\mathrm{i}\omega)D(\mathrm{i}\omega)$ is away from 1 at the frequency ω_p for which the open-loop system phase equals $-180°$. If the PM is large, then there may be correspondingly large errors in the modeling of the phase of the system (due, for example, to unmodeled delays in the system) before risking closed-loop instability, whereas if the GM is large, then there may be correspondingly large errors in the modeling of the gain of the system (due, for example, to uncertainty in the actuator authority or sensor sensitivity) before risking closed-loop instability.

The Bode plot illustrated in Figure 10.7 also depicts the typical constraints considered during the design of the controller $D(s)$. A *large open-loop gain* $|G(i\omega)D(i\omega)|$ *is generally sought at low frequencies* to ensure adequate tracking of the reference input, and a *small open-loop gain is generally sought at high frequencies* to ensure adequate attenuation of high-frequency disturbances[5]. Thus, at some intermediate frequency [dubbed the **(gain) crossover frequency** ω_g], $|G(\mathrm{i}\omega_g)D(\mathrm{i}\omega_g)| = 1$. As in (8.17), the following convenient **approximate design guides** may be identified by examining a range of step responses[6]:

$$\omega_g \approx \omega_n \approx \omega_{BW}/1.4, \quad \zeta \approx \mathrm{PM}/100 \quad \begin{cases} \zeta \gtrsim 0.5 & \text{for} \quad M_p \leq 15\% \\ \zeta \gtrsim 0.7 & \text{for} \quad M_p \leq 5\%. \end{cases} \tag{10.8}$$

[5] I.e., for small ω, $\frac{Y(\mathrm{i}\omega)}{R(\mathrm{i}\omega)} = \frac{G(\mathrm{i}\omega)\,D(\mathrm{i}\omega)}{1+G(\mathrm{i}\omega)D(\mathrm{i}\omega)} \approx 1$ if $|G(\mathrm{i}\omega)D(\mathrm{i}\omega)| \gg 1$; for large ω, $\frac{U(\mathrm{i}\omega)}{W(\mathrm{i}\omega)} = \frac{-G(\mathrm{i}\omega)D(\mathrm{i}\omega)}{1+G(\mathrm{i}\omega)D(\mathrm{i}\omega)} \approx 0$ if $|G(\mathrm{i}\omega)D(\mathrm{i}\omega)| \ll 1$.
[6] For clarity of presentation, the definition of ω_{BW} is deferred to §10.2.4 and Figure 10.9b.

Thus, a target value for the crossover frequency ω_g may be determined from rise time or tracking constraints, and a target value for the PM may be determined from the overshoot constraint. Noting Figure 10.7, when performing controller design using a Bode plot, one typically first tunes the phase of the controller $D(s)$ to achieve the desired PM at the target ω_g, then tunes the overall gain of the controller to achieve gain crossover at this target frequency. Next, if needed, the controller is adjusted at both low and high frequencies to meet the tracking and robustness constraints, in addition to ensuring an adequate GM, and the overall gain retuned to maintain gain crossover at the target ω_g. Finally, the step response of the closed-loop system is checked, and further fine tuning is performed to meet the design constraints (e.g., increasing ω_g to reduce the rise time, increasing the PM to reduce the overshoot, etc.). As discussed in §10.3, this design process can usually be performed via a methodical combination of lead compensation, lag compensation, and low-pass filtering.

As noted in Fact 8.15, a remarkable and useful feature of the Bode plot is that it is *additive*; that is,

$$\log|G(\mathrm{i}\omega)D(\mathrm{i}\omega)| = \log|G(\mathrm{i}\omega)| + \log|D(\mathrm{i}\omega)|, \quad \text{and} \quad \angle G(\mathrm{i}\omega)D(\mathrm{i}\omega) = \angle G(\mathrm{i}\omega) + \angle D(\mathrm{i}\omega). \tag{10.9}$$

Thus, when examining the Bode plot of the plant $G(s)$, it is usually clear what is needed in terms of the gain and phase of the controller $D(s)$ such that the cascade of the controller and plant have the appropriate overall behavior to meet the design guides discussed above (e.g., the rise time and overshoot constraints on the closed-loop system). Control design leveraging the Bode plot of $G(s)D(s)$ like this is referred to as **loop shaping**.

As noted at the end of §8.4, in systems with no RHP zeros or poles, the gain and phase curves are related in a simple fashion: a gain slope of -2 over a particular range of frequencies corresponds to $\sim 1/(\mathrm{i}\omega)^2$ behavior of the transfer function, and thus a phase of about $-180°$; similarly, a gain slope of -1 corresponds to a phase of about $-90°$, and a gain slope of 0 corresponds to a phase of about $0°$. Thus, a rule of thumb for achieving a good PM (and, thus, good damping and low overshoot) in many systems is to *attempt to achieve crossover at a gain slope of approximately* -1; if crossover is attempted at a gain slope of closer to -2, the PM will often be unacceptably small (and, thus, the overshoot will be unacceptably high).

Example 10.1 The primary analysis tools of classical control design are: root loci [§10.2.1; see RR_rlocus], open-loop and closed-loop Bode plots [§10.2.2 and §10.2.4; see RR_bode], Nyquist plots [§10.2.3; see RR_nyquist], and closed-loop step responses [§10.2.4; see RR_step]. These tools, applied to $G(s) = (s+.3)/[s^2(s+2)(s+10)]$ and $D(s) = K$, are illustrated side by side in Figures 10.6-10.9. Use of these tools is discussed in §10.3. △

10.2.3 The Nyquist plot

As illustrated in Figure 10.8, *a Nyquist plot is just an (open-loop) Bode plot drawn in polar coördinates*, and may be generated using RR_nyquist. Defining $L(s) = G(s)\,D(s)$, the Nyquist contour is a curve in the L-plane comprised of points with modulus $|L(s)|$ and phase $\angle L(s)$, drawn for $s = \mathrm{i}\omega$ with the parameter ω varying from $-\infty$ to 0 to ∞. The PM and GM indicated in the Bode plot in Figure 10.7 are both readily identified in the Nyquist plot in Figure 10.8; a third measure, the **vector margin** (**VM**), quantifies, in a sense, the minimum distance over all ω to the critical condition indicating marginal stability of the closed-loop system [$L(\mathrm{i}\omega) = -1$] via a modification of both the gain *and* the phase of the open-loop system $L(s)$.

The Nyquist stability criterion

Though it is easy to discern from a root locus whether or not a closed-loop system is stable [by checking that all of the closed-loop poles are in the LHP], it is sometimes valuable[7] to discern closed-loop system stablility by looking at the corresponding Nyquist plot[8]. A method for doing this follows straightforwardly from Cauchy's argument principle (Fact B.1) a careful review of which is advised before continuing.

[7] Specifically, when the frequency response (that is, the Bode plot) of the open-loop system is available, but a system model is not.
[8] It is difficult to discern stability from a Bode plot though, as shown below, it is straightforward to discern it from a Nyquist plot.

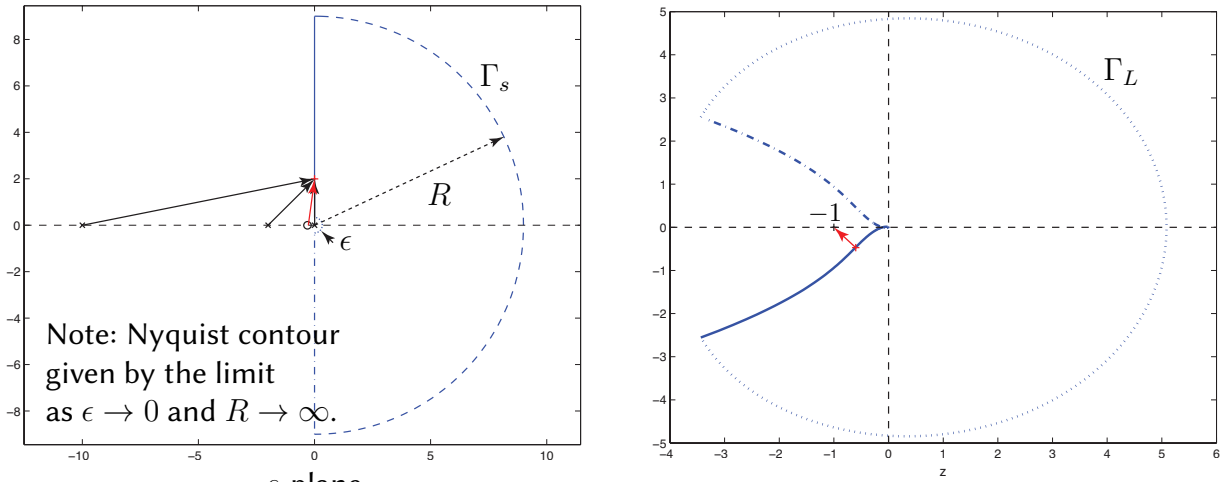


Figure 10.5: (a) Approximating contour Γ_s of the **Nyquist contour** in the s-plane, and (b) the corresponding contour Γ_L in the L-plane [for $L(s) = G(s)\,D(s)$ defined as in Figures 10.6a through 10.9]. The actual Nyquist contour is found by taking the limit as $R \to \infty$ and $\epsilon \to 0$ of these approximating contours.

Fact 10.3 (Nyquist stability criterion) *Define the* **Nyquist contour** Γ_s *as a D-shaped contour of radius R in the RHP of the s-plane, as illustrated in Figures 10.5a and 10.22, where a half circle of radius ϵ is taken into the RHP around every open-loop pole on the imaginary axis, in the limit that $R \to \infty$ and $\epsilon \to 0$. Define also the corresponing contour Γ_L in the L-plane by applying the transform $L(s) = G(s)\,D(s)$ to all points $\overline{s}$ on the contour Γ_s in the s-plane (see, e.g., Figure 10.5b). It follows that, if the number of poles of $L(s)$ with positive real part is P, then the closed-loop system $T(s) = L(s)/[1 + L(s)]$ is stable if and only if the contour Γ_L in the L-plane encicles the $L = -1$ point counterclockwise exactly P times*[9].

Proof: As noted in (10.3), denoting the open-loop system $L(s) = G(s)\,D(s)$, the poles of the closed-loop system $T(s) = L(s)/[1 + L(s)]$ are exactly the zeros of $F(s) = 1 + L(s)$; thus, if $T(s)$ has Z RHP poles, then $F(s)$ will have Z RHP zeros. Now assume that $L(s)$ has P RHP poles; since $F(s) = 1 + L(s)$, $F(s)$ has P RHP poles as well. By design, as illustrated in Figure 10.5a, the Nyquist contour (that is, Γ_s in the limit that $R \to \infty$ and $\epsilon \to 0$) encloses the entire right half plane of s. It thus follows by Cauchy's argument principle (Fact B.1) that the contour Γ_F in the F-plane makes $(P - Z)$ counterclockwise encirclements of the origin, and thus (since $L(s) = F(s) - 1$), Γ_L in the L-plane makes $(P - Z)$ counterclockwise encirclements of the point $L = -1$. Thus, if $Z = 0$ (that is, if $T(s)$ is stable), then the contour Γ_L in the L-plane makes exactly P counterclockwise encirclements of the point $L = -1$; if $Z > 0$ (that is, if $T(s)$ is unstable), then the contour Γ_L in the L-plane makes less than P counterclockwise encirclements of the point $L = -1$. □

Note that the Nyquist contour Γ_L and associated stability criterion (Fact 10.3) depend directly on the expression $L(s) = G(s)\,D(s)$; this is particularly convenient, because a rational factored form of $L(s)$, as well as its Bode plot, follow immediately from those of $G(s)$ and $D(s)$. In contrast, a formulation in terms of $T(s)$ (like the root locus) or in terms of $F(s)$, is less convenient, because a rational factored form for these expressions is not as easy to determine by hand. Note also that the contour Γ_s must take a curve of radius ϵ into the RHP to avoid every pole of the open-loop system $L(s)$ that happens to lie on the imaginary axis, as indicated in the vicinity of the origin for the case depicted in Figure 10.5.

[9] In the case of Figure 10.5, with a stable $L(s)$ with $P = 0$ unstable open-loop poles, one might say that the closed-loop system is stable if the **Pac-Man** does not engulf the dot at $L = -1$, thus implying $Z = 0$ unstable closed-loop poles; in contrast, in the case of Figure 10.10, with an unstable $L(s)$ with $P = 1$ unstable open-loop poles, $P - Z = 1$ counterclockwise encirclements of $L = -1$ is required for stability, thus implying $Z = 0$ unstable closed-loop poles. Note also that, mapping onto their respective Riemann spheres (see Figure B.2), a contour near the south (north) pole in the s plane maps to a contour near the north (south) pole in the L plane.

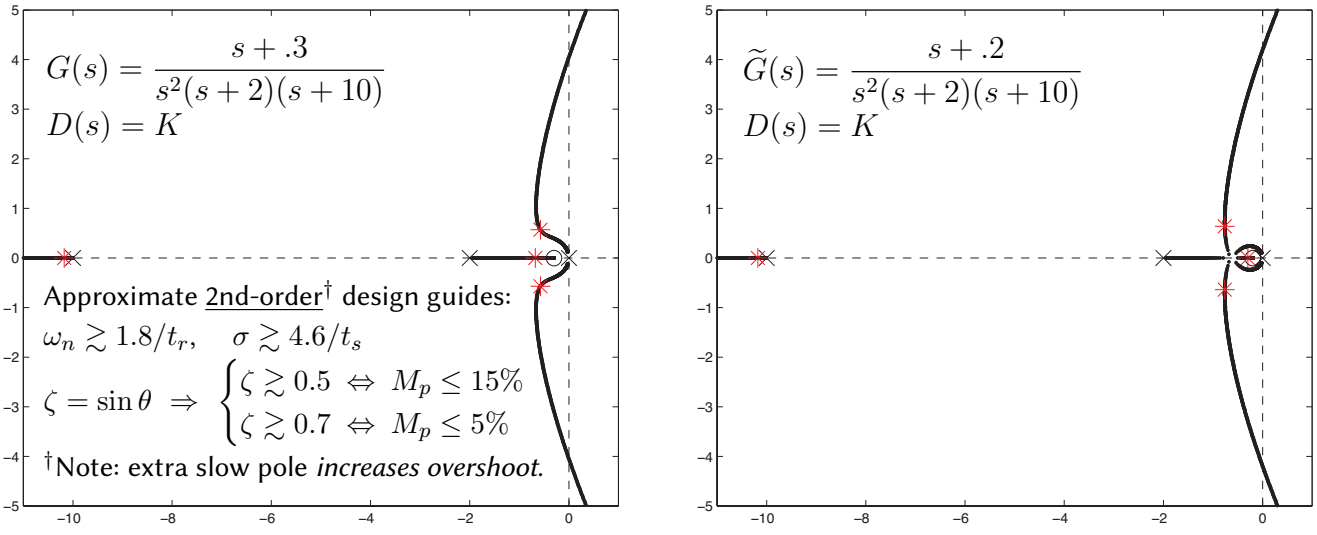


Figure 10.6: Root loci of (left) $G(s)$ and (right) the slightly modified $\widetilde{G}(s)$, with proportional control $D(s) = K$ applied, and with $*$ marking the four closed-loop poles for $K = 15$. Though the locus makes a rather sudden reconnection, the step response of each of these systems is quite similar (i.e., the locus reconnection is inconsequential). If a system is dominated by second-order behavior, the closed-loop pole locations should generally lie in the region suggested by the approximate design guides specified by the rise time, settling time, and/or overshoot constraints (see Figure 8.4) on the closed-loop system. Note that the present closed-loop system has a pair of complex poles, plus two stable poles on the negative real axis. The pole near $s = -10$ is stable and (comparatively) fast, and thus has little effect on the closed-loop dynamics. The other real pole is of about the same speed as the dominant pair of complex poles; this generally results in significant extra overshoot of the step response, thus motivating increased damping on the complex poles than otherwise indicated by the approximate design guides (developed for second-order systems) to compensate appropriately.

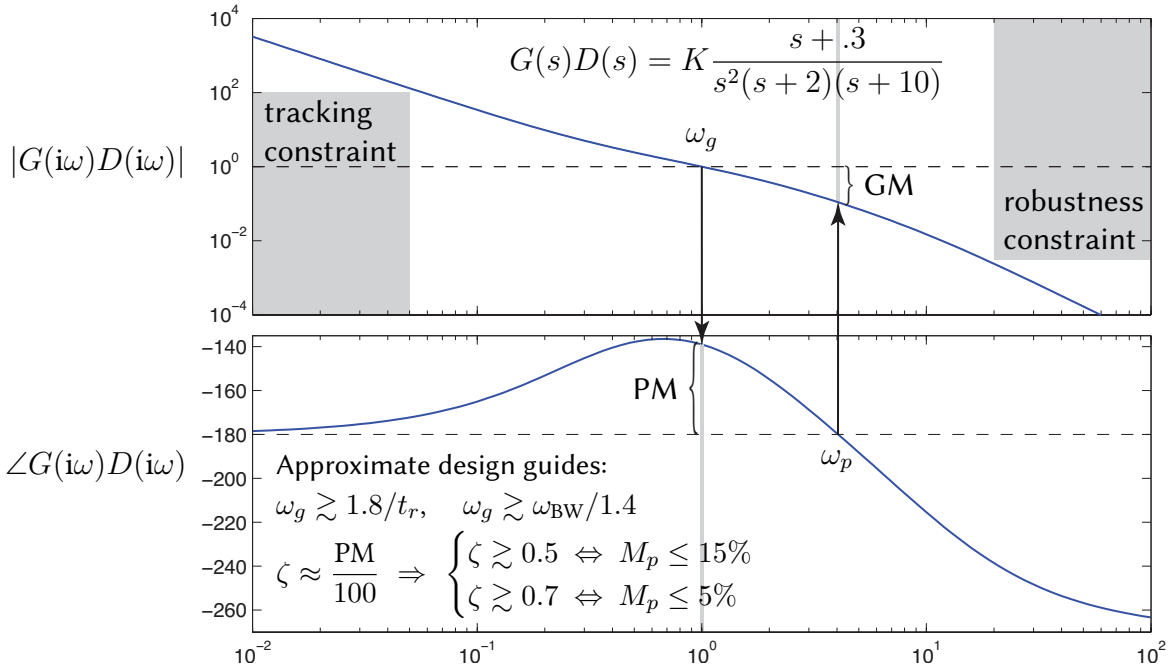


Figure 10.7: Open-loop Bode plot of $G(s)\,D(s)$, with $G(s)$ as in Figure 10.6a and $D(s) = K$, with K adjusted to give crossover at $\omega_g = 1$. The PM is $180^\circ + \angle G(i\omega_g)D(i\omega_g)$, whereas the GM is $1/|G(i\omega_p)D(i\omega_p)|$, where ω_p is defined as that frequency where $\angle G(i\omega_p)D(i\omega_p) = -180^\circ$ (if it exists; otherwise, GM$= \infty$).

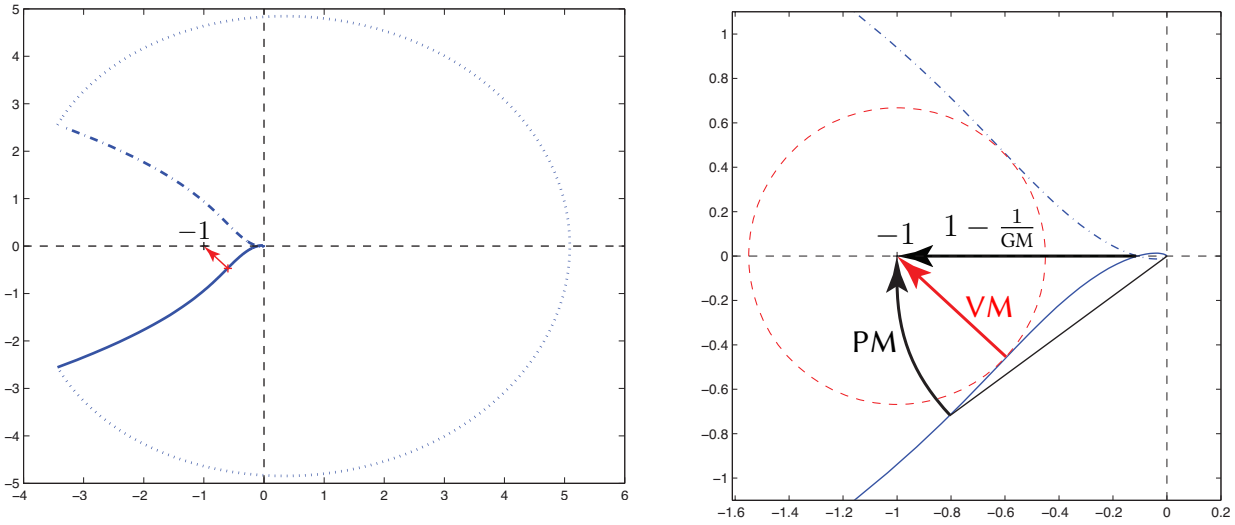


Figure 10.8: Nyquist plot of system considered in Figures 10.6a and 10.7, (a) big-picture view, with a representation of the outer contour, and (b) close up near the $G(\mathrm{i}\omega)D(\mathrm{i}\omega) = -1$ point. The GM and PM marked in Figure 10.7 also appear in this Nyquist plot; another measure, the VM, quantifies the distance of the point on the Nyquist contour closest (via a change in both gain *and* phase) to the critical $G(\mathrm{i}\omega)D(\mathrm{i}\omega) = -1$ condition.

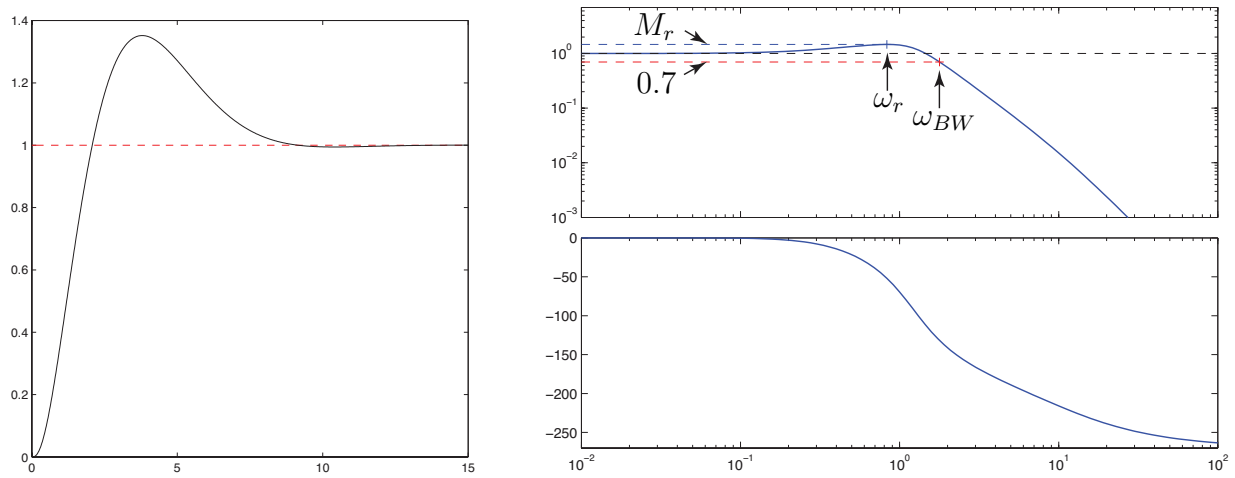


Figure 10.9: Final checks: (left) step response and (right) Bode plot of *closed-loop* system $T(s) = L(s)/[1+L(s)]$ for $L(s) = G(s)D(s)$ considered in Figures 10.6a, 10.7, & 10.8. The rise time, settling time, and overshoot of the step response are defined as in Figure 8.3; the rise time and overshoot are approximately related to the values of ω_g and PM in the corresponding Bode plot (Figure 10.7). The closed-loop Bode plot illustrates good tracking at low frequencies ($|T(\mathrm{i}\omega)| \approx 1$ and $\angle T(\mathrm{i}\omega) \approx 0$) and good disturbance rejection at high frequencies ($|T(\mathrm{i}\omega)| \ll 1$). Peaks of the gain curve of the closed-loop system, if present, occur at the **resonant frequencies** $\omega_{r,i}$, at which the gain is given by the **resonant peaks** $M_{r,i}$. The frequency at which the closed-loop gain falls below 0.7, and thus the output ceases to **track** the reference input faithfully, is the **bandwidth**, ω_{BW}.

Guideline 10.2 *The figures on pages 10-12 and 10-13 typify how classical tools are used together for targeted (see Guideline 10.1) feedback control design: a stabilizing controller is first found with the aid of a root locus (Figure 10.6) if $G(s)$ is known, or with the aid of a Nyquist plot (Figure 10.8) if $G(s)$ is unknown. The controller $D(s) = D_{lead}(s) \cdot D_{lag}(s) \cdot D_{LPF}(s)$ [see §10.3.2] is then tuned as a function of frequency (a design process known as* **loop shaping***) using a Bode plot (Figure 10.7). Finally, the closed-loop performance is checked by examining the rise time, settling time, and overshoot of the step response, and the tracking accuracy, resonant peaks, and bandwidth in the closed-loop Bode plot (Figure 10.9). Such a design process is usually iterative.*

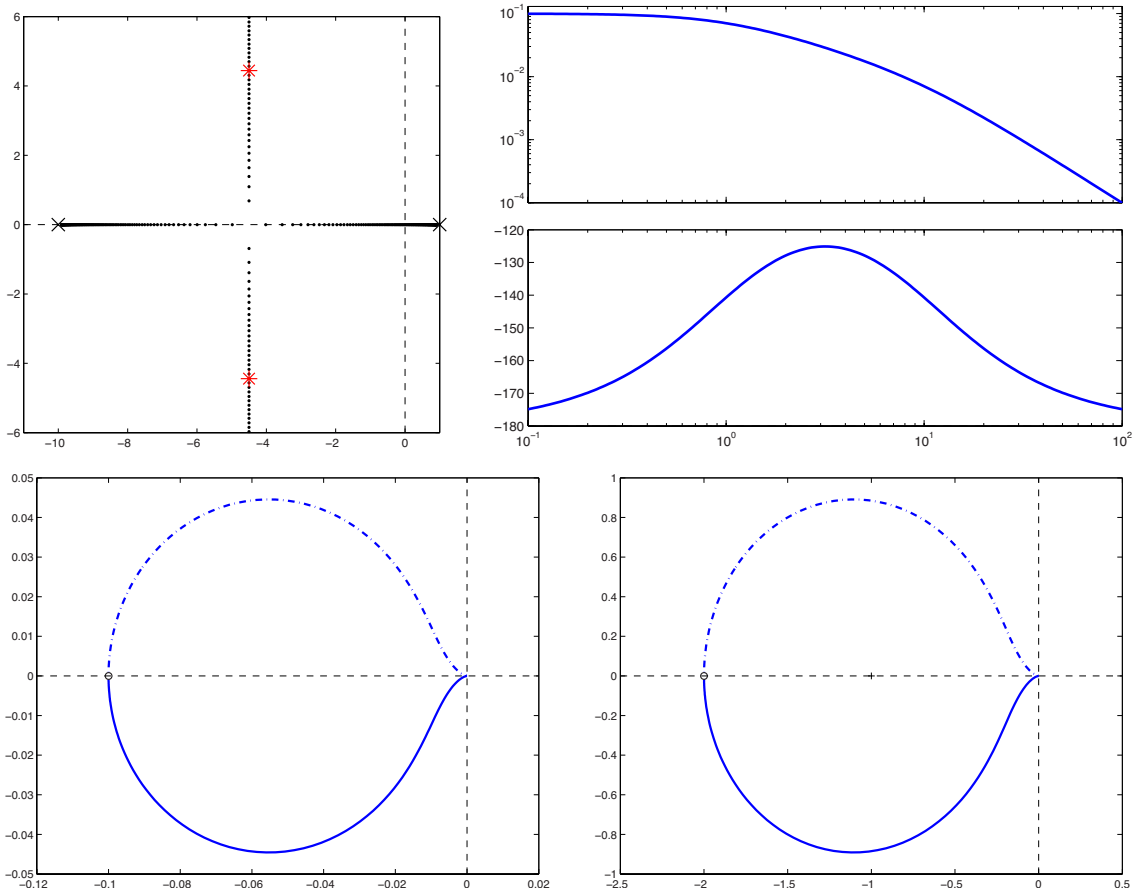


Figure 10.10: Analysis for $L(s) = G(s)D(s) = K/[(s-1)(s+10)]$, with $P = 1$ open-loop pole: (a) Root locus of $L(s)$, with closed-loop poles for $K = 20$ marked, (b) Bode plot of $L(s)$, (c) Nyquist plot of $L(s)$ for $K = 1$ (with $P - Z = 0$ encirclements of $L = -1$, and thus $Z = 1$, and an unstable closed loop), (d) Nyquist plot of $L(s)$ for $K = 20$ (with $P - Z = 1$ encirclement of $L = -1$, and thus $Z = 0$, and a stable closed loop).

By the Nyquist stability criterion, Figure 10.5b indicates *stability* of the closed-loop system for the controller gain K used, as the open-loop system $L(s) = G(s)\,D(s)$ has $P = 0$ RHP poles, and the Nyquist contour does not encircle $L = -1$. A detail view illustrating this Nyquist contour near the origin is given in Figure 10.8; note that if the gain is increased by a factor larger than the GM, then the Nyquist contour would encircle the $L = -1$ point, in which case the Nyquist stability criterion would indicate *instability* of the closed-loop system. This is consistent with the corresponding root locus in Figure 10.6a, which indicates closed-loop *stability* for small gain, and closed-loop *instability* (2 poles in the RHP) for sufficiently large gain.

A case in which the open-loop system $L(s) = G(s)\,D(s) = K/[(s-1)(s+10)]$ has $P = 1$ RHP pole is indicated in Figure 10.10. Note that, in this case, the root locus (Figure 10.10a) indicates closed-loop *instability* (1 pole in the RHP) for small gain and closed-loop *stability* for sufficiently large gain. Transforming the Bode plot of $L(s)$ (see Figure 10.10b) into polar coördinates to get the Nyquist plot for $K = 1$ (see Figure 10.10c), it is seen that the Nyquist contour in this case does *not* encircle the $L = -1$ point; that is, by the Nyquist stability criterion, $P - Z = 0$, and thus $Z = 1$, which indicates *instability* of the closed-loop system. On the other hand, plotting the Nyquist plot for $K = 20$ (see Figure 10.10d), it is seen that the Nyquist contour in this case encircles the $L = -1$ point exactly $P - Z = 1$ time, which by the Nyquist stability criterion indicates $Z = 0$ closed-loop RHP poles, and thus *stability* of the closed-loop system. Thus, the conclusions drawn from the root locus and the Nyquist stability criterion are again consistent.

type	impulse	step	ramp	parabolic	cubic
$r=0$	0	finite	∞	∞	∞
$r=1$	0	0	finite	∞	∞
$r=2$	0	0	0	finite	∞
$r=3$	0	0	0	0	finite

Table 10.1: Steady-state error, $\lim_{t\to\infty} e(t)$, of a closed-loop system as a function of the type r, where $G(s)\,D(s) = b(s)/[s^r a_0(s)]$ with $[a_0(s)]_{s=0} \neq 0$ and $[b(s)]_{s=0} \neq 0$.

10.2.4 Final checks: the closed-loop step response and the closed-loop Bode plot

Once a controller is designed using the classical (that is, root-locus or Nyquist, and Bode) control design tools, final checks on its behavior may be performed by plotting the closed-loop system's **step response** (see Figure 10.9a, generated using RR_step) and by plotting the **closed-loop Bode plot** (see Figure 10.9b, generated using RR_bode), both applied to $T(s) = G(s)\,D(s)/[1+G(s)\,D(s)]$. The former indicates directly if the rise time, settling time, and overshoot constraints on the closed-loop system were indeed met, whereas the latter reveals the frequencies at which the closed-loop system accurately tracks the reference input (as well as the precision of this tracking), the peak magnitude of any **resonances** (that is, the possibly amplified response of the closed-loop system at certain frequencies), and the **bandwidth frequency** ω_{BW} above which the gain of the closed-loop system **rolls off** and the system output no longer tracks the reference input. Based on these final checks, some final tweaking of the control design is often required.

10.2.4.1 System type and loop prefactors

Returning to Figure 10.1a, note that we may write

$$E(s) = R(s) - Y(s) = R(s) - G(s)\,D(s)E(s) \quad \Rightarrow \quad \frac{E(s)}{R(s)} = \frac{1}{1+G(s)\,D(s)}.$$

Now assume that the controller $D(s)$ is chosen, based on the plant $G(s)$, such that **the closed-loop system is stable**, and that we can write $G(s)\,D(s) = b(s)/[s^r a_0(s)]$, where $[a_0(s)]_{s=0} \neq 0$ and $[b(s)]_{s=0} \neq 0$, for some value of r (referred to as the **type** of the open-loop system). We may thus write

$$E(s) = \frac{s^r a_0(s)}{s^r a_0(s) + b(s)}\,R(s). \tag{10.10}$$

Noting the CT final value theorem (Fact 8.4), we may write the **steady state error** $\lim_{t\to\infty} e(t) = \lim_{s\to 0} sE(s)$. Considering impulse, step, ramp, parabolic, and cubic reference inputs[10], we may easily compute the behavior of the steady-state error as a function of type using (10.10), as listed in Table 10.1.

Focusing specifically on the case with unit step input (i.e., for $R(s) = 1/s$), you might notice in certain situations that, though you have designed a stabilizing controller $D(s)$ for a given plant $G(s)$, the step response $y(t)$ of the closed-loop system $T(s)$ does not approach unity as $t \to \infty$. By the above paragraph, if the system $G(s)$ is not accurately known, the change required to fix this problem is to build a sufficient number of integrators into $D(s)$ to make the open-loop system $G(s)\,D(s)$ type 1 [that is, $|G(\mathrm{i}\omega)D(\mathrm{i}\omega)| \to \infty$ as $\omega \to 0$ in the Bode plot] and thus, by the CT final value theorem,

$$\lim_{t\to\infty} y(t) = \lim_{s\to 0} sY(s) = \lim_{s\to 0} sT(s)R(s) = \lim_{s\to 0} T(s) = \lim_{s\to 0} \frac{G(s)\,D(s)}{1+G(s)\,D(s)} = 1,$$

regardless of any uncertainty in the overall gain or phase of $G(s)$.

[10] That is, $r(t) = \delta^{\lambda,m}(t)$, $h_1(t)$, $t\,h_1(t)$, $t^2\,h_1(t)$, and $t^3\,h_1(t)$, with Laplace transforms $R(s) = 1, 1/s, 1/s^2, 2/s^3$, and $6/s^4$.

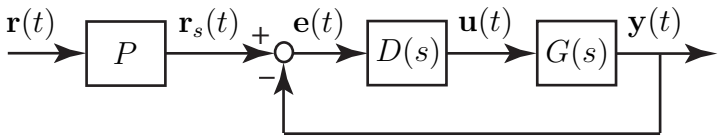


Figure 10.11: Introduction of a loop prefactor P [see (10.11)] to correct for a closed-loop with nonzero steady-state error in its unit step response [due to $G(\mathrm{i}\omega)\,D(\mathrm{i}\omega)$ being finite as $\omega \to 0$].

If, on the other hand, $|G(\mathrm{i}\omega)D(\mathrm{i}\omega)|$ is *finite* as $\omega \to 0$ and (importantly) the values of $G(s)$ and $D(s)$ are known to be relatively accurate at low frequencies, then simply selecting a prefactor

$$P = \left.\frac{1}{T(s)}\right|_{s=0} = \left.\frac{1+G(s)\,D(s)}{G(s)\,D(s)}\right|_{s=0} \tag{10.11}$$

and incorporating as in Figure 10.11 fixes the problem, bringing the step response back to $y(t) \to 1$ as $t \to \infty$.

10.2.5 Extending the Bode, Nyquist, and root locus tools to DT systems

As suggested by (8.27), when plotting frequency response (i.e., a Bode or Nyquist plot) in discrete time, one simply uses $z = e^{\mathrm{i}\omega h}$ in lieu of $s = \mathrm{i}\omega$; RR_bode and RR_nyquist thus easily generate DT Bode and Nyquist plots when called appropriately. Examples of both are given by Figures 10.12a-b.

Further, as rational functions of s and z combine via identical rules when closing a feedback loop, a root locus in z for a DT system may be drawn with *exactly* the same code (e.g., RR_rlocus) as is used to draw a root locus in s for a CT system. It is only the target region, specified by the approximate design guides, that changes (recall the mapping of the curves in Figure 8.4 into the z-plane, as indicated in Figure 8.6b).

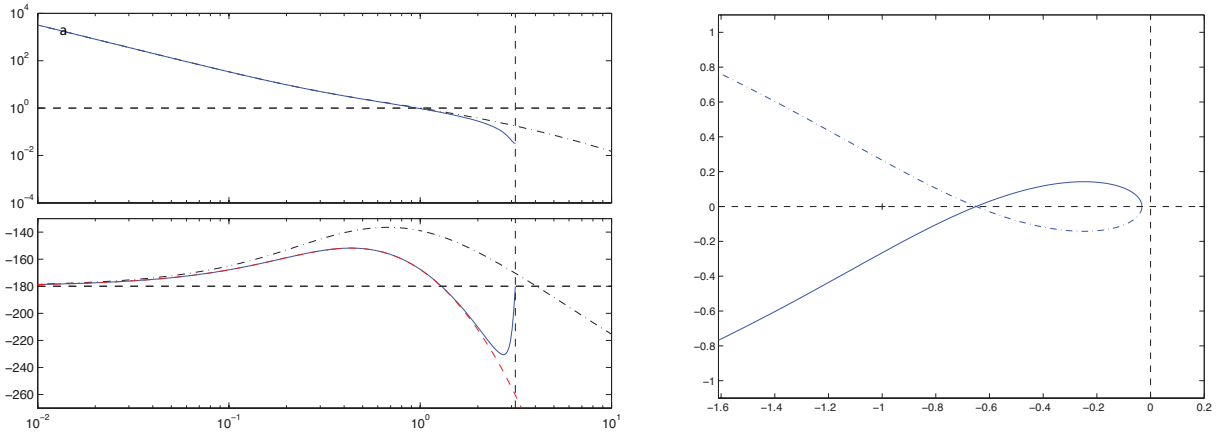


Figure 10.12: (a) Bode and (b) Nyquist plots of (solid) the DT system $G(z)$ given (see §8.3.3.1) by the cascade of a DAC with a ZOH, the system $G(s) = (s+.3)/[s^2(s+2)(s+10)]$ considered in Figures 10.6a through 10.5, and an ADC, with $h = 1$. The highest frequency that can be represented uniquely in discrete time (see Figure 8.6) is the Nyquist frequency $\omega_{\text{Nyquist}} = \pi/h$, indicated by the vertical dashed line in the Bode plot. The Bode plot of the corresponding CT system $G(s)$, (dot-dashed), is shown for comparison. The effect of an $h/2$ time delay on the phase of this CT Bode plot [that is, an $h\,\omega/2$ phase delay at any frequency ω; see (10.31)] is also illustrated (dashed), and accounts for the bulk of the discrepancy between the Bode plots of $G(s)$ and $G(z)$, as explained by Figure 10.33; this phase loss gradually (over an order of magnitude in ω) approaches $h\,\omega_{\text{Nyquist}}/2 = \pi/2$ as $\omega \to \omega_{\text{Nyquist}}$, which is substantial (and, in closed loop, potentially detrimental!). Note also that the DT Nyquist plot does not quite reach the origin in the L plane; the closest point to the origin is achieved for $\omega = \omega_{\text{Nyquist}}$.

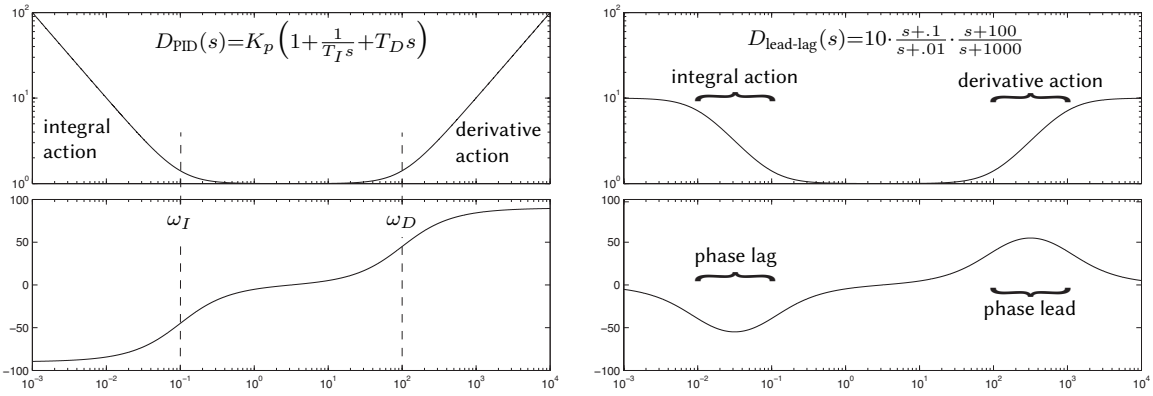


Figure 10.13: (a) Bode plot of the PID controller $D_{\text{PID}}(s)$ given in (10.12) with $K_p = 1$, $T_I = 10$ ($\omega_I = 0.1$), and $T_D = 0.01$ ($\omega_D = 100$). (b) Bode plot of (in application) a nearly-equivalent lead-lag controller $D_{\text{lead-lag}}(s)$, with "roll-off" applied at low and high frequencies of the PID, thus applying the integral and derivative actions each over only finite ranges of frequencies; this roll-off, which is built in (at prespecified frequencies) in commercial "black-box" PID controllers, can be adjusted precisely using the lead/lag techniques presented in §10.3.2.

10.3 Primary techniques used for classical control design

As exemplified in Figures 10.6–10.9, the process of **classical** (i.e., transform-based) linear control design includes

- identifying an appropriate family of stabilizing controllers using root locus or Nyquist plotting tools,
- tuning the control design via the (open-loop) Bode plot, a process known as **loop shaping**, and
- checking the resulting closed-loop performance via the step response and closed-loop Bode plot,

noting Guideline 10.1. To accomplish this, as illustrated below, there are a number of prototype transfer functions (e.g., lead, lag, notch, and low-pass filters) that may be effectively cascaded and tuned.

Regarding the second point above, it is seen that, to be maximally effective in classical control design, you must think in frequency [that is, in terms of the frequency response reflected in the Bode plot of $L(s)$], not simply in terms of the closed-loop step response or pole locations.

10.3.1 PID (Proportional-Integral-Derivative) controllers

The simple **PID** (**Proportional-Integral-Derivative**) controller is by far the most common controller implemented in industry. It is the ultimate "black box" controller, and is characterized by three simple "knobs":

- a constant of proportionality K_p (the gain of the controller at intermediate frequencies),
- a time constant T_I for the integral term (below which the controller gain rises $\propto 1/\omega$ as $\omega \to 0$), and
- a time constant T_D for the derivative term (above which the controller gain rises $\propto \omega$ as $\omega \to \infty$).

The "ideal" PID controller may be written in transfer function form, for finite T_I and T_D, as

$$D_{\text{PID}}(s) = K_p\Big(1 + \frac{1}{T_I s} + T_D s\Big) = K_p T_D \frac{s^2 + s/T_D + 1/(T_I T_D)}{s} = K\frac{(s+z_+)(s+z_-)}{s}, \tag{10.12}$$

where, usually, $T_I > T_D$. Note that $K = K_p T_D$ and, if $T_I \gg T_D$, it follows that

$$z_\pm = \big[1 \pm \sqrt{1 - 4T_D/T_I}\big]/(2T_D) \approx \big[1 \pm (1 - 2T_D/T_I)\big]/(2T_D) = \big\{1/T_I,\ 1/T_D\big\}.$$

The Bode plot of an ideal PID controller $D_{\text{PID}}(s)$ is given in Figure 10.13a.

The reader is encouraged to understand the impact of PID control on the closed-loop system of interest by examining the effect of the three knobs $\{K_p, T_I, T_D\}$ on the corresponding Bode plot, to construct a controller of

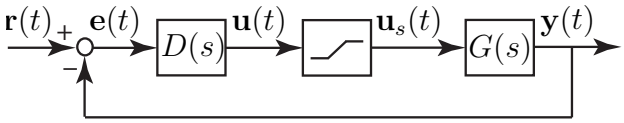


Figure 10.14: Schematic representation of the saturation nonlinearity caused by an actuator being driven to its limits. In this simple representation, the actuator response is modeled to be linear between the limits, but the control signal $u(t)$ is clipped to the min or max values outside these limits. Actuator saturation of this sort can cause integrator windup when PID control is applied, with potentially harmful consequences.

this class that just meets the performance specification on the closed-loop system (see Guideline 10.1). Noting (10.12), taking $T_I \to \infty$ and $T_D \to 0$ reduces the PID controller to the proportional (P) controller considered throughout §10.2; taking $T_I \to \infty$ or $T_D \to 0$ alone reduces PID to PD or PI, respectively. Now recall the typical Bode design constraints on $G(\mathrm{i}\omega)D(\mathrm{i}\omega)$ listed in Figure 10.7, and examine the Bode plot of $D_{\text{PID}}(s)$ in Figure 10.13a. Introducing the derivative term, by dialing T_D up from zero so that $\omega_D = 1/T_D$ is near ω_g, bumps up the phase at crossover, thereby improving the PM and reducing the overshoot of the closed-loop system. Introducing the integral term, by dialing T_I down from ∞ so that $\omega_I = 1/T_I$ is up to an order of magnitude below ω_g, bumps up the low-frequency gain of the open-loop system without diminishing substantially the phase at ω_g, thus improving low-frequency tracking.

Note from (10.12) that the PID controller is governed by the differential equation

$$\frac{U(s)}{E(s)} = K_p\Big(1 + \frac{1}{T_I s} + T_D s\Big) \quad \Leftrightarrow \quad u(t) = K_p\Big(e(t) + \frac{1}{T_I}\int_0^T e(t)\,dt + T_D\frac{de(t)}{dt}\Big). \tag{10.13}$$

Given the prevalence of PID control in industry, it is important to identify the two primary and potentially catastrophic effects that simple PID control of this form can introduce in practice.

The first problem, associated with the derivative term of the PID, is its inherent **amplification of high-frequency noise**. Note on page 10-2 that the control sensitivity is $U(s)/V(s) = D(s)/[1 + G(s)\,D(s)]$. Even if the plant is characterized by a $G(s) \propto 1/s^2$ dependence at high frequencies (many are), a controller with a $D(s) \propto s$ dependence is problematical if the noise $v(t)$ has substantial high frequency components (it usually does), as such a controller amplifies this noise and sends this amplified high frequency garbage in the control signal $u(t)$ to the actuators, which then waste energy doing unnecessary work, or simply burn out.

The second problem, associated with the integral term of the PID, is **integrator windup** in the presence of **actuator saturation** (see Figure 10.14). Linear control theory is often found to be effective even on systems which are only "mostly linear". A nonlinearity that often arises when applying control to physical systems is actuator saturation; that is, actuators used to apply a desired control input $u(t)$ to a physical system typically provide an actual control input $u_s(t)$ to the physical system (a force, torque, displacement, velocity, etc.) that varies between two bounds. In such a situation, proportional controllers usually suffer only a slight performance loss, exhibiting, effectively, reduced values of $K = u_s(t)/e(t)$ when the actuator is saturated. However, a saturated controller with an integrator accumulating a (potentially, nonzero) $e(t)$ causes $u(t)$ to grow linearly (over, potentially, a long period of time), while the saturated value of $u_s(t)$ applied to the plant remains at its bound. This is generally not a problem until the controller needs to again reduce the control input applied to the plant. With $u(t)$ possibly driven to very high values (that is, "wound up"), it can possibly take a correspondingly very long time for $u(t)$ to decrease to the point that the actual control applied to the plant, $u_s(t)$, finally decreases. This delayed responsiveness of the actuation can easily lead to closed-loop instability.

Considered in terms of the frequency response of the PID controller (see Figure 10.13a), the first problem is associated with the derivative part growing without bound as ω is increased, whereas the second issue is associated with the integral part growing without bound as ω is decreased. The cure to both is to **roll off**

the magnitude of the controller response at both ends of the spectrum, as illustrated in Figure 10.13b. In fact, *black box PID controllers implement such roll-off at both ends of the spectrum in order to alleviate the two issues discussed above*; however, they don't give the control designer the ability to specify the break points at which such roll-off sets in, or the degree of roll-off applied. The lead and lag controllers discussed in §10.3.2, together with the low-pass filtering discussed in §10.3.3, provide precisely this capability. That is:

Guideline 10.3 *Lead and lag controllers are the responsible way to apply derivative and integral control actions, respectively, over finite ranges of frequencies (see, e.g., Figure 10.13b), thus adhering to Guideline 10.1.*

There is thus actually no compelling reason to use the restrictive PID control paradigm once the coordinated use of lead, lag, low-pass components is well understood. However, due to their prevalence in industry, it is perhaps prudent to discuss the tuning of PID controllers a bit further below in order to familiarize the reader.

Ad hoc PID tuning

PID controllers may be tuned to be effective on a variety of simple plants. Despite the two problems mentioned above, they are so simple and intuitive that many controls engineers hesitate to use anything else. PID controllers are sometimes tuned to achieve essentially the highest rise time possible given the parameters of the plant and the form of the PID controller while respecting the design constraints on the overshoot and tracking. While there is not a unique way of acheiving this trade-off, one commonly strategy is to first apply proportional feedback $D(s) = K$ to the system $G(s)$ of interest and dial up the gain until a critical value $K = K_u$ is reached at which the system oscillates at nearly constant amplitude, with a frequency which we denote $\omega_u = 1/T_u$. Knowledge of K_u and T_u is, in many cases, enough to tune an effective PID control strategy, which may be accomplished by setting $K_p = \alpha\, K_u$, $T_I = \beta\, T_u$, and $T_D = \gamma\, T_u$, for appropriate values of the parameters $\{\alpha, \beta, \gamma\}$, various values for which have been suggested in the literature; some of the most popular are: **Ziegler-Nichols P**: $\{0.5, \infty, 0\}$, **Ziegler-Nichols PI**: $\{0.45, 0.83, 0\}$, **Ziegler-Nichols PID**: $\{0.6, 0.5, 0.125\}$, **Pessen PID**: $\{0.7, 0.4, 0.15\}$, **Tyreus-Luyben PI**: $\{0.31, 2.2, 0\}$, **Tyreus-Luyben PID**: $\{0.45, 2.2, 0.16\}$. These suggestions are nothing more than rules of thumb that were found to be effective on the problems of interest by the authors proposing them. An example of the effect they have in application is discussed below.

Example 10.2 Control of a first-order system with a delay (cruise control of an automobile) with PID

As derived in Example **??**, the linearized equations of motion of an automobile at cruise may be written

$$\left(\frac{d}{dt} + a\right) v'(t) = C\, u'(t - d); \tag{10.14}$$

note that the model accounts for a slight delay d between the actuation of the throttle and its effect on the force applied to accelerate the vehicle. Noting (8.8), we may approximate the transfer function of the vehicle as

$$\frac{V'(s)}{U'(s)} = G(s) = C\,\frac{e^{-ds}}{s+a} \approx \frac{C}{s+a}\cdot\frac{1-(ds)/2+(ds)^2/12}{1+(ds)/2+(ds)^2/12}. \tag{10.15}$$

For the vehicle considered in Example **??** cruising at $\overline{v} = 34$ m/s $= 76$ mph, the constants in this transfer function are $C = 6.58 \times 10^{-4}$, $a = 39.3$, and $d = 0.04$; we use these values below.

The root locus of this system with proportional feedback applied is given in Figure 10.15a. Due to the delay[11], proportional feedback drives the system unstable at some critical gain $K = K_u$; the ad hoc tuning strategies

[11] Any Padé approximation of a delay [see (8.8)] has RHP poles; it is thus seen that *any system with a delay will be destabilized by high gain feedback*, as in Figure 10.15a. System delays often go unmodeled, again motivating Guideline 10.1.

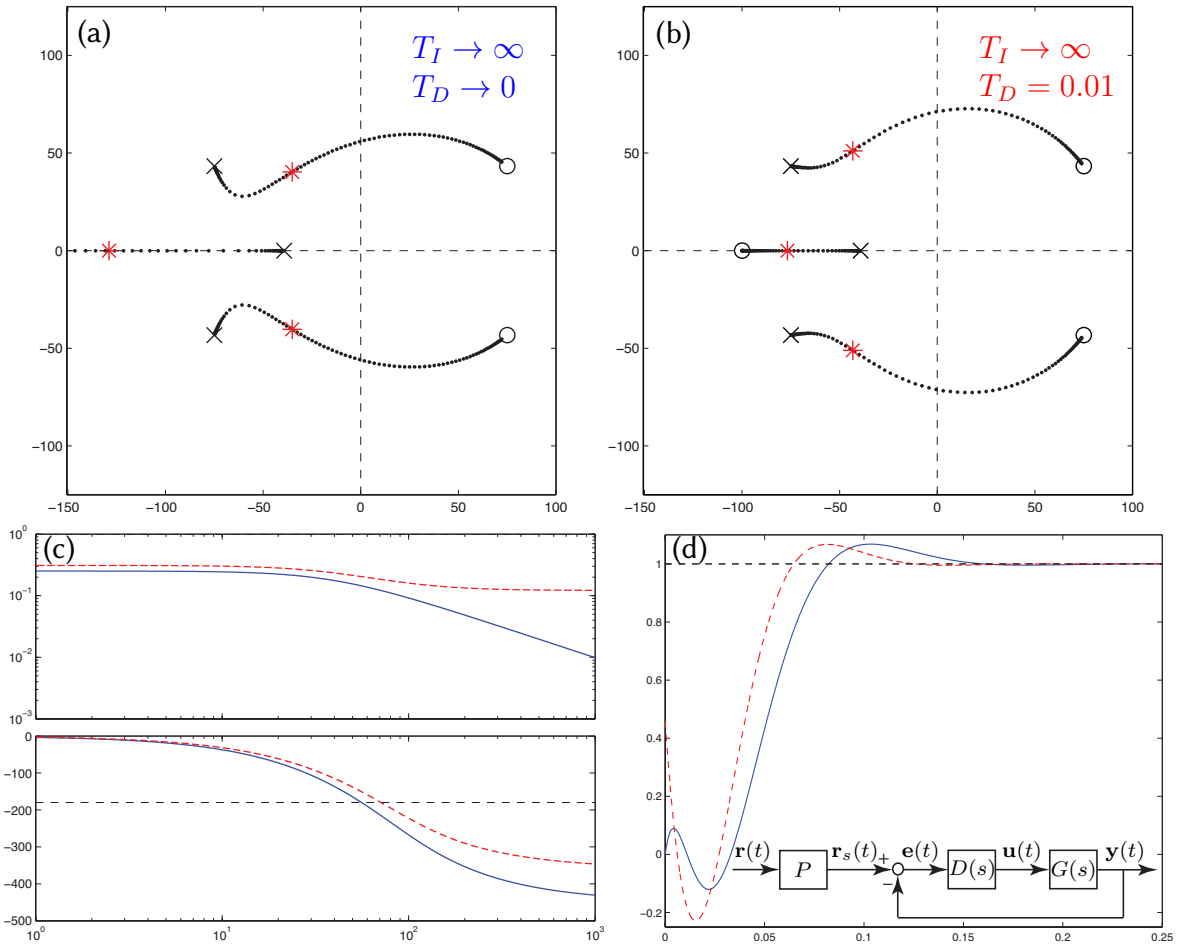


Figure 10.15: Application of P and PD control to the cruise control problem (10.15). (a) Root locus with respect to K when P control is applied. (b) Root locus with respect to K_p when PD control is applied, taking $T_D = 0.01$. (c) Bode plot and (d) step response in the (solid) P and (dashed) PD cases, with gains as marked in (a) and (b). In both cases, a loop prefactor P [see (10.11) and inset in (d)] has been used to achieve zero steady-state error in the step response, assuming $G(s)$ has been modeled accurately.

above suggest how one can back off from this critical value of K a bit, then dial in derivative compensation to improve the phase lead at crossover, and integral compensation to improve the low-frequency tracking. A controller designed using such rules is aggressive, as the K is selected to give a rise time that is essentially as fast as possible given the modeled value of the delay d. Such an approach is sensitive to a unmodeled effects, such as additional delays; a less aggressive approach (see Guideline 10.1) is thus generally preferred.

Figure 10.15 illustrates the effects of P and PD control applied to the automobile cruise control problem (10.15) with the delay approximated by its $n = m = 2$ Padé approximation $F_{2,2}(s)$ [see (8.8)]. Note that adding bit of derivative compensation to the proportional controller bumps up the phase at high frequencies, thus facilitating a slightly faster rise time for the same overshoot (as shown) or, alternatively, a slightly reduced overshoot for the same rise time. The low-frequency gain in both systems depicted in Figure 10.15c is finite (in fact, less than one); in order to achieve a zero steady-state error in the step response, a loop prefactor is used [see inset in Figure 10.15d], assuming $G(s)$ is modeled accurately so that the appropriate value of P can be computed. If instead there were significant modeling uncertainty (e.g., variable road grade), it would be beneficial to add integral compensation (or at least some significant lag compensation) to increase the open-loop low-frequency gain, $G(\mathrm{i}\omega)\,D(\mathrm{i}\omega)$ for small ω, and thus reduce the steady-state error even in the presence of such modeling error, as illustrated in Figure 10.16; note that applying such integral compensation slows down the response

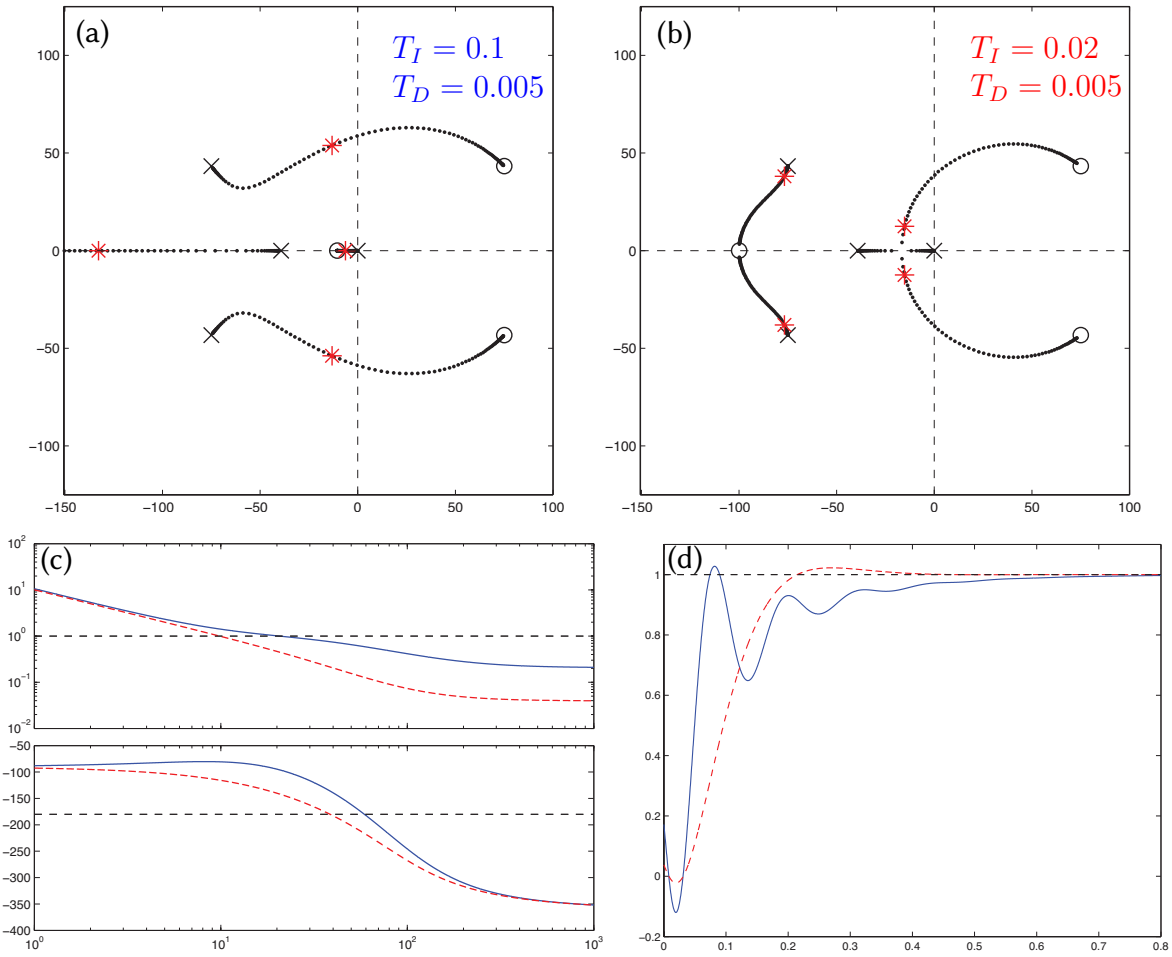


Figure 10.16: Application of PID control to the cruise control problem (10.15), taking $T_D = 0.005$. (a) Root locus with respect to K_p, taking $T_I = 0.1$. (b) Root locus with respect to K_p, taking $T_I = 0.02$. (c) Bode plot and (d) step response taking (solid) $T_I = 0.1$ and (dashed) $T_I = 0.02$, with gains as marked in (a) and (b). In both cases, the integral component of the controller $D(s)$ ensures a zero steady-state error in the step response, without a loop prefactor, even in the presence of significant uncertainty in the modeling of $G(s)$.

time as compared with that achievable when using a loop prefactor to scale the step response (Figure 10.15).

Finally, note that "performance" is not just about rise time, settling time, and overshoot. There is also a less quantifiable question of the response "quality"; a cruise control with the solid response curve depicted in Figure 10.16d would certainly lead to a very uncomfortable ride. Design engineers who tune the values of $\{K_p, T_I, T_D\}$ in the various PID loops in automobiles are thus to a large degree responsible for establishing the "feel" of a vehicle when these control systems are engaged, and are thus the ones that get paid the big bucks. △

The transfer function $G(s)$ in (10.15) is a good starting point for modeling systems of unknown structure. This model has three parameters, $\{C, d, a\}$, that can be tuned to match three critical system features that may be determined experimentally: the low-frequency gain $C_0 = C/a$ (including its sign), and both the gain $C_c = C/\sqrt{\omega_g^2 + a^2}$ and phase $\phi_c = -d - \text{atan}\,(\omega_g/a)$ of this system at a desired crossover frequency ω_g. If the system being modeled is open-loop stable (and thus $a > 0$), the values of $\{C_0, C_c, \phi_c\}$ of the open-loop system may be measured directly, from which the parameters $\{C, d, a\}$ may be determined immediately. If the system is unstable (that is, $a < 0$ in the model), appropriate model parameters $\{C, d, a\}$ may often be determined by applying stabilizing proportional feedback $D(s) = K$ and measuring both the lower and upper bounds, K_l and K_u, on the gain K for closed-loop stability, in addition to the frequency of oscillation, $\omega_u = 1/T_u$ at $K = K_u$.

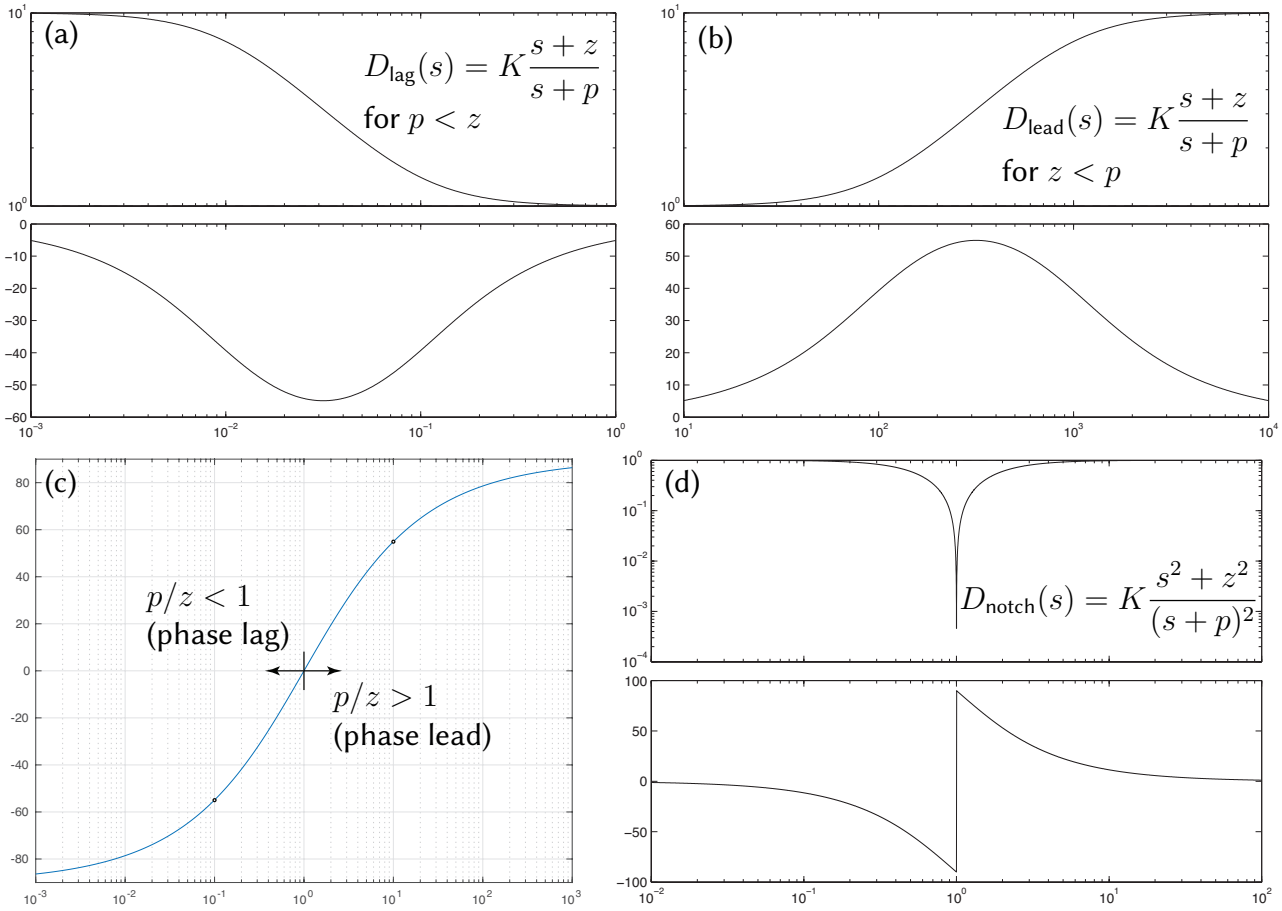


Figure 10.17: Bode plots of (a) $D_{\text{lag}}(s) = K(s+z)/(s+p)$ with $p < z$ (taking $K = 1$, $p = .01$, $z = .1$), for applying *integral* action over a targeted range of frequencies, and (b) $D_{\text{lead}}(s) = K(s+z)/(s+p)$ with $z < p$ (taking $K = p/z$, $z = 100$, $p = 1000$), for applying *derivative* action over a targeted range of frequencies. The cascade of these two of these controllers, $D_{\text{lead-lag}}(s) = D_{\text{lag}}(s) \cdot D_{\text{lead}}(s)$, gives the Bode plot in Figure 10.13b. (c) The peak phase lag (for $p/z < 1$) or lead (for $p/z > 1$) of a lag or lead controller, evaluated at $\omega = \sqrt{p\,z}$ and plotted versus $\alpha = p/z$. (d) Bode plot of a **notch** controller $D_{\text{notch}}(s) = K(s^2+z^2)/(s+p)^2$ (with $K = 1$, $z = 10$, $p = 10$), which is an alternative to lead control for oscillatory plants that are accurately modeled, effectively "knocking out" the oscillatory poles.

10.3.2 Lead, lag, and notch controllers

As illustrated in Figures 10.13b and 10.17a-b, **lead** and **lag** controllers provide the responsible way of adding derivative and integral actions over targeted ranges of frequencies; as with the derivative and integral components of a PID (Figures 10.13a), the lead and lag components are often used together; their cascade is referred to as a **lead-lag** controller. As illustrated in Figure 10.17c (see RR_tf/RR_evaluate), the peak value of the phase lead or lag provided by these controllers is about $55°$ for $|p/z| = 10$; this peak value approaches $90°$ for $|p/z| \to \infty$.

A third type of controller for oscillatory open-loop systems which are accurately modeled, called a **notch** controller (see Figure 10.17d), puts two complex controller zeros near the two oscillatory poles of the plant (in a sense, *knocking out* the oscillatory plant dynamics), replacing these two oscillatory poles with two stable poles on the negative real axis, far enough to the left to achieve a sufficiently fast settling time.

The effect of lead, lag, and notch controllers may be initially appreciated by considering their effects on the root loci of appropriate plants, but are more precisely tuned by considering the corresponding Bode plots. The tuning lead, lag, and notch controllers is best exemplified by considering the following examples.

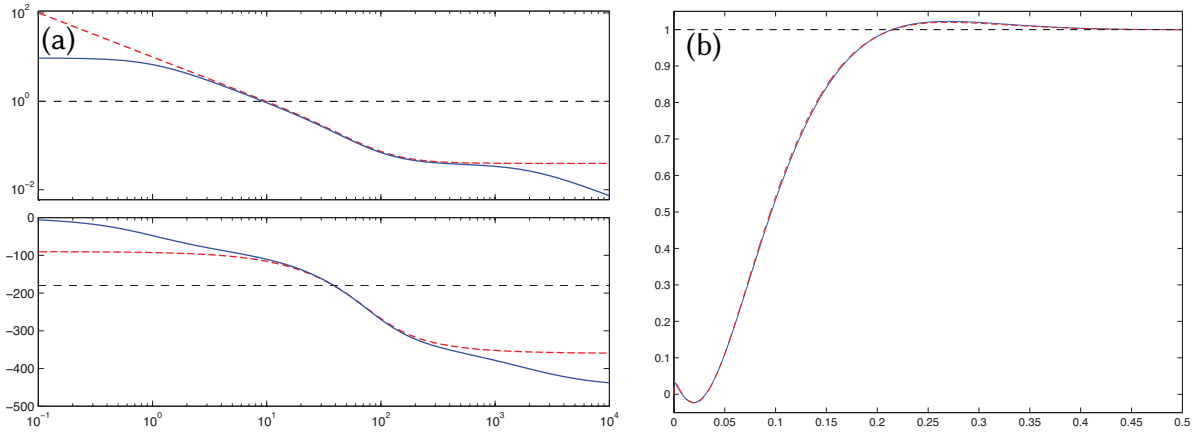


Figure 10.18: Comparison of (dashed) the best PID controller determined in Example 10.2 for the automobile cruise control problem [see (10.16a)] with (solid) the corresponding lead-lag controller of Example 10.3 [see (10.16b)] with $z/p = 100$ in the lag and $p/z = 20$ in the lead. (a) Bode plots, (b) step responses.

Example 10.3 Control of a first-order system with a delay, revisited using lead-lag control

We now revisit the PID controller design for the first-order system with a delay (a cruise control of an automobile) as developed in Example 10.2. The final PID control design achieved there is of the form given in (10.12) with $T_I = 0.02$, $T_D = 0.005$, and $K_p = 12000$, and thus

$$D_{\text{PID}}(s) = K_p\Big(1 + \frac{1}{T_I s} + T_D s\Big) = 60\frac{(s+100)^2}{s}. \tag{10.16a}$$

The closed-loop performance of the system considered with this controller applied is denoted by the dashed curve in Figure 10.16d, and is actually quite good in many respects, with a fast rise time and settling time (given the inherent limitations of the system imposed by the delay), low overshoot, zero steady-state error to a step input, and a high "quality" step response (little oscillation). However, what is hidden in this figure is the two problems associated with PID control discussed in §10.3.1: specifically, the amplification of high-frequency measurement noise, and integrator wind-up in the presence of actuator saturation. To alleviate both of these problems, as suggested in §10.3.1, we can instead apply lead-lag control of the form

$$D_{\text{lead-lag}}(s) = K \cdot \frac{s+100}{s+1} \cdot \frac{s+100}{s+2000} \qquad \text{where} \qquad K = 60 \cdot 2000 \cdot 0.95, \tag{10.16b}$$

where K has been selected (including the fudge factor 0.95) so that $D_{\text{lead-lag}}(s)$ in (10.16b) and $D_{\text{PID}}(s)$ in (10.16a) give about the same crossover frequency in Figure 10.18a. The (low-frequency) lag controller used here has $z/p = 100$, the (high-frequency) lead controller has $p/z = 20$, and the zeros of the lead-lag controller (10.16b) are in the same locations as the corresponding PID controller (10.16a). As seen in Figure 10.18b, *the system with PID implemented and the system with lead-lag implemented have nearly identical step responses, though the lead-lag controller has reduced gain at both high frequencies and low frequencies, thereby alleviating the two key problems with PID mentioned previously*. Note that a prefactor $P = 1.1$ [see (10.11)] has again been used in the case of the lead-lag controller; since the lead-lag controller's finite low-frequency open-loop gain, $G(\mathrm{i}\,\omega)\,D(\mathrm{i}\,\omega)$ for small ω, is relatively large (about 10; see Figure 10.18a), the tracking of the system will still be quite good even in the presence of minor modeling errors. △

Example 10.4 Speeding up a (generic) first-order system with lag or lead control

The previous example highlighted the utility of lag compensation to boost the open-loop gain at frequencies well below the crossover frequency, thereby reducing the steady-state error of a step response. This is, perhaps, the primary use of lag control; however, lag control isn't always used below crossover.

Consider now the generic form of a linear first-order system

$$\frac{dy(t)}{dt} + ay(t) = C\Big[\frac{du(t)}{dt} + bu(t)\Big] \quad \Leftrightarrow \quad \frac{Y(s)}{U(s)} = G(s) = C\,\frac{s+b}{s+a}, \tag{10.17}$$

where a and b are real, with $a \neq b$. If this system is controlled with proportional feedback $u = K\,y$ then, by the discussion in the first paragraph of §10.2.1, the pole of the closed-loop system is given by

$$(s+a)/K + C(s+b) = 0 \quad \Rightarrow \quad s = -\frac{a + KCb}{1 + KC}.$$

For small values of K, the pole of the closed-loop system is near the pole of the open-loop system, $s = -a$, and for large values of K, the pole of the closed-loop system is near the zero of the open-loop system, $s = -b$. For intermediate values of K, the pole of the closed-loop system is somewhere in between (on the real axis between $s = -a$ and $s = -b$). For some systems, this is adequate to move the closed-loop pole sufficiently far to the left to achieve the specified rise time and settling time criteria. For other systems, however, none of these possible closed-loop pole locations is acceptable. Rather than flip the sign of the gain and use a potentially high feedback amplitude (following the 0° root locus rules to determine a K that achieves stability and an adequate rise time and settling time), one can instead use (positive-gain) lag or lead control, as appropriate, and an approximate pole-zero cancellation in the LHP (see §10.2.1.2).

To illustrate, if in (10.17) the plant zero is to the left of the plant pole (e.g., if the plant itself has the form of a lag compensator) and the plant zero is stable (that is, $b > 0$), one may use a lag controller

$$D_{\text{lag}}(s) = K\frac{s+z}{s+p} \quad \text{with} \quad z > p \approx b$$

to "replace" the slow stable plant zero with a faster stable controller zero; tuning K then provides the requisite rise time and settling time in the closed-loop system, as well as providing low steady-state error in the step response even in the presence of significant uncertainty in the modeling of $G(s)$.

If, on the other hand, if in (10.17) the plant pole is to the left of the plant zero (e.g., if the plant itself has the form of a lead compensator) and the plant pole is stable (that is, $a > 0$), one may use a lead controller

$$D_{\text{lead}}(s) = K\frac{s+z}{s+p} \quad \text{with} \quad p > z \approx a$$

to replace the slow stable plant pole with a faster stable controller pole; tuning K then again provides the requisite rise time and settling time in the closed-loop system.

In either case, to achieve sufficiently low steady-state error in the step response even in the presence of significant uncertainty in the modeling of $G(s)$, we might need to boost up the low-frequency gain of the controller $D(s)$ beyond that provided by the controllers described above; this may be accomplished via additional lag compensation (e.g., applying $D_{\text{lag}}(s)$, or $[D_{\text{lag}}(s)]^2$, from Figure 10.17) at low frequencies.

Additionally, in either case, to achieve sufficient robustness to external disurbances, we might need to diminish the high-frequency gain of the controller $D(s)$; this may be accomplished via additional low-pass filtering (again, see §8.5 and Figure 8.8) at high frequencies. △

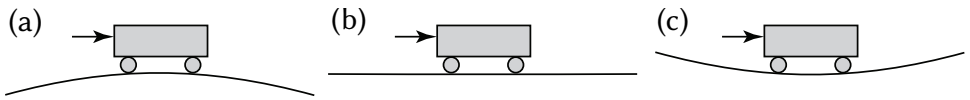


Figure 10.19: A second-order rolling cart system governed by (10.18) in: (a) an unstable configuration with $\alpha < 0$, (b) a neutrally-stable configuration with $\alpha = 0$, and (c) an oscillatory configuration with $\alpha > 0$.

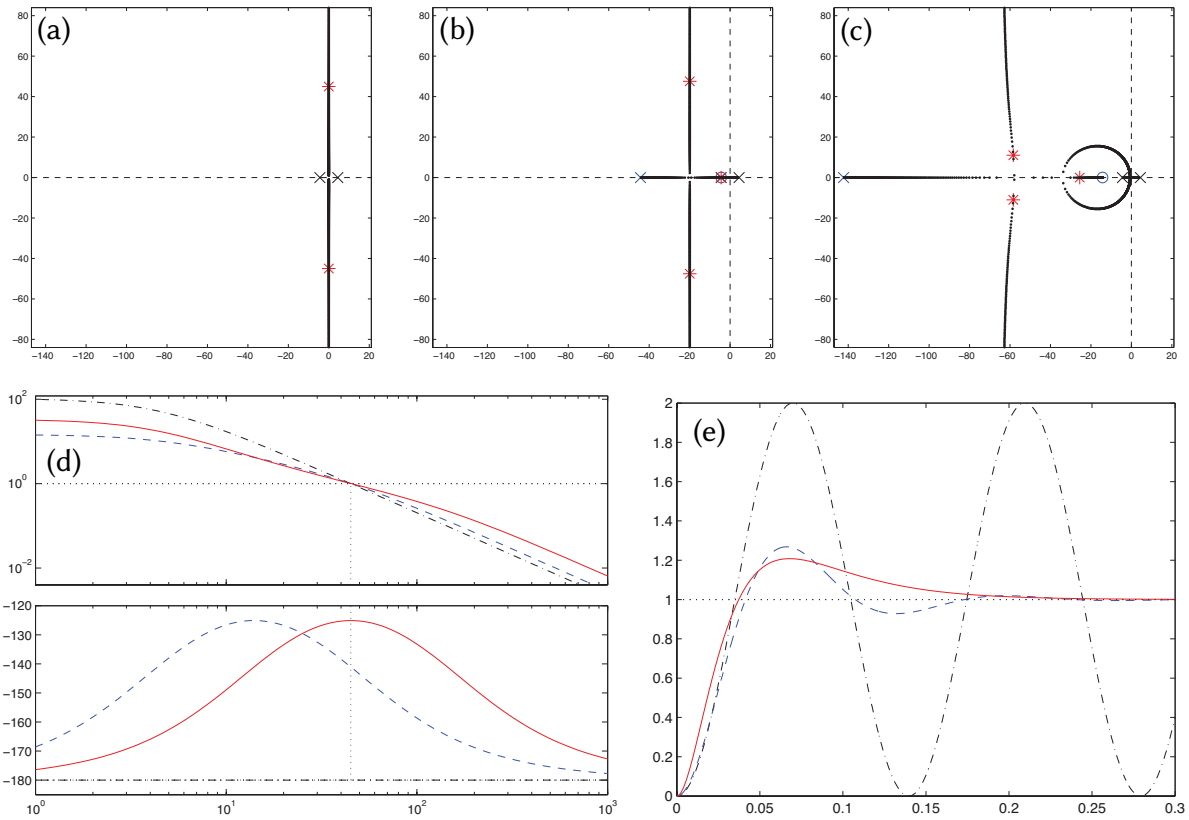


Figure 10.20: Control of the unstable second-order system of Figure 10.19a. (a) Root locus using proportional control. (b) Root locus using lead control designed for simple pole/zero cancellation. (c) Root locus using lead control designed (using the Bode plot) for maximum performance. (d) Bode plot and (e) step response using (dot-dashed) proportional control, (dashed) lead control designed (using root locus) for simple pole/zero cancellation, taking $p/z = 10$, and (solid) lead control designed for maximum PM given $p/z = 10$ (by taking $\sqrt{p\,z} = \omega_g$), with gains as marked in (a), (b), and (c).

Example 10.5 Stabilizing a second-order rolling cart system with lead or notch control

Consider now the problem of controlling the position of a simple rolling cart (see Figure 10.19) governed by

$$m\,\frac{d^2x}{dt^2} + c\,\frac{dx}{dt} + m\,g\,\sin\left(\frac{dy}{dx}\right) = u, \quad y = \beta\,x^2 \quad \xrightarrow[\text{linearization}]{} \quad \frac{d^2x}{dt^2} + \left(\frac{c}{m}\right)\frac{dx}{dt} + (2\beta g)\,x = \left(\frac{1}{m}\right)u$$
$$\Rightarrow \quad \frac{X(s)}{U(s)} = \frac{C}{s^2 + a_1 s + a_0} \quad \text{where} \quad a_1 = \frac{c}{m}, \;\; a_0 = 2\beta g, \;\; C = \frac{1}{m}. \tag{10.18}$$

Case (a): $\beta = -1$, $c = 0$, $m = 1$. The plant in this case, $G(s) = C/[(s+\overline{p})(s-\overline{p})]$ with $\overline{p} = \sqrt{2|\beta|g}$, is unstable. Lead control designed using the root locus with an approximate pole-zero cancellation in the LHP, $D_{\text{lead}}(s) = K\,(s+z)/(s+p)$ with $p > z \approx \overline{p}$ and the controller pole $s = -p$ taken sufficiently far into the LHP, stabilizes this system, as illustrated in Figure 10.20b. However, lead control designed using the Bode plot, with the pole/zero pair [and, thus, the boost in gain caused by $D_{\text{lead}}(s)$] centered at the desired crossover frequency

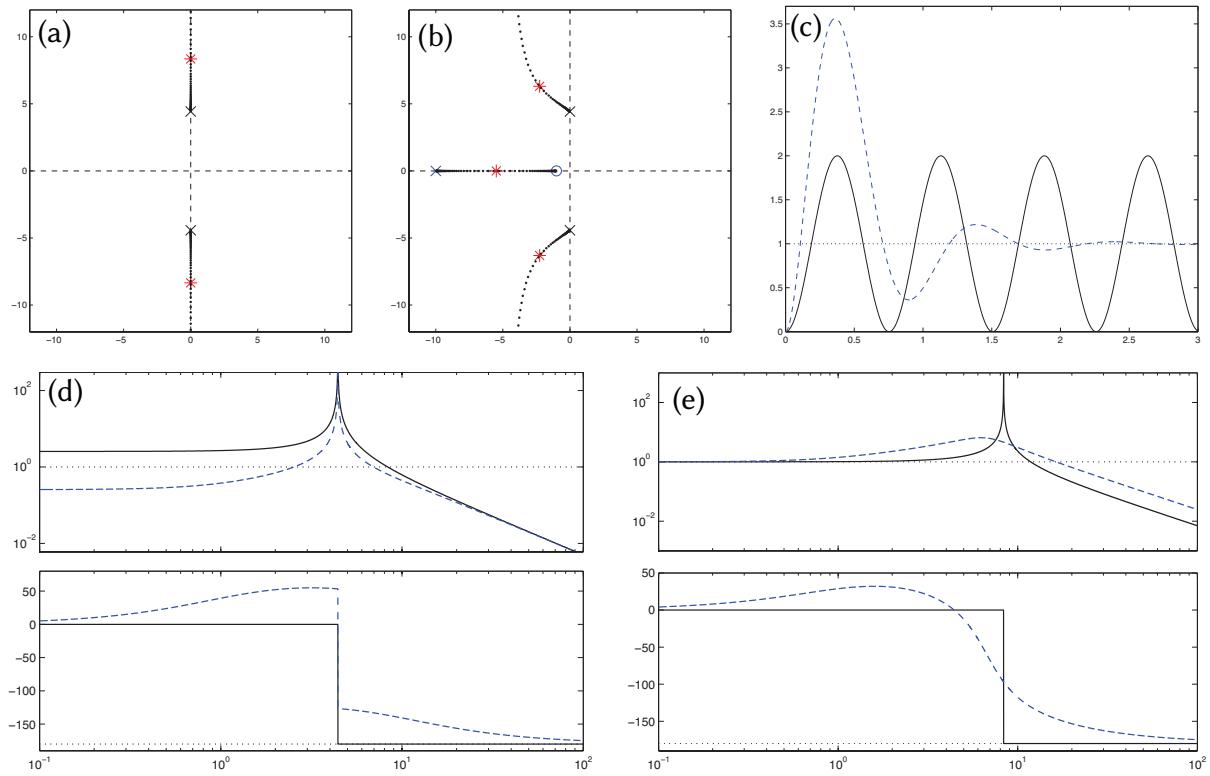


Figure 10.21: Control of the oscillatory second-order system of Figure 10.19c using proportional control and lead control. (a) Root locus using proportional control. (b) Root locus using lead control. (c) Step response, (d) open-loop Bode plot, and (e) closed-loop Bode plot using (solid) proportional control and (dashed) lead control taking $p/z = 10$, with gains as marked in (a) and (b). Note the peaks in the open-loop and (in the case of proportional control) closed-loop Bode plots due to the open-loop and (in the case of proportional control) closed-loop poles on the imaginary axis. Unfortunately, when using lead control in this case, there is significant overshoot in the step response due to the third pole on the negative real axis.

as determined from the rise time specification (that is, with $\sqrt{pz} = \omega_g$, as shown in Figure 10.20d), improves the response (see Figure 10.20e) for the same value of $\alpha = p/z$ in the lead controller (taking $\alpha = 10$ in this case[12], as shown in Figure 10.20d), even though the root locus in this case is a bit more complicated (Figure 10.20c). Note finally that a bit of damping in the plant (taking $c > 0$) changes the pole locations slightly, but leaves the control problem essentially unchanged.

Case (b): $\beta = 0$, $c = 0$, $m = 1$. The plant in this case, $G(s) = C/s^2$, is known as a **double integrator**, and the lead control strategy using the Bode plot, as described in Case (a), works effectively.

Case (c): $\beta = 1$, $c = 0$, $m = 1$. The plant in this case, $G(s) = C/[(s + \mathrm{i}\overline{p})(s - \mathrm{i}\overline{p})]$ with $\overline{p} = \sqrt{2\beta g}$, is known as a **second-order oscillator**. Lead control can be designed for this system using the Bode plot as in the previous two cases, leading to the step response illustrated in Figure 10.21. Note the increased overshoot in this case due to the extra closed-loop pole on the negative real axis; this increased overshoot is generally the weakness of using lead control on an oscillatory plant. The strength of this control design is that it generally works adequately (but, not particularly well...) for a broad range of $\overline{p}$; that is, the control design is *robust* to uncertainty in the plant parameters.

[12] Note that a lead controller designed with $\sqrt{pz} = \omega_g$ and $p/z = \alpha$ is given simply by taking $z = \omega_g/\sqrt{\alpha}$ and $p = \omega_g\sqrt{\alpha}$.

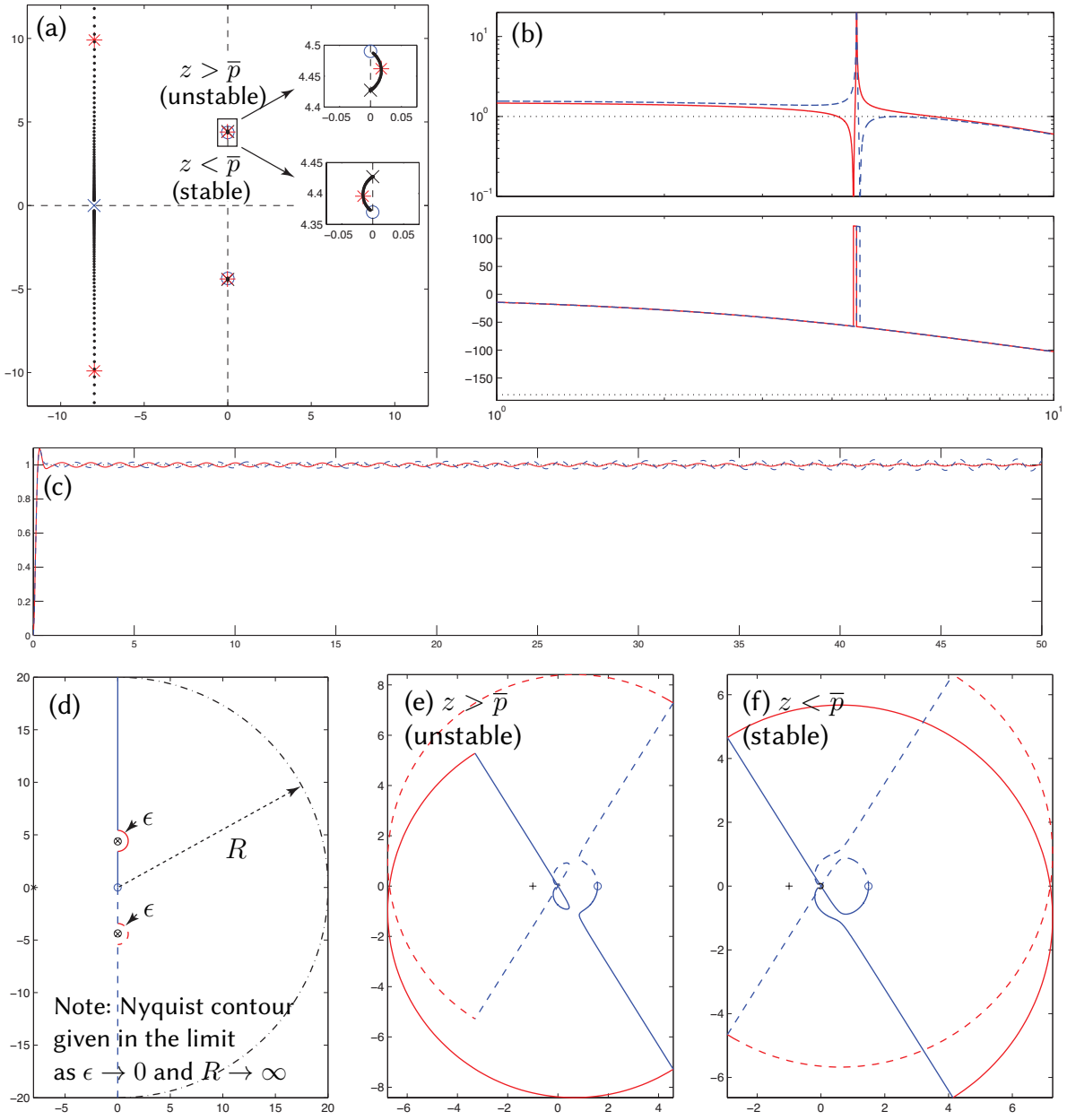


Figure 10.22: Control of the oscillatory second-order system of Figure 10.19c using notch control for two possible values for the controller zeros. (a) Root locus. (b) Bode plot and (c) step response taking (solid) $z < \overline{p}$ and (dashed) $z > \overline{p}$, with gains as marked in (a). The inexact cancellation of the plant pole $\overline{p}$ by the controller zero z leads to a stable closed-loop system, in this case, if $z < \overline{p}$, and instability if $z > \overline{p}$. This is seen in the root locus very near the imaginary axis (see insets); it is seen in the step response only by looking over a long period of time, as the (initially, small) oscillations grow or decay slowly. It is difficult to determine stability from the Bode plot. Designing an appropriate Nyquist contour in the s plane, (d), and mapping via $L(s) = G(s)\,D(s)$ to obtain the corresponding Nyquist plots [see (e) for $z > \overline{p}$, and (f) for $z < \overline{p}$], it is seen in (e) that the contour encircles the $L = -1$ point twice when $z > \overline{p}$, thus [by Fact 10.3] indicating two unstable closed-loop poles, whereas in (f) the contour does not encircle the $L = -1$ point when $z < \overline{p}$, indicating a stable closed loop.

Case (c): $\beta = 1$, $c = 0$, $m = 1$, *revisited using notch control.* If the plant is known accurately, a notch controller $D_{\text{notch}}(s) = K(s^2 + z^2)/(s + p)^2$ (see Figure 10.17d) can be used instead in case (c), approximately "knocking out" the neutrally-stable plant poles with nearby controller zeros, and supplanting these oscillatory poles with a pair of stable controller poles sufficiently far into the LHP to achieve the desired rise time and settling time, as illustrated in Figure 10.22. The strength of this control design is its performance: the closed-loop system in this case doesn't suffer from the increased overshoot experienced when using lead control. The weakness of this control design is that it is sensitive to uncertainty in the plant parameters. Recall that cancellation of plant poles/zeros with controller zeros/poles must always be considered as approximate, as plant parameters are never known precisely. If a pole/zero cancellation is well into the LHP, the fact that the cancellation is only approximate isn't a problem, as discussed in §10.2.1.2. However, if the approximate pole/zero cancellation is on or near the imaginary axis, then it is critical that the controller zero be placed *on the appropriate side*[13] of the plant pole, so that the small branch of the root locus that results from the inexact nature of the cancellation swings into the LHP instead of the RHP. In other words, the controller zero should not be put at the expected *value* of the plant pole, but must instead be put, conservatively, on the appropriate side of the expected *range* of where the plant pole might lie. The smaller this range is (that is, the more certain you are about the plant parameters), the better the performance that can be obtained, as this cancellation can be made more precise without risking instability. If the range is too large, the more robust lead controller should be used instead. △

10.3.3 Sensor dynamics, and noise suppression via low-pass and notch filtering

All sensors have associated with them some level of noise, the intensity of which generally varies as a function of frequency. This noise intensity often does not diminish very quickly with increasing frequency[14], though the strength of the signal of interest usually does; thus, sensors ultimately become essentially useless at high frequencies, and feedback applied at such high frequencies is counterproductive. The typical Bode plot depicted in Figure 10.7 therefore has the important constraint of reduced open-loop gain at high frequencies.

To suppress the high-frequency gain, a low-pass filter (see §8.5) is sometimes necessary. Recall the Bode plots of the simple first-order and second-order low-pass filters given in Figure 8.8, and of the higher-order Butterworth and Bessel filters given in Figure 8.10. Unfortunately, such filters generally bring with them significant *phase delay*, even well below the filter's corner frequency ω_c, as indicated in these Bode plots. Recall also that one generally designs a feedback system with a certain minimum phase at the crossover frequency, given by $-180°+$ PM where, as suggested by (10.8), PM $\approx \zeta \cdot 100$, and ζ is the desired damping selected to meet the overshoot specification, as suggested by (8.17). If a low-pass filter is used with corner frequency ω_c within an order of magnitude or so of the crossover frequency ω_g of the system, this phase loss should be accounted for during the controller design.

Thus, the selection and tuning of a low-pass filter to suppress the response of a system to the high-frequency noise picked up by the sensors is nontrivial, and reflects a compromise between the loss of phase and the rate of roll-off of the gain near the corner frequency ω_c of the low-pass filter. Many sensors already have low-pass filters built in, or superciliously added by the experimentalist constructing the sensor. As the responsible controls engineer, if one is designing for maximum performance, it is useful to identify exactly what kind of filter is already implemented in such settings, so the phase loss associated with this filter at crossover can be accounted for during the controller design.

[13] Note that which side (above or below) is the appropriate side that gives stability depends on the rest of $G(s)$; you need to plot the root locus for your problem to check!

[14] In many sensors (e.g., thermocouples) the noise intensity can actually be well approximated as **white** (that is, with intensity nearly constant across a broad range of frequencies, like that of white light; for further discussion, see §5 of *NR*).

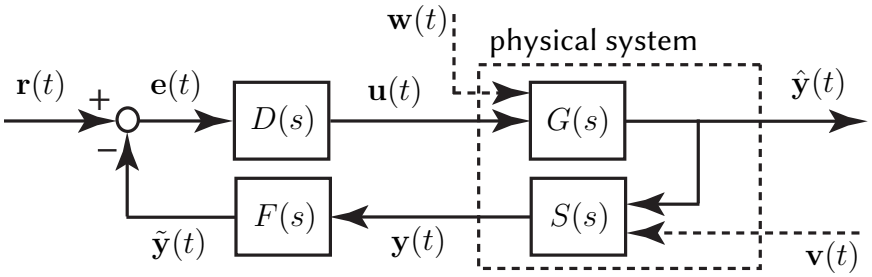


Figure 10.23: Closed-loop system with a **sensor dynamics block** $S(s)$ and a **filter block** $F(s)$ in the return portion of the feedback loop, where $\mathbf{y}(t)$ denotes the actual measurement, $\hat{\mathbf{y}}(t)$ denotes the ideal measurement (that is, the quantity of interest), and $\tilde{\mathbf{y}}(t)$ denotes the filtered measurement.

Some sensors have inherent dynamics that may be significant near the target crossover frequency. For example, a **MEMS**[15] accelerometer is essentially a small floating mass balanced by a spring, and therefore has a response magnitude that is inherently a function of the forcing frequency[16]. Some **COTS** (**commercial off-the-shelf**) sensors have appropriate compensation circuits already built in which attempt to counteract the effects of such sensor dynamics, while others allow the controls engineer access to the raw measured signal, allowing the appropriate compensation to be developed as a part of the closed-loop system design.

In order to maximize performance [that is, the fidelity of the system response to a reference input $\mathbf{r}(t)$] while minimizing sensitivity [that is, the response of the system to measurement noise $\mathbf{v}(t)$], control in the presence of sensor dynamics and/or low-pass filtering is often implemented in the manner indicated in Figure 10.23. In this case, again noting the mnemonic in (10.2), the **closed-loop transfer function** is

$$T(s) = \frac{\hat{Y}(s)}{R(s)} = \frac{G(s)\,D(s)}{1 + G(s)\,D(s)\,F(s)\,S(s)}, \tag{10.19a}$$

whereas the **sensitivity** [of $\hat{\mathbf{y}}(t)$ to the measurement noise $\mathbf{v}(t)$] is

$$\frac{\hat{Y}(s)}{V(s)} = \frac{G(s)\,D(s)\,F(s)\,S(s)}{1 + G(s)\,D(s)\,F(s)\,S(s)}. \tag{10.19b}$$

It is thus seen that low-pass filter $F(s)$ [with $|F(\mathrm{i}\,\omega)| \to 0$ as $\omega \to \infty$] in the return portion of the feedback loop reduces the sensitivity of the quantity of interest $\hat{\mathbf{y}}(t)$ to the high-frequency components of the measurement noise $\mathbf{v}(t)$, but not to the high-frequency components of the reference input $\mathbf{r}(t)$. Similarly, a notch filter $F(s)$ (see Figure 10.17c) reduces the response of the system to a dominant frequency in the spectrum of the measurement noise $\mathbf{v}(t)$. Note finally that measurement noise sometimes has a dominant frequency component at a certain frequency (often, 50 or 60 Hz); this **buzz** can be mitigated with a notch filter in the return portion of the feedback loop in a similar fashion (see Example 9.31).

[15] A **Micro-Electro-Mechanical-System** is a very small physical system made using the same mask/expose/etch technology used to manufacture silicon chips. Today, this technology is very mature, and several types of MEMS sensors are mass produced on a large scale. For example, MEMS accelerometers are used in airbag deployment systems in automobiles, video game controllers & smartphones, and laptop hard disk drives (to park the read head, before impact, if the laptop is dropped, thus preventing damage to the disk). MEMS gyros are also mass produced, albeit on a somewhat smaller scale, for use in video game controllers & smartphones.

[16] An expanded dynamic range of such devices may generally be obtained by active electrostatic **force rebalancing**; that is, by closing a control loop around the sensor itself, applying an electrostatic force that is just sufficient to keep the floating mass from moving, then measuring the electrostatic force required to "rebalance" the floating mass within the device in order to determine the acceleration applied to the entire system. This essentially supplants the mechanical time constant of the device, $\sqrt{m/k}$, with the electrical time constants of the sensor control circuit, RC and L/R, which are generally much faster.

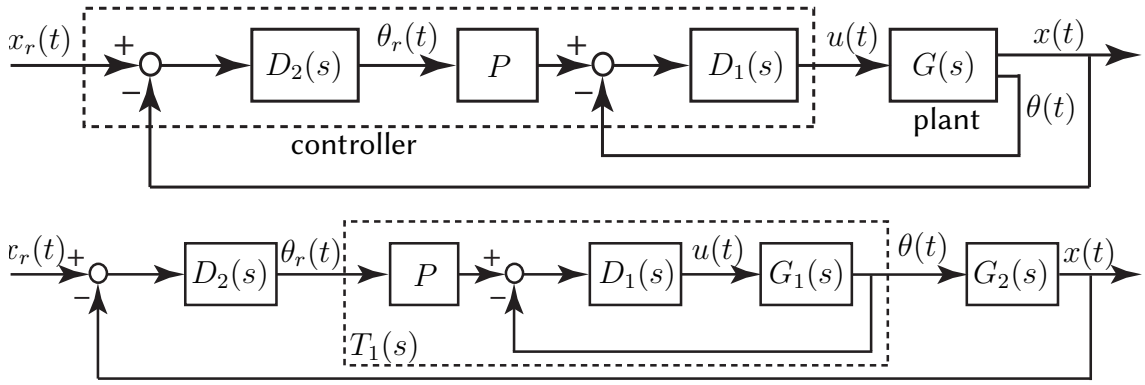


Figure 10.24: Successive loop closure of a SIMO plant $G(s)$ designed with "fast" $\theta(t)$ dynamics and "slow" $x(t)$ dynamics: (a) block diagram of physical realization; (b) idealization for the purpose of control design, as considered in Example 10.6 (see Figure 6.5a).

10.3.4 Successive loop closure (SLC) leveraging frequency separation

Classical control design techniques are used primarily for SISO systems. However, many systems are characterized by a substantial **frequency separation** of the modes of interest; that is, the system modes are characterized by natural frequencies which are an order of magnitude or so apart, or the frequencies of such modes can at least be made this way with control feedback. In such systems, it is straightforward to nest a "fast" feedback loop within one or more outer, "slower" feedback loops in order to string multiple SISO controllers together to stabilize a SIMO system with multiple unstable modes, leveraging the observation that a slow outer loop doesn't significantly alter the dynamics of a fast inner loop. As with the other controls problems considered previously, this control strategy is best introduced by example.

Example 10.6 Stabilization of an unstable fourth-order system (an inverted pendulum on a cart)

We now consider the control of the classic linearized inverted pendulum problem illustrated in Figure 6.5a and depicted in block diagram form in Figure 10.24, taking typical laboratory values for the constants: $m_c = 2$ kg for the mass of the cart, $m_p = 1$ kg for the mass of the pendulum (a uniform rod), $\ell = 1$ m for the distance from the cart to the center of mass of the pendulum (the half-length of the rod), $I_p = m_p\ell^2/3$ for the moment of inertia of the pendulum about its center of mass, $g = 9.8$ m/s^2 for the acceleration due to gravity, and $c_1 \approx 0.01$ and $c_2 \approx 0.05$ for the friction coefficients. As derived in Example 6.5, small perturbations of this system from the inverted state are governed by the coupled equations

$$\text{pendulum dynamics:} \quad (I_p + m_p\ell^2)\frac{d^2\theta}{dt^2} + c_1\frac{d\theta}{dt} - m_p g\ell\,\theta = m_p\ell\frac{d^2x}{dt^2}, \tag{10.20a}$$

$$\text{cart dynamics:} \quad (m_c + m_p)\frac{d^2x}{dt^2} + c_2\frac{dx}{dt} = m_p\ell\frac{d^2\theta}{dt^2} + u, \tag{10.20b}$$

Taking the Laplace transform of both (10.20a) and (10.20b) and combining appropriately, we may write

$$G_1(s) = \frac{\Theta(s)}{U(s)} = \frac{b_1 s}{a_3 s^3 + a_2 s^2 - a_1 s - a_0}, \quad G_2(s) = \frac{X(s)}{\Theta(s)} = \frac{\bar{b}_2 s^2 + \bar{b}_1 s - \bar{b}_0}{s^2} \tag{10.21}$$

where $b_1 = m_p\ell$, $a_3 = (m_c+m_p)\,I_p + m_c\,m_p\,\ell^2$, $a_2 = (m_c+m_p)\,c_1 + (I+m_p\,\ell^2)\,c_2$, $a_1 = (m_c+m_p)\,m_p\,g\,\ell + c_1\,c_2$, $a_0 = m_p\,g\,\ell\,c_2$, $\bar{b}_2 = (I + m_p\,\ell^2)/(m_p\,\ell)$, $\bar{b}_1 = c_1/(m_p\,\ell)$, and $\bar{b}_0 = g$, thereby facilitating interpretation of the SIMO system (10.20) in the **successive loop closure** configuration depicted in Figure 10.24. The problem of stabilizing the **hanging pendulum**, which is oscillatory (not unstable), is closely related.

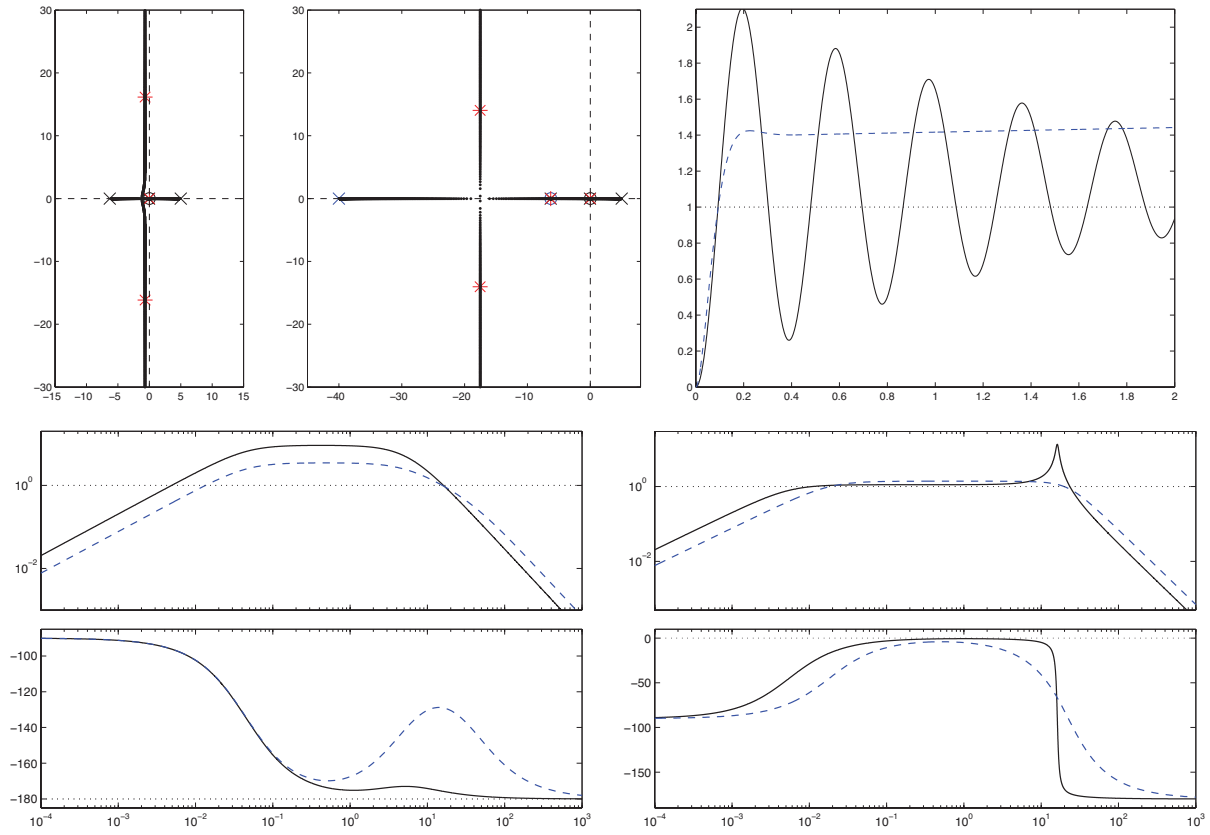


Figure 10.25: Classical control design techniques applied to the *inner loop* of the successive loop closure problem considered in Example 10.6. (a) Root locus using proportional control. (b) Root locus using lead control. (c) Step response, (d) Bode plot, and (e) closed-loop Bode plot using (solid) proportional control and (dashed) lead control, with gains as marked in (a) and (b).

The key to extending the essentially SISO control techniques discussed thus far to this problem is to first ignore the state $x(t)$ associated with what we have assigned as the "slow" outer loop, and focus our attention at first simply on designing an inner-loop controller $D_1(s)$ to stabilize (relatively quickly; say, with a rise time of $t_r \approx 0.1$ sec) the variable $\theta(t)$ back to a reference state $\theta_r(t)$, nominally taken to be zero.

Taking $c_1 = c_2 = 0$ reduces the plant considered in the inner loop to $G_1(s) = b_1/(a_3 s^2 - a_1)$, and thus the inner loop control problem reduces to the problem solved in Figure 10.20. In the problem considered now, neither of these friction coefficients is zero; however, as shown in Figure 10.25, using lead control and the root locus and Bode design techniques presented previously, stabilization of the high-speed dynamics of the inner loop is again quite straightforward. The subsystem to be controlled in the inner loop, $G_1(s)$, has an open-loop zero at $s = 0$ and open-loop poles at $s \approx -6.3$, $s \approx -0.045$, and $s \approx 5$. Thus, proportional control results in a pair of lightly damped, relatively fast closed-loop poles, and one very slow unstable closed-loop pole on the positive real axis close to $s = 0$. As before, the dominant fast modes of the system are easily stabilized with lead compensation. Due to the proximity of the open-loop zero at $s = 0$ to the open-loop pole at $s \approx -0.045$, the coefficient of the (exponentially growing) component of the response associated with the very slow unstable closed-loop pole is nonzero but small, as illustrated in the (dashed) step response in Figure 10.25c. Again, it is difficult to determine stability versus instability directly from a Bode plot, though this would be straightforward to determine from a Nyquist plot. This is unnecessary in the present situation, as we already know that the inner loop is unstable by looking at the root locus.

Note in Figure 10.25c that the step response of the controlled inner loop rises quickly to 1.4, and begins to drift from there due to the slow unstable inner-loop pole. Thus, as suggested by Figure 10.11, we take $P = 1/1.4$

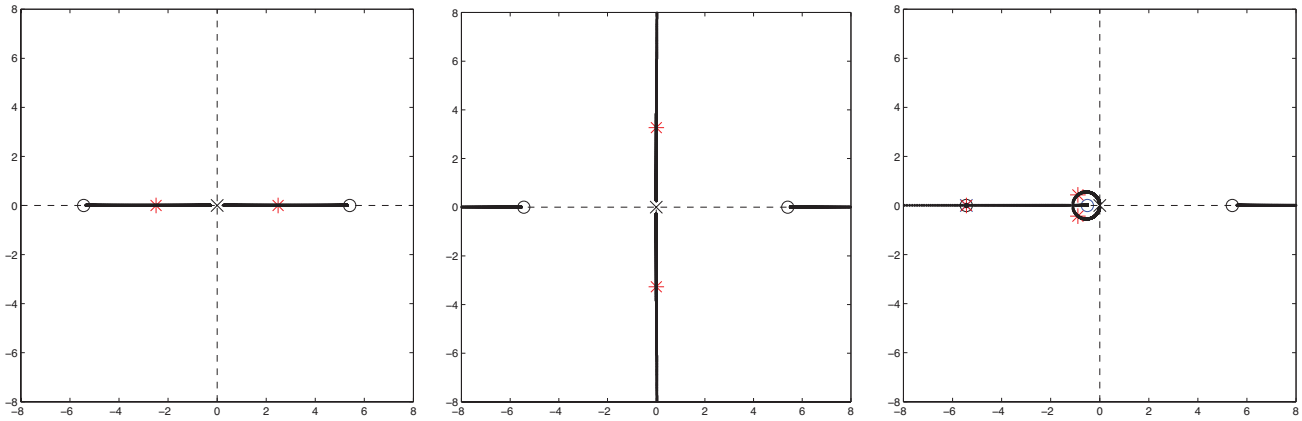


Figure 10.26: Classical control design techniques applied to the *idealized outer loop* [taking $T_1(s) = 1$] of the successive loop closure problem considered in Example 10.6. (a) Root locus using proportional control. (b) Root locus using proportional control with negative gain. (c) Root locus using lead control with negative gain. The corresponding Bode plot and step response are given as the (dashed) curves of Figures 10.27b-c.

in Figure 10.24. For the time being, we simply ignore the unstable slow pole of the inner loop, as the outer-loop control design, considered below, changes significantly the low-speed dynamics of the system. Doing the calculation, the actual closed inner loop $T_1(s) = P\,G_1(s)\,D_1(s)/[1+G_1(s)\,D_1(s)]$ has poles at $s \approx 0.018$ (which, again, is unstable, as seen clearly in Figure 10.25b) and $s \approx -17.5 \pm 14$i (which are fast and stable), a zero at $s = 0$, and an approximate (stable) pole/zero cancellation at $s \approx -6.3$.

Assuming the closed inner loop has a transfer function of $T_1(s) \approx 1$ at the frequencies of interest (see dashed curve in Figure 10.25e, scaled by $P = 1/1.4$), we next design an outer-loop controller $D_2(s)$ to stabilize $G_s(s)$ in (10.21), in the outer loop of Figure 10.24b, relatively slowly (say, with $t_r \approx 1$ sec). Noting $G_2(s)$, it is seen in Figure 10.26 that a $D_2(s)$ with negative gain and some lead compensation effectively stabilizes the idealized outer-loop system.

Finally, applying the outer-loop controller $D_2(s)$ to $G_2(s)$ and the actual closed inner loop, $T_1(s)$, results in the root locus, Bode plot, and step response in Figure 10.27. The root locus of this full outer-loop system (Figure 10.27a) indicates essentially the same behavior at low frequencies (small $|s|$) as does the root locus of the idealized outer-loop system (Figure 10.26c), as well as a pair of well-damped closed-loop poles near $s \approx -16.3 \pm 3$i, the latter of which (in slightly modified positions) is also seen in the root locus of the inner-loop system (Figure 10.25b), as well as a couple of (stable) approximate pole/zero cancellations at $s \approx -6.3$ and $s \approx -5.4$. The Bode plot of this full outer-loop system (solid curve in Figure 10.27b) is similar to that of the idealized outer-loop system (dashed curve in Figure 10.27b), but exhibits a significant gain reduction and loss of phase at high frequencies. It is thus clearly evident in this Bode plot why frequency separation is needed in order to apply the SLC approach: the loss of phase due to the (fast) inner loop dynamics erodes the PM of the outer loop if the crossover frequency ω_g of the outer loop is too close to the bandwidth frequency ω_{BW} of the closed inner loop, as evident in the closed-loop Bode plot of the inner loop in Figure 10.25e. Due to the only slight loss of phase from the inner loop dynamics for the rise times achieved in the present control solution ($t_r \approx 0.08$ sec for the inner loop, and $t_r \approx 0.8$ sec for the outer loop), the step response of the full outer-loop system has only slightly higher overshoot than the step response of the idealized outer-loop system, as indicated in Figure 10.27c.

We now reëxamine the slow outer-loop dynamics of the full system, revisiting the interplay between the slow dynamics of the closed inner loop and the outer loop. As mentioned above, and illustrated clearly by the root loci of Figure 10.26 and the Bode plot and step response given as the dashed curves of Figures 10.27b-c, control of the idealized outer-loop system $G_2(s)$ [with two poles at the origin and two zeros at $s \approx \pm 5.4$] is straightforward using negative gain and lead compensation.

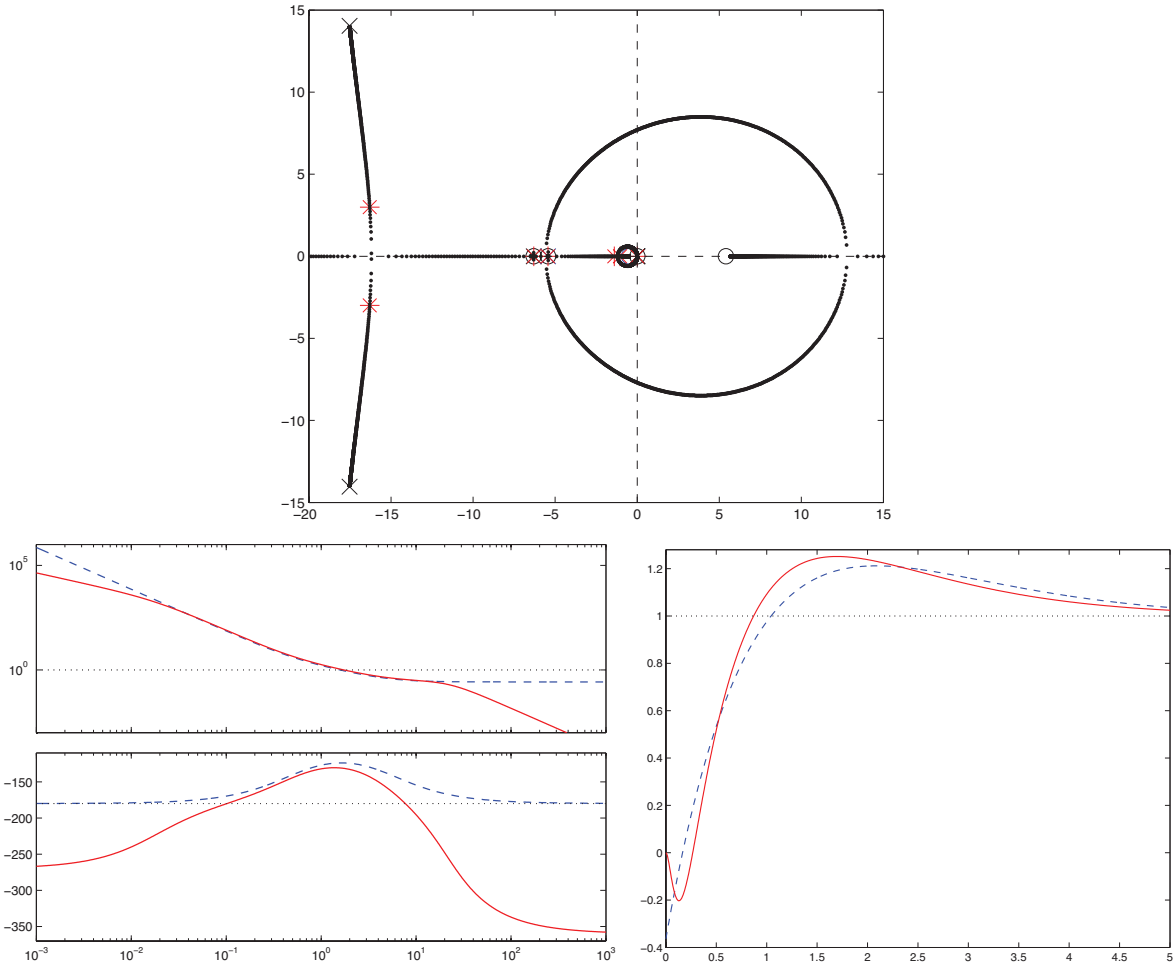


Figure 10.27: (a) Root locus of the full outer loop, incorporating the actual closed inner loop $T_1(s) = P\,G_1(s)\,D_1(s)/[1 + G_1(s)\,D_1(s)]$, of the SLC problem in Example 10.6. (b) Bode plot, and (c) step response of (dashed) the idealized outer loop [taking the idealized closed inner loop $T_1(s) = 1$] corresponding to the root locus in Figure 10.26c, and (solid) the full outer loop, incorporating the actual closed inner loop $T_1(s)$, corresponding to the root locus in (a).

When forming the product $G_2(s)\,T_1(s)$, one of the poles at the origin of $G_2(s)$ cancels (*exactly*[17]) the zero at the origin of $T_1(s)$, leaving two poles remaining in the vicinity of the origin (one at $s = 0$ from $G_2(s)$, and one at $s \approx 0.018$ from $T_1(s)$), thus leading to a root locus in the vicinity of the origin in Figure 10.27a which is quite similar to that seen in Figure 10.26c (again, the outer loop stabilizes those two inner-loop poles near the origin, one of which was actually unstable). Finally, recall that the slope of the Bode plot of $T_1(s)$ in Figure 10.25e for small ω is $+1$. As a result, the Bode plot of the full $G_2(s)\,T_1(s)\,D_2(s)$ system in Figure 10.27b for small ω is -1 (see solid curve), instead of -2 as it is for the idealized $G_2(s)\,D_2(s)$ (taking $T(s) \approx 1$; see dashed curve). Effectively, there is now only one pole at the origin in the full $G_2(s)\,T_1(s)\,D_2(s)$ system rather than two (as when considering the idealized inner loop, taking $T(s) \approx 1$); this detail presents no significant problems. △

[17] Recall from §10.2.1.2 that *approximate* cancellations of controller zeros/poles with plant poles/zeros are problematical in the RHP, and must never be attempted; similarly, as seen in Figure 10.22, approximate pole/zero cancellations on the imaginary axis (including near the origin) must be handled with care to ensure stability. However, the pole/zero cancellation when forming the $G_2(s)\,T_1(s)$ product in this case comes from our *representation* of the SIMO plant as the casecade of two separate SISO blocks, $G_1(s)$ and $G_2(s)$. The pole/zero cancellation arising from this representation of a single plant as a two-block cascade is exact. Some software packages (e.g., Matlab) must be used with care in order to realize this exact pole/zero cancellation in the simulation of the outer loop, as small numerical errors in the calculation of $T_1(s)$ can sometimes conceal (numerically) an exact pole/zero cancellation of this sort.

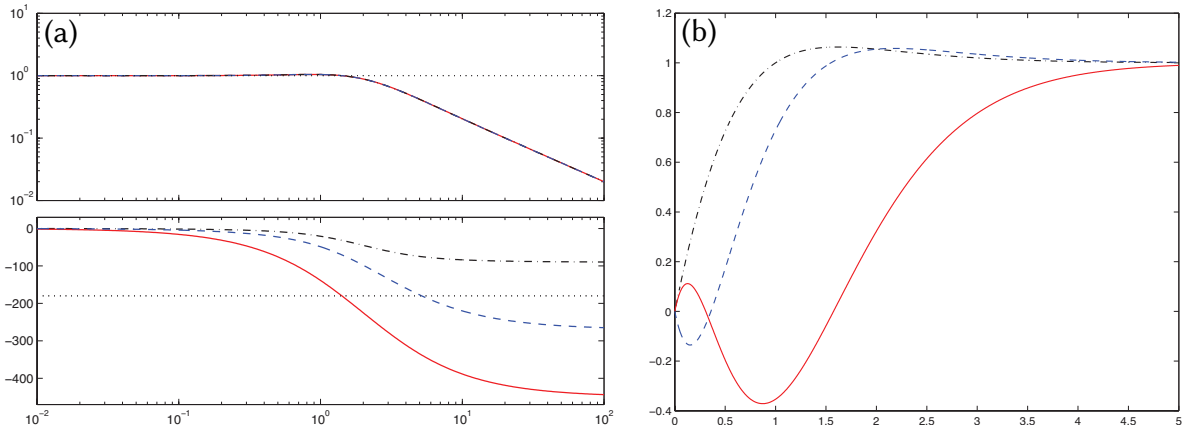


Figure 10.28: (a) Bode plot and (b) step response of three stable systems: (dot-dashed) $T_1(s)$, (dashed) $T_2(s)$, and (solid) $T_3(s)$ [see (10.22)].

10.3.4.1 Nonminimum phase systems

We conclude this section with a brief discussion of a characteristic behavior exhibited by the overall response (given by the solid curve in Figure 10.27c) to a step reference input, x_r, to the inverted pendulum system, for which the control feedback was determined using SLC in Example 10.6. In particular, note that this step response goes the wrong way (negative) before it goes the right way (positive). This behavior is not a fluke, and is characteristic of the step response in any stable (or stabilized) system with a single RHP zero. Any child who has balanced a meterstick (or, in the United States, a yardstick...) on his hand is intuitively familiar with the need to take a small step backward before walking forward with such a system.

To consider this effect in greater detail, consider the three stable systems

$$T_1(s) = 2\frac{(s+1)(s+4)}{(s+2)^3}, \quad T_2(s) = -2\frac{(s+1)(s-4)}{(s+2)^3}, \quad T_3(s) = 2\frac{(s-1)(s-4)}{(s+2)^3}. \tag{10.22}$$

All three of these systems have identical (stable) poles; it is only the sign of the zeros, and the sign of the overall gain, which distinguishes them. The Bode plots and step responses of these three systems are given in Figure 10.28. The magnitude of the Bode plot is identical in all three cases. The phase change between the left end of the Bode plot and the right end of the Bode plot is minimized by $T_1(s)$ [with no RHP zeros, called a **minimum-phase system**], and is maximized by $T_3(s)$ [with only RHP zeros, called a **maximum-phase system**]; this phase change takes some intermediate value for $T_2(s)$ [with both LHP and RHP zeros, sometimes called a **mixed-phase system**]; note also that maximum-phase systems and mixed-phase systems are commonly referred to simply as **nonminimum phase systems**[18]. Note in Figure 10.28b that the step response in the presence of one RHP zero goes the wrong way before going the right way, and the step response in the presence of two RHP zeros goes the right way, then the wrong way, then eventually the right way[19].

The characterization given above—specifically, that a stable transfer function with stable zeros is characterized by the minimum change in phase from the left side of the Bode plot to the right side of the Bode plot—extends to any transfer function for which the degree of the denominator is greater than or equal to the degree of the numerator [that is, for a **proper** CT transfer function (see §8.2.3.1), or a **causal** DT transfer function (see §8.3.3.2)].

[18] The next time you are at a conference in an elevator that goes the wrong way before it goes the right way, say "oh! nonminimum phase!" and see who gets the joke, thus identifying immediately those of your colleagues who know classical control theory...

[19] If your elevator does this, you should probably consider changing hotels.

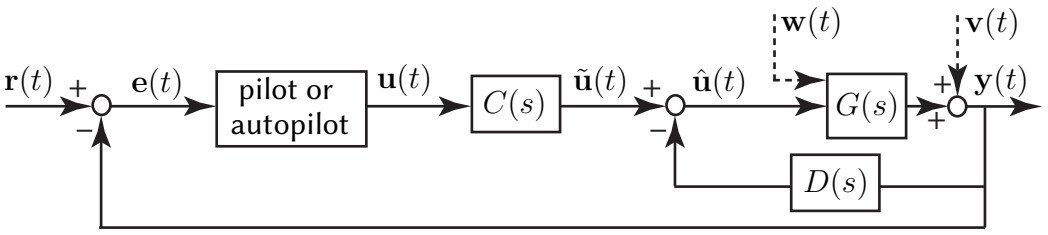


Figure 10.29: Block diagram of a **Stability and Control Augmentation System** (**SCAS**), in which the control input $\mathbf{u}(t)$ from the pilot or autopilot is augmented by small amounts of both **feedforward modification** $C(s)$ and **feedback stabilization** $D(s)$ to improve the handling qualities of an aircraft (cf. Figures 10.1 and 10.24).

10.3.5 Stability and Control Augmentation Systems (SCAS)

As introduced in Figure 10.1, we've concentrated thus far mainly on the design of controllers to be implemented in the forward part of a feedback loop in order to change substantially the input-output transfer function of a dynamic system, focusing on one or both of the following design objectives:

- tracking a reference input, $\mathbf{r}(t)$, with the output of the plant, $\mathbf{y}(t)$, for the sinusoidal components $\sin(\omega\, t)$ of the reference input $\mathbf{r}(t)$ up to a given bandwidth frequency ω_{BW}, and
- meeting specifications on rise time, settling time, and overshoot of the response $\mathbf{y}(t)$ to a step in $\mathbf{r}(t)$.

As emphasized by Guideline 10.1, one generally seeks to achieve these objectives with minimum excitation by control feedback to reduce the sensitivity of the closed-loop system to both external disturbances and internal modeling errors, especially unmodeled delays. We have also explored (in §10.3.3) the use of filters in the return portion of a feedback loop to reduce the sensitivity of the system to measurement noise.

There are certain situations, especially in the control of aircraft, in which the control objective is a bit more modest: in such situations, we don't want to change the input-output transfer function completely, but rather simply nudge the controls gently to dampen the unfavorable, oscillatory, or unstable modes of the vehicle to make it more easily or "naturally" controllable by its operator (e.g., a pilot or autopilot). Note that the various aircraft handling characteristics deemed "natural" are subjective, and have evolved over the years from the "feel" of revered classic (stable) transport and fighter aircraft, such as the DC-3 Gooney Bird and the Supermarine Spitfire. Preferred handling characteristics for military aircraft are described in detail in a number of military regulations, such as MIL-STD 1797 and MIL-SPEC 8785C (both exciting reads!); to achieve such handling qualities throughout the entire flight envelope, stability and control augmentation systems are almost always necessary in modern aircraft designs, most of which sacrifice the inherent aerodynamic stability of vintage aircraft (achieved via large horizontal and vertical stabilizers, and in some cases significant wing dihedral) for greatly improved agility, efficiency, or stealth, and are thus characterized by "poor" stability characteristics (from a handling perspective), at least in a portion of the flight envelope, before feedback is applied[20].

This section follows that on SLC, because the problem here is similar: based on an external objective or "reference input" on the system (e.g., maintain straight-and-level flight, initiate a climbing unaccelerated turn, etc.), the "outer-loop" controller (a pilot or an autopilot) gives appropriate control inputs $\mathbf{u}(t)$ to an inner loop, akin to the signal denoted $\theta_r(t)$ in the SLC paradigm as implemented in Figure 10.24. This control input from the pilot or autopilot is then augmented and applied to the system, as depicted in Figure 10.29; such a strategy is referred to as a **Stability and Control Augmentation System** (**SCAS**[21]). The use of an SCAS is, again, best illustrated by a few examples; *the examples below are based on the linearized dynamic models of aircraft developed in Example 6.11, a review of which is suggested before proceeding.*

[20] The handling characteristics of early prototypes of the **F-117 stealth fighter** were so poor, it earned the nickname Wobblin Goblin; these characteristics were largely cured by appropriately-implemented SCAS systems in the production version of the aircraft.

[21] Some authors distinguish between a Stability Augmentation System (SAS) and a Control Augmentation System (CAS), though this distinction is, perhaps, a bit superfluous.

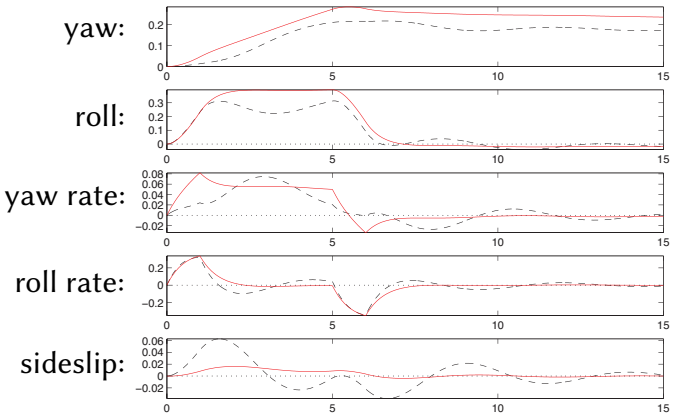


Figure 10.30: A turn with (solid) and without (dashed) opposite rudder applied to counteract **adverse yaw**.

Example 10.7 Counteracting adverse yaw

Normally, an aircraft turns by banking left or right, pitching up, then banking back to straight-and-level flight. When banked, a component of the lift force generated by the wings counters gravity, and another component of the lift force acts to turn the aircraft. The primary use of the aircraft rudder in non-emergency situations is simply to counteract the undesired sideslip (a.k.a. **adverse yaw**) generated when banking: when entering and exiting a bank, one wing produces more lift than the other, thus causing the aircraft to roll—the wing that generates more lift unfortunately also generates more drag, thus causing the aircraft to yaw towards the side generating the extra lift. In some aircraft, this effect is so strong that the pilot actually has to step on the rudder pedal every time a bank is initiated in order to maintain coördinated flight.

To illustrate, the linearized lateral/directional dynamics of a large transport aircraft on approach (at sea level, 137 knots, a flap angle of 1 degree, and a nominal cg location) may be modeled [see (6.33)] as

$$\begin{matrix}\text{yaw:}\\ \text{roll:}\\ \text{yaw rate:}\\ \text{roll rate:}\\ \text{sideslip:}\end{matrix} \quad \frac{d}{dt}\begin{pmatrix}\psi\\ \phi\\ p\\ r\\ \beta\end{pmatrix} = \begin{pmatrix}0 & 0 & 1 & 0 & 0\\ 0 & 0 & .199 & 1 & 0\\ 0 & -.002 & -.194 & -.167 & .748\\ 0 & -.003 & .636 & -2.020 & -5.374\\ 0 & .136 & -.970 & .198 & -.148\end{pmatrix}\begin{pmatrix}\psi\\ \phi\\ p\\ r\\ \beta\end{pmatrix} + \begin{pmatrix}0 & 0\\ 0 & 0\\ .053 & -.74\\ .865 & .904\\ .002 & .047\end{pmatrix}\begin{pmatrix}\delta_a\\ \delta_r\end{pmatrix}.$$

We now perform a numerical simulation of this system with $\delta_r(t) = 0$ and $\delta_a(t)$ executing a **doublet**:

$$\delta_a(t) = \begin{cases}1 & t_0 < t < t_1,\\ -1 & t_2 < t < t_3,\\ 0 & \text{otherwise},\end{cases}$$

thus entering a roll from $t_0 = 0$ s to $t_1 = 1$ s, and leaving the roll from $t_2 = 5$ s to $t_3 = 6$ s. The resulting system behaves as shown (dashed) in Figure 10.30; note the significant sideslip that accompanies the roll.

Adverse yaw is a predictable dynamic effect. One can thus implement a compensator $C(s)$ [see Figure 10.29] to apply an appropriate **feedforward** rudder correction every time the pilot commands an aileron deflection, thus improving the qualitative **handling quality** of the aircraft. To illustrate, taking $\tilde{\delta}_r(t) = \delta_r(t) - 0.1\delta_a(t)$ to add a small rudder correction with each application of the ailerons mitigates the adverse yaw in this system, as shown (solid) in Figure 10.30, reducing the peak sideslip by a factor of five, and reducing the oscillation in all of the state variables, thereby improving handling quality (see also Example 10.2 and Figure 10.16d). This relieves the busy seasoned pilot, or the n00b cadet, from having to perform such feedforward corrections. △

Example 10.8 Yaw damping of a stable "dutch roll" lateral/directional mode

Physically, the **dutch roll mode** corresponds to an oscillatory perturbation involving first a bit of roll, which creates adverse yaw towards the upward moving wing, which in turn causes a loss of lift on the upward perturbed wing, which then causes roll in the other direction, etc.; looking aft out the top of the cockpit using a periscopic sextant (which you could do in most long-haul vintage transport aircraft, like the C124 Old Shaky, in order to perform celestial navigation), the tail of the aircraft repeatedly draws an infinity sign on the horizon when in a dutch roll limit cycle. The **roll subsidence mode** is an exponentially stable (and usually relatively fast) mode that quantifies how much the aircraft continues to roll once a slight roll is initiated then the ailerons neutralized. The **spiral mode** is an exponentially stable (and usually relatively slow) mode coupling the yaw rate and the roll (but not necessarily involving sideslip, it is often a nearly coördinated motion).

Using NR_SS2TF, the state-space form of a large transport aircraft on approach [see (6.34)] may be converted to transfer function form, from rudder deflection $\delta_r(t)$ [taking the second column of B in (6.34)] to yaw rate $p(t)$ [taking $C = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \end{pmatrix}$ and $D = 0$], as

$$\frac{p(s)}{\delta_r(s)} = \frac{1.20(s - .0528)(s - 2.18)(s + 1.94)}{(s + .0679)(s + .696)(s + .403 + 2.01\mathrm{i})(s + .403 - 2.01\mathrm{i})}$$

where $p(s)$ is the Laplace transform of $p(t)$ [the yaw rate of the vehicle, in deg/s] and $\delta_r(s)$ is the Laplace transform of $\delta_r(t)$ [the rudder deflection, in deg]. At these flight conditions, two of the zeros of this open-loop system are in the RHP, and thus its dynamics are *nonminimum phase* (see §10.3.4.1). Recalling Figure 8.2, it is seen that the oscillatory poles, corresponding to the so-called **dutch roll mode**, have a natural frequency of $\omega_n = \sqrt{.403^2 + 2.01^2} = 2.05$ rad/sec, a period of $2\pi/\omega_n = 3.1$ sec, and a damping ratio of $\zeta = .403/\omega_n = .20$. The fast exponentially stable mode, corresponding to the so-called **roll subsidence mode**, has a time constant of $2\pi/.696 = 9.0$ sec. The slow exponentially stable mode, corresponding to the so-called **spiral mode**, has a time constant of $2\pi/.0679 = 93$ sec.

The dutch roll mode is destabilized at cruise speed and altitude in swept-wing commercial aircraft, and without a functioning SCAS is nearly impossible for the pilot to stabilize manually at this flight condition. As a result, all such aircraft (e.g., the Boeing 727) now have redundant SCAS systems implemented. △

Example 10.9 Damping of the "short-period" longitudinal mode

The linearized longitudinal dynamics of an F-16 Fighting Falcon in straight-and-level flight at 300 knots at sea level, when the center of mass is[22] $0.3\bar{c}$ ahead of the center of pressure (a stable configuration), is given in (6.35), and may be written in transfer function form, from elevator deflection to pitch rate, as

$$\frac{q(s)}{\delta_e(s)} = \frac{-10.5s(s + .987)(s + .0218)}{(s + .00765 + .0781\mathrm{i})(s + .00765 - .0781\mathrm{i})(s + 1.20 + 1.49\mathrm{i})(s + 1.20 - 1.49\mathrm{i})},$$

where $q(s)$ is the Laplace transform of $q(t)$ [the pitch rate of the vehicle, in deg/s] and $\delta_e(s)$ is the Laplace transform of $\delta_e(t)$ [the elevator deflection, in deg]. Note that, at these flight conditions, all the poles and zeros of this open-loop system are in the LHP. It is seen that the slow oscillatory poles, corresponding to the oscillatory **phugoid mode**, have a natural frequency of $\omega_n = \sqrt{.00765^2 + .0781^2} = .0785$ rad/sec, a period of $2\pi/\omega_n = 80$ sec, and a damping ratio of $\zeta = .00765/\omega_n = .097$. The fast oscillatory poles, corresponding to the so-called **short period mode**, have a natural frequency of $\omega_n = \sqrt{1.20^2 + 1.49^2} = 1.91$ rad/sec, a period of $2\pi/\omega_n = 3.3$ sec, and a damping ratio of $\zeta = 1.20/\omega_n = .63$. That is, at this flight condition, the phugoid mode is relatively slow and lightly damped, whereas the short period mode is relatively fast and well damped[23]; neither necessitates

[22] The **mean aerodynamic chord** $\bar{c}$ is the average distance from the leading edge to the trailing edge of the wing.

[23] Physically, the **phugoid mode** corresponds to a slow oscillatory exchange between potential and kinetic energy (climbing and slowing followed by diving and accelerating). The **short period mode** corresponds to a fast oscillation of the pitch θ and the angle of attack α (at almost constant airspeed and altitude).

modification in order to be easily controllable by the pilot at this flight condition. However, as flight conditions change (specifically, as the speed of the aircraft decreases, and/or the center of mass is shifted further aft), the short-period mode is destabilized. In such situations (for example, when a modern fighter aircraft performs a carrier landing), substantial lead compensation implemented in the $D(s)$ block in Figure 10.29 is often essential to stabilize the short period mode (via feedback-controlled substantial movement of the horizontal stabilator), thus resulting in an outer-loop system that is much more easily controlled by the pilot. △

The above three examples illustrate how classical control theory may be applied to counteract adverse yaw using feedforward control, and to dampen the dutch roll lateral mode and the short-period longitudinal mode (two common undesired modes of high-speed aircraft) using feedback control. Since modern aircraft have many redundant control surfaces and many states of interest, state-space control design tools, as developed in *NR*, are often preferred in such applications over the simple transfer-function based (aka classical) control approaches discussed here, as state-space control design techniques are inherently developed for multiple-input, multiple output (MIMO) systems, whereas transfer-function based control design techniques are primarily developed for single-input, single-output (SISO) systems. Nonetheless, the simple and robust SISO approaches discussed here are sometimes still used for these cutting-edge applications.

10.3.6 Unstable controllers for pathological SISO systems; pole placement

Up to now, we have developed just a few fundamental types of classical control components for SISO systems:

- **lead** and **lag**: $D(s) = K(s+z)/(s+p)$ with $p > z > 0$ (lead) or $z > p > 0$ (lag),
- **low-pass**: e.g., $D(s) = \omega_c^2/(s^2 + 2\,\zeta\,s\,\omega_c + \omega_c^2)$ for some damping ζ and corner frequency ω_c, and
- **notch**: $D(s) = K(s^2+z^2)/(s+p)^2$ for $z > 0$ and $p > 0$.

Note that *all of these components have their poles and zeros in the LHP*. Such components may be replicated and cascaded as appropriate, with the overall gain magnitude and sign selected as necessary to construct a controller with the desired closed-loop characteristics. In particular,

- a **lead/lag** controller is a cascade of lead and lag components,
- a **PD** controller is a special case of a lead with no rolloff of the derivative action at high frequencies,
- a **PI** controller is a special case of a lag with no rolloff of the integral action at low frequencies, and
- a **PID** controller is a cascade of PD and PI controllers.

The SLC control design paradigm extends classical control tools developed in the basic SISO setting to the SIMO case by wrapping slower SISO outer loops around faster SISO inner loops. The SCAS setting illustrates how classical control concepts may be used to improve the stability of a system while still leaving the steering of the system up to the user. The venerable lead, lag, notch, and low-pass filtering techniques summarized above may be used together to stabilize most simple systems of practical interest, shifting their closed-loop poles into the LHP. Such stabilizing controllers may then be tuned using the Bode plot via the process of **loop shaping** described previously. In certain pathological problems, however, a controller with RHP poles and/or zeros is required[24]. This situation is best illustrated by example, as done below.

[24] A controller with RHP poles is sometimes referred to as an **unstable controller**. This name, however, is something of a misnomer; a controller is designed to be used together with a plant *in closed loop*, so whether or not the controller itself has poles in the RHP is actually a matter of reduced practical consequence. It is, rather, the location of the *closed-loop poles* that ultimately matter.

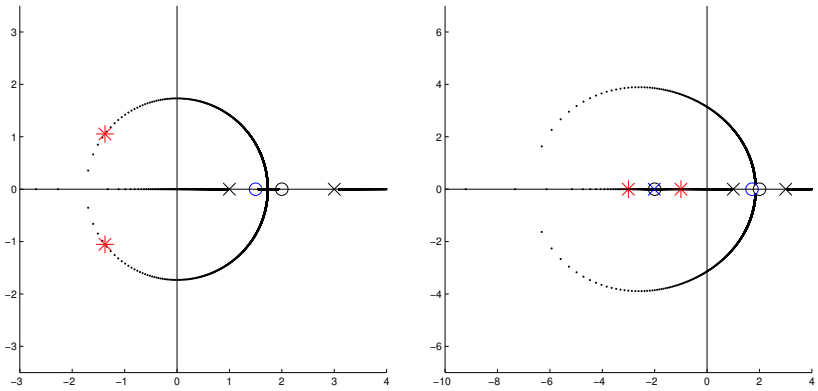


Figure 10.31: Root locus of the plant (10.23) with (a) the "lucky guess" controller design (10.24) applied, and (b) the minimal-order MESC design (10.25b) applied, denoting: ($\times$, $\circ$) plant poles and zeros; ($\times$, $\circ$) controller poles and zeros; ($*$) closed-loop poles. For clarity, the pole/zero cancellations in the LHP are not marked.

Example 10.10 Pathological pendula

Consider the SISO linear system

$$G(s) = \frac{(s+2)(s-2)}{(s+1)(s-1)(s+3)(s-3)} = \frac{s^2-4}{s^4-10s^2+9}. \tag{10.23}$$

Following the rules in §10.2.1 for plotting 180° and 0° root loci, it is clear that, if all of the poles and zeros of $D(s)$ are in the LHP, then, *regardless of the precise form of* $D(s)$, there will be a closed-loop pole on the positive real axis, either somewhere in the range $1 < s < 2$, or somewhere in the range $2 < s < 3$. Further, as illustrated in §10.2.1.2, we must never attempt to cancel unstable plant poles/zeros with controller zeros/poles. It is thus difficult to identify a stabilizing controller $D(s)$ for the plant in (10.23) using the techniques presented thus far.

A "lucky guess". To identify a stabilizing controller $D(s)$ for the plant $G(s)$ given in (10.23), we first note that designing a controller that cancels *all* of the plant poles and zeros in the LHP is not a problem (it is pole/zero cancellations in the RHP that are problematic). If the controller additionally has a zero z_3 somewhere in the range $1 < z_3 < 3$, and a negative gain is used, then, following the rules for plotting the 0° root locus in §10.2.1, the root locus depicting the possible closed-loop pole locations will start at the (uncancelled) open-loop poles at $s = 1$ and $s = 3$ and move out towards infinity on the positive and negative real axes. Depending on the precise value of z_3, these two branches of the locus will meet on the real axis somewhere in the LHP (see Figure 10.31a), at the point at infinity, or on the real axis somewhere in the RHP. Wherever these two branches meet, the locus breaks off of the real axis and loops through the complex plane back to the pair of zeros on the positive real axis. It turns out that taking $1 < z_3 < 2$ leads to the two branches of the 0° root locus meeting in the LHP, and thus the possibility of achieving a stabilizing controller if the appropriate gain is used. Taking, for example,

$$D(s) = -1.08\frac{(s+1)(s+3)(s-1.5)}{s+2} \tag{10.24}$$

leads to closed-loop stability, as illustrated by the corresponding root locus in Figure 10.31a; note that, in addition to two LHP pole/zero cancellations, the remaining two closed-loop poles are also in the LHP.

The approach of requiring an "lucky guess" to achieve stability in feedback control design borders on the RCD approach mentioned previously (see Footnote 2 on Page 10-5), and is entirely unsatisfactory (like PID, one might even be inclined disparage it to the point of calling it ad hoc...). In similar pathological problems, sufficient inspiration to achieve stability is likely unavailable. Further, the $D(s)$ suggested above is improper, with infinite high-frequency gain, and is thus not implementable, and simply cascading a low-pass filter of sufficient order in series with this controller likely forfeits stability of the closed-loop system. A more systematic approach of designing a stabilizing controller is thus required.

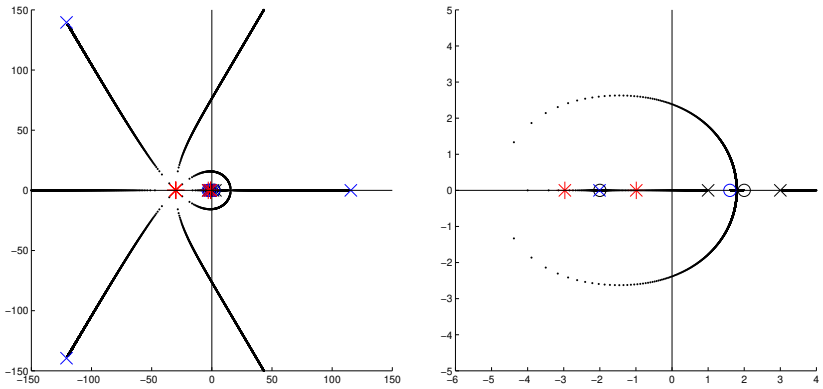


Figure 10.32: Root locus of the modified (strictly proper) MESC design (10.26) applied to (10.23): (left) complete locus, (right) close up around the origin. (×, ○) plant poles and zeros; (×, ○) controller poles and zeros; (∗) closed-loop poles. For clarity, the pole/zero cancellations at $s = -3$ and $s = -1$ are not marked.

Systematic computation of the Minimum Energy Stabilizing Controller (MESC) via Pole Placement. In *NR*, after some considerable mathematical development, we establish the following fact:

Fact 10.4 *A linear stabilizing feedback controller $D(s)$ with minimum total control energy $E = \int_0^\infty [u(t)]^2\,dt$ places the poles of a closed-loop linear system, $T(s) = G(s)D(s)/[1 + G(s)D(s)]$, at the union of the stable open-loop poles of $G(s)$, and the reflection of the unstable open-loop poles of $G(s)$ into the LHP.*

We might thus follow a so-called **pole-placement** strategy, guided by Fact 10.4, to develop an initial stabilizing controller which can be further tuned (see Fact 10.6) if necessary. For the example system in (10.23), we have

$$G(s) = \frac{b(s)}{a(s)} = \frac{s^2 - 4}{s^4 - 10\,s^2 + 9}, \quad D(s) = \frac{y(s)}{x(s)}, \quad T(s) = \frac{g(s)}{f(s)} = \frac{b(s)\,y(s)}{a(s)\,x(s) + b(s)\,y(s)}.$$

Given the plant $G(s) = b(s)/a(s)$, the problem at hand is simply the selection of $x(s)$ and $y(s)$ to solve the **polynomial Diophantine equation**[25] $f(s) = a(s)\,x(s) + b(s)\,y(s)$ for the target $f(s)$ suggested by Fact 10.4,

$$f(s) = (s+1)^2(s+3)^2. \tag{10.25a}$$

An efficient general code for solving the polynomial Diophantine equation is discussed in detail in §A.7.1, and is implemented in RR_diophantine. Using this solver[26], the "best" [lowest-order $y(s)$] solution in this case, which we dub the MESC (following Fact 10.4), is

$$D(s) = \frac{56s^3 + 128s^2 - 216s - 288}{-56s - 113} = -\frac{(s+1)(s+3)(s-1.714)}{s + 2.018}, \tag{10.25b}$$

which is remarkably close to the "lucky guess" controller proposed in (10.24). The corresponding root locus is shown in Figure 10.31b; in addition to the (stable) pole/zero cancellations, the remaining four closed-loop poles are placed at precisely $s = -1$ and $s = -3$, as specified by (10.25a).

[25] Placing the poles of a closed-loop system in certain desirable stable locations is a special case of the approach discussed in §10.3.7.

[26] You might be tempted to simply multiply out both sides of $f(s) = a(s)\,x(s) + b(s)\,y(s)$, for assumed polynomial forms of $a(s)$ and $b(s)$, and match the coefficients of like powers of s on the LHS and RHS, leading to a system of n linear equations in n unknowns, which then must be solved (typically, using a computer) for the coefficients of the polynomials $a(s)$ and $b(s)$. This approach is algebraically cumbersome, and highly prone to error; the author thus strongly advises against it. Instead, please use the convenient and easy-to-use solver provided in §A.7.1. It's FREE!

Modifying the MESC design to achieve a strictly-proper form. A systematic procedure may be followed to determine a proper or strictly-proper controller with performance that is in some sense close to that achieved by the (improper) MESC determined in (10.25b). To accomplish this, we may simply add k stable, relatively fast poles to the target $f(s)$ for the closed-loop system (it doesn't really matter exactly where we put them; effectively, they just apply some low-pass filtering to the MESC response). Note that every time we add a pole to $f(s)$ on the LHS of $f(s) = a(s)\,x(s) + b(s)\,y(s)$, we add *two* degrees of freedom to the RHS [one in $x(s)$ and one in $y(s)$]; these added degrees of freedom (i.e., for sufficiently large k) can ultimately be leveraged to reduce the relative degree n_r in the "best" solution returned by the Diophantine solver.

In the present case, we simply add $k = 4$ stable, relatively fast real poles to our target $f(s)$:

$$f(s) = (s+1)^2(s+3)^2(s+30)^4. \tag{10.26a}$$

Solving as before (using RR_diophantine) gives a modified controller that is strictly proper, but is otherwise quite similar in closed-loop behavior to the Minimum Energy Stabilizing Controller:

$$D(s) = K\,\frac{(s+1)(s+3)(s-1.604)}{(s+2.0104)(s-115.6)(s^2+2\zeta\omega_c+\omega_c^2)}. \tag{10.26b}$$

where $K = 4.08e6$, $\omega_c = 185$, and $\zeta = 0.654$. The corresponding root locus is shown in Figure 10.32; near the origin, this root locus is quite similar to that of the MESC in Figure 10.31b, with the appearance of four more stable, relatively fast poles out at $s = -30$. In addition to the (stable) pole/zero cancellations, the remaining eight closed-loop poles are placed as specified in (10.26a). Without such a systematic procedure, it would be nearly impossible to make the "lucky guess" that would lead to such a stabilizing strictly proper controller. △

10.3.7 Simple parameterizations of all stabilizing controllers†

We focus in most of §10 on the tuning of stabilizing controllers for stable and unstable plants $G(s)$. In certain cases (see, e.g., Example 10.10 above), identifying a controller $D(s)$ that results in an internally stable system (see Fact 10.1) can itself be a difficult problem, and (starting from that) having a simple parameterization of all stabilizing controllers is handy. The difficulty is related to the fact that the relationship between $D(s)$ and $T(s)$ in (10.1) is nonlinear:

$$T(s) = \frac{G(s)\,D(s)}{1+G(s)\,D(s)} \quad \Leftrightarrow \quad D(s) = \frac{1}{G(s)}\cdot\frac{T(s)}{1-T(s)}. \tag{10.27a}$$

If we instead write $T(s) = Q(s)\,G(s)$ and design $Q(s)$ corresponding to a stable $T(s)$, the control design problem is easier.

For a $G(s) = b(s)/a(s)$ with only LHP poles and zeros (that is, stable and minimum phase), once $Q(s)$ is specified, $D(s)$ is given by

$$Q(s) = \frac{D(s)}{1+G(s)\,D(s)} = \frac{p(s)}{q(s)} \quad \Rightarrow \quad D(s) = \frac{Q(s)}{1-G(s)\,Q(s)} = \frac{a(s)\,p(s)}{a(s)\,q(s)+b(s)\,p(s)}. \tag{10.27b}$$

Further, $S(s) = 1 - Q(s)\,G(s)$, $S_u(s) = Q(s)$, and $S_i(s) = [1-Q(s)\,G(s)]\,G(s)$. It follows (see Fact 10.1) that

Fact 10.5 (Youla-Kučera parametrization 1) *If $G(s)$ is proper with all LHP poles and zeros (i.e., stable and minimum phase), then the set of all proper controllers $D(s)$ that give an internally-stable closed loop may be written in the form given in (10.27b) for all rational $Q(s)$ that are stable and proper.*

However, if $G(s)$ possibly has RHP poles or RHP zeros, the construction of $D(s)$ in (10.27b), which as seen in (10.27a) simply "cancels" the entire plant $G(s)$ with the controller $D(s)$, is [by parts (e) and (f) of Fact 10.1] insufficient to ensure an internally-stable closed loop. In this case, we instead consider a controller of the form:

$$D(s) = \frac{y(s) + a(s)\,\overline{Q}(s)}{x(s) - b(s)\,\overline{Q}(s)} \tag{10.28a}$$

where the polynomials $\{x(s), y(s)\}$ solve an associated Diophantine equation, as done in Example 10.10 above:

$$a(s)\,x(s) + b(s)\,y(s) = f(s), \tag{10.28b}$$

where $\{a(s), x(s), b(s), y(s), f(s)\}$ are polynomials, $\overline{Q}(s)$ is a rational transfer function that is stable and proper but otherwise arbitrary, and $f(s)$ has all of its roots in the LHP and is of sufficiently high order that a proper $D(s) = y(s)/x(s)$ solving (10.28b) exists, but is otherwise arbitrary. The four sensitivities may now be written

$$T(s) = G(s)\,D(s)/[1 + G(s)\,D(s)] = Q(s)\,G(s) = b(s)\,[y(s) + \overline{Q}(s)\,a(s)]/f(s), \tag{10.29a}$$

$$S(s) = 1/[1 + G(s)\,D(s)] = 1 - Q(s)\,G(s) = a(s)\,[x(s) - \overline{Q}(s)\,b(s)]/f(s), \tag{10.29b}$$

$$S_u(s) = D(s)/[1 + G(s)\,D(s)] = Q(s) = a(s)\,[y(s) + \overline{Q}(s)\,a(s)]/f(s), \tag{10.29c}$$

$$S_i(s) = G(s)/[1 + G(s)\,D(s)] = [1 - Q(s)\,G(s)]\,G(s) = b(s)\,[x(s) - \overline{Q}(s)\,b(s)]/f(s); \tag{10.29d}$$

all four of these forms are stable as long as $\overline{Q}(s)$ is stable and $f(s)$ has its roots in the LHP. Note the simple relationship between $Q(s)$ and $\overline{Q}(s)$ in (10.29c), that $S_i(s) = G(s)\,S(s)$, that $T(s) = G(s)\,S_u(s)$, and that

- the factor of $a(s)$ is in the numerator of the expressions for $S(s)$ and $S_u(s)$, which implies that the poles of $G(s)$ appear as zeros of $S(s)$ and $S_u(s)$, thus satisfying parts (a) and (c) of Fact 10.1, and
- the factor of $b(s)$ is in the numerator of the expressions for $T(s)$ and $S_i(s)$, which implies that the zeros of $G(s)$ appear as zeros of $T(s)$ and $S_i(s)$, thus satisfying parts (b) and (d) of Fact 10.1.

It follows that:

Fact 10.6 (Youla-Kučera parametrization 2) *If $G(s)$ is proper but possibly has RHP poles and/or zeros, the set of all proper controllers $D(s)$ that give an internally-stable closed loop may be written in the form given in (10.28a) for some $\{x(s), y(s)\}$ that solves (10.28b) [for some $f(s)$ with all of its roots in the LHP, and of sufficiently high order that (10.28b) is solvable with a proper $D(s) = y(s)/x(s)$] and for all rational $\overline{Q}(s)$ that are stable and proper.*

Note that the expression for $D(s)$ in (10.27b) is a special case of that in (10.28a) that is applicable for $G(s)$ with only LHP poles and zeros; indeed, taking $f(s) = a(s)$ in this case results in $x(s) = 1$ and $y(s) = 0$ and $Q(s) = \overline{Q}(s)\,a(s)$, thus reducing (10.28a) to (10.27b).

The entire discussion above extends immediately to DT problems simply by replacing s with z, replacing the phrase "LHP" with "inside the unit circle", and replacing the phrase "RHP" with "outside the unit circle".

10.3.8 Implementation of CT linear controllers in analog electronics†

In order to achieve the **loop shaping** described previously (see §10.2.2 and Figure 10.7), it is straightforward to implement a CT controller $D(s)$ that cascades together a number of the individual filters discussed above and developed as simple analog circuits with low output impedance[27] in §9.2. In most cases[28], this involves

- lag filter(s) [see Figure 10.17a] to improve tracking, implemented at frequencies well *below* the crossover frequency ω_g so as to not substantially erode the PM,
- lead filter(s) [see Figure 10.17b] to increase the PM and thus reduce overshoot, centered at the crossover frequency ω_g (that is, taking $\sqrt{p/z} = \omega_g$) so as to maximize their beneficial effect,
- low-pass filter(s) [see Figures 8.8a, 8.8b, 8.10a, and 8.10b] to improve robustness, implemented at frequencies well *above* the crossover frequency ω_g so as to not substantially erode the PM, and/or
- notch filter(s) [see Figures 10.17c and 10.21] to knock-out characteristic system oscillations, implemented carefully near the frequencies of the oscillatory plant poles to avoid closed-loop instability.

[Note that one of the more delicate matters to attend to in classical control design is effectively "squeezing" the lag and low-pass filtering of $D(s)$ (situated on the Bode plot below and above ω_g, respectively) in as close as possible to ω_g while still achieving the required PM, in order to maximize the ranges of frequencies over which these filters have their beneficial effects.] The following results from §9.2 are of particular importance:

- a general adder/subtractor circuit is developed in Example 9.28 and illustrated in Figure 9.31c,
- lead, lag, P, I, D, PI, PD, and first-order low-pass filters are all special cases of the single op-amp circuit developed in Example 9.30 and illustrated in Figure 9.32a,
- second-order and fourth-order low-pass filter circuits are developed in Exercise 9.5, and
- a notch filter circuit is developed in Example 9.31 and illustrated in Figures 9.32b and 9.33.

Due to the issue of aliasing (see Figures 8.6-8.7), low-pass filters used to reject disturbances generally need to be implemented in CT using analog electronics. [That is, after you sample a signal with high-frequency noise, it is too late to distinguish the low-frequency signal from the (sampled, aliased) high-frequency noise.] However, due to their low cost and ease of programming (and reprogramming when the system changes), DT microcontrollers are, today, often best suited for implementing the rest of the controller (as DT difference equations) even when controlling CT plants, as discussed at length in §10.4.

10.3.9 Extending the PID, lag, lead, low-pass, and notch techniques to DT systems

As discussed in §10.2.5, the root locus, Bode, and Nyquist techniques extend immediately to DT systems. These tools may thus be used to tune DT linear controllers $D(z)$ formed as a cascade of lag, lead, low-pass, and notch filters in an essentially identical manner.

10.3.10 Implementation of DT linear controllers in microcontrollers

Via inverse Z transform of $D(z)$ and writing the resulting difference equation, such as (8.24a), in a convenient form, such as (8.24c), it is easy to see how the resulting difference equation may be implemented in DT in a microcontroller. As discussed in §8.3.3.2, for its corresponding difference equation to be implementable, the DT controller $D(z)$ must be **causal** (that is, $n \geq m$, where n is the order of the denominator and m is the order of the numerator). **Strictly causal** $D(z)$, with $n > m$, are somewhat easier to implement in practice than those arising from **semi-causal** $D(z)$, with $n = m$, as they give a full timestep to compute the next value of u_k.

[27] The low output impedance of each op amp circuit mentioned here (cf. the passive circuits of Example 9.2) simplifies the circuit design process significantly, as it effectively decouples each individual stage of the cascade, allowing them to be designed separately.

[28] Notable pathological exceptions requiring something different are discussed in §10.3.6-10.3.7.

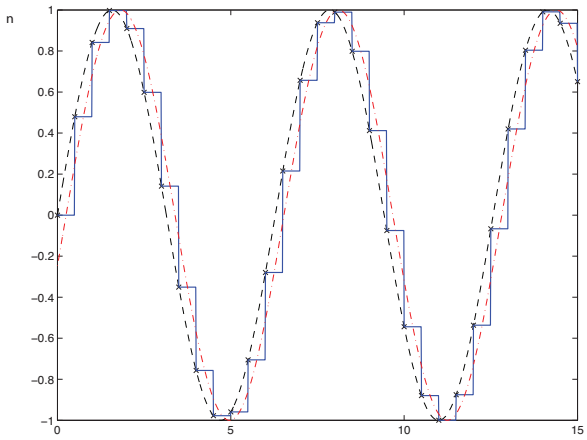


Figure 10.33: The principal effect of a zero-order hold (ZOH) of a DAC applied to a sampled sinusoid is the introduction of an $h/2$ delay in the signal. This is indicated here by (solid) the ZOH reconstruction of the (dashed) CT sinusoidal input from the samples (denoted by the $\times$ symbols). The dot-dashed curve indicates this ZOH reconstruction after the removal of, effectively, a small "sawtooth" wave at twice the Nyquist frequency $\omega_{\text{Nyquist}} = \pi/h$, which is relatively unimportant, illustrating clearly this (detrimental) $h/2$ delay.

10.4 Classical control design of DT controllers for CT plants

When controlling a CT physical system $G(s)$ with a DT controller $D(z)$ implemented in the digital logic of a microcontroller, an interesting mix of CT and DT components arises. Such mixed systems may be analyzed using one of two methods, as discussed in §10.4.1 and §10.4.2.

Note that the interconnection of CT plants with DT controllers requires both **digital-to-analog converters** (**DACs**) to convert the DT output from the digital electronics $D(z)$ [that is, from the microcontroller] into a CT input to the plant $G(s)$, and **analog-to-digital converters** (**ADCs**) to convert the CT output from the plant $G(s)$ into a DT input to the controller $D(z)$. To keep production costs down, nearly all commercial DACs incorporate a **zero-order hold** (**ZOH**): that is, the output $u(t)$ of the DAC at any time t is taken simply as the input u_k to the DAC at the most recent timestep t_k. We thus assume that all DACs incorporate a ZOH strategy in the remainder of this text. The influence of the ZOH of the DAC is significant (see Figures 10.33 and 10.12a) and detrimental, as it can significantly reduce the PM (and, thus, increase overshoot) of a closed-loop system if $\omega_{\text{Nyquist}} = \pi/h$ is not taken sufficiently high (that is, if h is not taken sufficiently small).

Nearly all ADCs incorporate an analog circuit implementing some kind of CT low-pass filter, to substantially reduce components of the input signal above ω_{Nyquist}, to prevent **aliasing** when sampling (see Figure 8.7a); depending on the intensity of the noise in your problem, you may actually want to filter even more. Recall that low pass filters typically erode the phase of the output well below their corner frequency (see §8.5). If the sample period h is sufficiently small, the influence of this low-pass filter on the phase of the open-loop system will be relatively small (but should still be accounted for - see §10.3.3). At the same time, we should not make h too small, as that would make the tap delays of the signal too close together, amplifying error in the calculation of DT difference equations. The choice of h thus reflects a delicate compromise.

In addition to discretizing CT signals in time, using a timestep h, ADCs convert real values (usually, voltages) to **finite-precision representations** in the microcontroller (MCU). If **fixed-point arithmetic** is being used (which is often required when implementing on an inexpensive PIC, AVR, or ARM microcontroller), the resulting **discretization errors** are well modeled as a bit of additive measurement noise[29].

[29] If **floating-point arithmetic** is being used, the discretization errors are better represented by a (more cumbersome) multiplicative noise model. However, as cheap 32-bit MCUs have largely eclipsed vintage 8-bit MCUs, modern microcontrollers implementing floating-point arithmetic mostly use half precision arithmetic or better (see §1.1.4), in which case discretization errors are negligible.

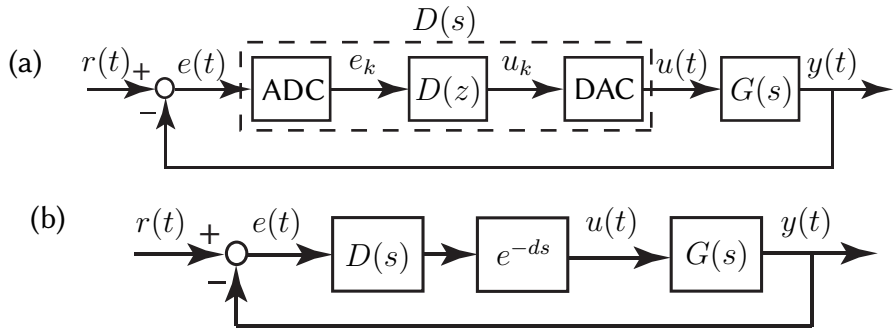


Figure 10.34: (a) A closed-loop configuration of a DT controller $D(z)$ and a CT plant $G(s)$ interpreted as a **discrete equivalent design**. When designing the CT transfer function $D(s)$ corresponding to the cascade of the ADC, the DT controller $D(z)$, and the DAC, the primary effect of the ADC & DAC is an $h/2$ delay due to the ZOH, as illustrated in Figure 10.33. (b) An equivalent circuit used for the design of $D(s)$, incorporating the $d = h/2$ delay using an appropriate Padé approximation [see (8.8)]; once $D(s)$ is designed in this manner, it may be converted into a DT transfer function with similar dynamics, $D(z)$, using Tusin's approximation with prewarping, then inverse transformed into a difference equation and implemented in a microcontroller.

10.4.1 Discrete equivalent design

In the **discrete equivalent design** approach, we consider the inputs and outputs of the closed-loop system in continuous time, and represent the cascade of the ADC (with sample period h), the DT controller $D(z)$, and the DAC (with a ZOH) as a single CT system, which we denote $D(s)$, as highlighted by the dashed box in Figure 10.34a. To proceed following this approach, we first design an appropriate CT controller $D(s)$ directly for the CT system $G(s)$ using the techniques introduced in §10.2 and explored at length in §10.3, then approximate the dynamics of this CT controller $D(s)$ with a rational expression in DT, $D(z)$. This discrete approximation is best computed via Tustin's approximation with prewarping (see §8.3.4), determining $D(z)$ from $D(s)$ via the following simple substitution:

$$D(z) = D(s)\Big|_{s=\frac{2}{fh}\frac{z-1}{z+1}} \quad \text{where} \quad f = \frac{2[1-\cos(\overline{\omega}h)]}{\overline{\omega}h\sin(\overline{\omega}h)}, \tag{10.30}$$

where $\overline{\omega}$ denotes the frequency of primary interest in the controller for which an accurate mapping is desired—that is, the notch frequency if $D(s)$ has a notch (see §10.3.2), or the crossover frequency (see §10.2.2) of the closed-loop system if $D(s)$ does not have a notch. Once $D(z)$ is obtained via this approach, the corresponding difference equation is easily determined via the inverse Z transform techniques presented in §8.3. This difference equation may then be implemented in digital electronics.

The most significant detrimental effect encountered when implementing a CT controller, $D(s)$, in a DT fashion on a microcontroller, as illustrated by the dashed box in Figure 10.34a, is the *effective* $h/2$ *delay resulting from the ZOH of the DAC*, where h is the sample period, as illustrated in Figure 10.33. This delay is *not* accounted for in (10.30). At frequencies well below the Nyquist frequency, the effect of this delay is negligible. At input frequencies within about an order of magnitude of the Nyquist frequency, however, the effect of this delay is a significant *phase loss*:

$$\text{phase loss} = 2\pi\,\frac{\text{time delay}}{\text{wave period}} = 2\pi\frac{h/2}{2\pi/\omega} = h\,\omega/2 \ \text{ rad}, \tag{10.31}$$

as illustrated in Figure 10.12a. *A corresponding amount of extra phase lead at the crossover frequency ω_g should thus be built in to the CT control design $D(s)$ to compensate.* Alternatively, the effect of this $d = h/2$ delay in the ultimate DT implementation of the controller may be accounted for during the design of $D(s)$ simply by including a Padé approximation of the delay in series with $G(s)$, as illustrated in Figure 10.34b.

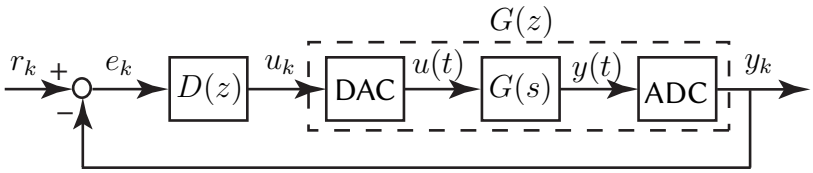


Figure 10.35: A closed-loop configuration of a DT controller $D(z)$ and a CT plant $G(s)$ interpreted as a **direct digital design**. The DT transfer function $G(z)$, given by the cascade of a DAC, a CT plant $G(s)$, and an ADC, is computed *exactly* in (8.26).

10.4.2 Direct digital design

In the **direct digital design** approach, we consider the inputs and outputs of the closed-loop system in DT, and represent the cascade of the DAC (with a ZOH), the CT plant $G(s)$, and the ADC (with sample period h) as a single DT system, which we denote $G(z)$, as highlighted by the dashed box in Figure 10.35, using the exact expression provided in (8.26); note that, as opposed to (10.30), this expression *does* account for the ZOH of the DAC. We then determine the DT controller $D(z)$ directly for this DT system $G(z)$, as introduced in §10.2.5 and discussed further in §10.3.9. If a CT controller $D(s)$ is known that is good for the CT plant $G(s)$, this controller may be approximated as an initial DT $D(z)$ with similar dynamics using Tustin's approximation with prewarping [see (8.3.4.2)], then tuned further directly in DT.

The primary benefit of the direct digital design approach is that it is *exact*. The primary drawback of this design approach is that it only considers the values of the output y_k *at* the timesteps, and thus doesn't detect whether or not there are actually significant oscillations in $y(t)$ *between* the timesteps, referred to as **intersample ripple**. *To avoid such intersample ripple, one should generally avoid placing the poles of the DT closed-loop system anywhere near the $z = -1$ point during the DT controller design.*

Example 10.11 Evidence of time delay from the ZOH when designing a DT controller for a CT plant

To illustrate that an $h/2$ time delay is in fact the leading-order (and detrimental) effect of the ZOH in the DAC when, following the **discrete equivalent design** paradigm in which a CT controller $D(s)$ is converted to DT with sample period h, consider the following unstable CT plant and corresponding CT lead controller acting in closed loop:

$$G(s) = \frac{1}{(s+10)(s-10)}, \quad D(s) = K\frac{s+z}{s+p}.$$

Taking $z = 10$ (for a stable pole/zero cancellation) and $p = 20$ (for a little bit of phase lead) leads to a root locus for the CT problem as shown in Figure 10.36a; further, taking $K = 380$ leads to the (stable) closed-loop poles indicated by $\ast$ in the root locus, gives crossover near the peak of the phase lead in the corresponding Bode plot, and gives about a 28% overshoot and a 0.1 second rise time in the corresponding step response. Looking at this root locus plot, it appears that turning up the control gain K to larger values would lead to lightly damped poles, but would apparently *not* lead to closed-loop instability.

Now consider the implementation of this controller in DT taking $h = .02$ seconds (that is, taking a sample frequency of 50 Hz), using a ZOH in the DAC. We will approximate our well-behaved CT control design $D(s)$ as a DT difference equation with similar dynamics using Tustin's approximation with prewarping, as reviewed in (10.30). To analyze in discrete time how well this controller works (see Figure 10.35), we may convert the DAC – $G(s)$ – ADC cascade to the exact expression for the corresponding $G(z)$ via (8.26) and consider the problem in the **direct digital design** setting; the corresponding DT root locus plot is shown in Figure 10.36b. Note that the DT closed-loop system goes unstable if the gain K exceeds a critical value.

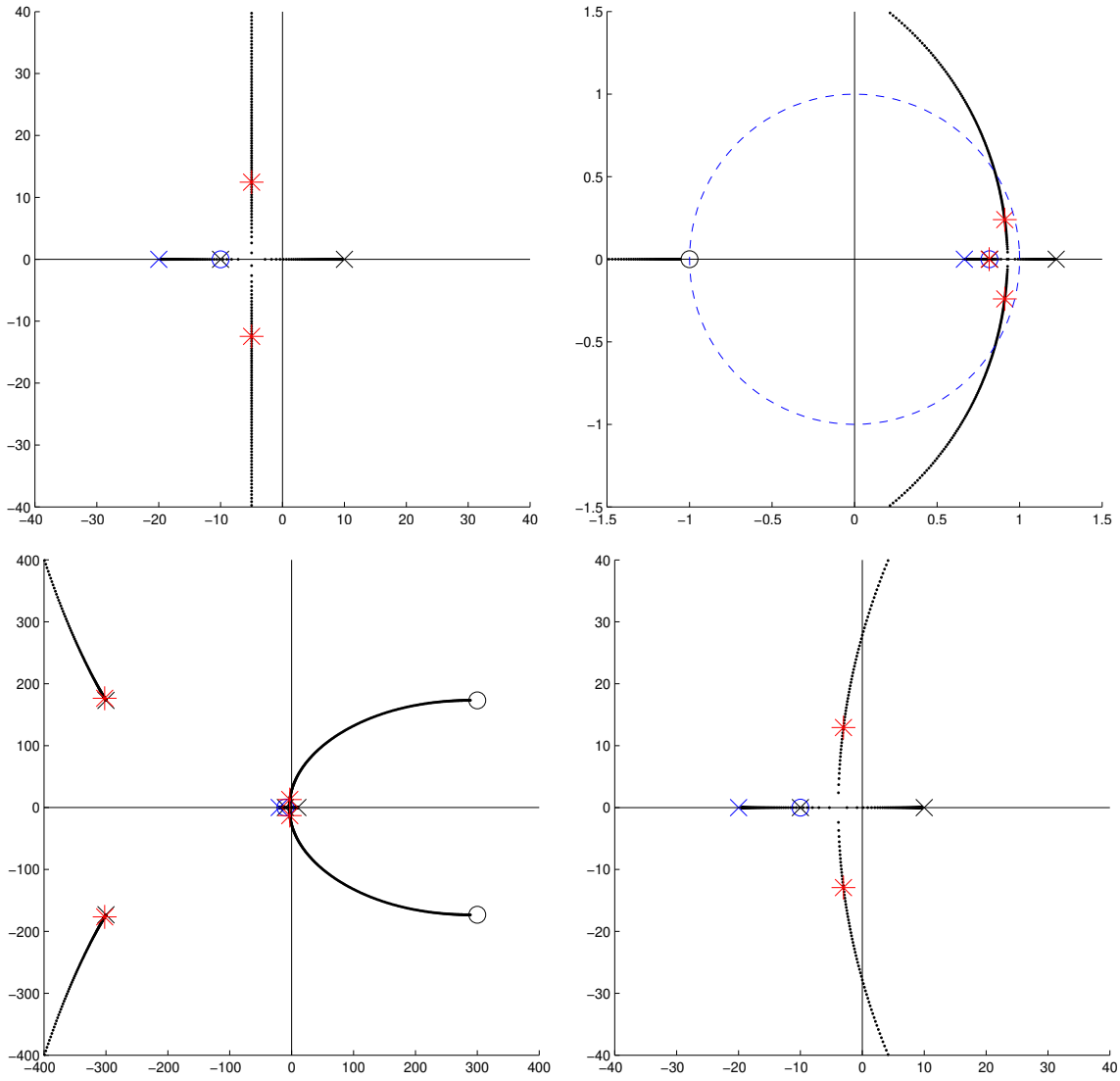


Figure 10.36: The root locus plots discussed in Example 10.11, illustrating the high-gain instability caused by the effective $h/2$ time delay caused by the ZOH of the DAC.

The reason why the DT root locus in Figure 10.36b goes unstable for large K, but the CT root locus in Figure 10.36a does not, is well explained by the effective $h/2$ delay caused by the ZOH in the DAC. Indeed, if we again analyze the system in CT, but now approximate the CT plant as $G(s) \cdot F_{2,2}(s)$, incorporating an $n = m = 2$ rational approximation [see (8.8)] of the Laplace transform of a delay e^{-ds}, with $d = h/2$, the corresponding root locus is shown in Figure 10.36c, a closeup of which (near the origin) is given in Figure 10.36d. It is seen that the delay is approximated by n poles in the LHP and m zeros in the RHP, and that the speed of these LHP poles and RHP zeros is inversely proportional to the delay d [that is, for small d, these LHP poles and RHP zeros are **fast** (i.e., far from the origin)]. The CT root locus ultimately connects to the fast RHP zeros, so the CT closed-loop system now goes unstable if the gain K exceeds a certain critical value. The value of K for which this CT model of the problem goes unstable coincides accurately with the value of K for which the corresponding DT case goes unstable. △

10.4.3 Deadbeat control: pole placement at the origin for DT settling in finite time

Deadbeat control is a special DT control technique that has no direct analog in CT. Recall from Figure 8.4 that the CT design guide for exponential settling of a step response with characteristic timescale t_s is given by placing all closed-loop poles in the s-plane such that their real parts are to the left of $s = -\sigma = -4.6/t_s$. Recall from Figure 8.7 that the corresponding DT design guide is given by placing all closed-loop poles in the z-plane inside a circle centered at the origin with radius $r = e^{-\sigma\, h} = e^{-4.6\, h/t_s}$. For rapid settling, then, we want closed-loop poles with large negative real parts in CT, and with radius much less than one in DT.

It is thus reasonable to consider a DT control design strategy that uses pole placement to design a causal $D(z) = y(z)/x(z)$ for a causal plant $G(z) = b(z)/a(z)$ that *puts all of the closed-loop poles at the origin*. This results in a DT closed-loop system for which the *output settles completely after a finite number of timesteps*; in the language of §8.3.3, the entire DT closed-loop system becomes an FIR filter instead of an IIR filter. Following this approach, the closed-loop transfer function must be

$$T(z) = \frac{G(z)\, D(z)}{1 + G(z)\, D(z)} = \frac{b(z)\, y(z)}{a(z)\, x(z) + b(z)\, y(z)} = \frac{g(z)}{z^\ell}. \tag{10.32}$$

There is flexibility both in the choice of ℓ [i.e., how many steps are taken until the closed-loop system output settles completely; ℓ must be large enough that the resulting controller $D(z)$ is causal] and in the choice of $g(z)$ [i.e., the (finite-time) dynamics expressed by the output of the closed-loop system as it settles].

Denote $deg\,\{p(z)\}$ the order of $p(z)$. For $D(z)$ to be causal, $deg\,\{x(z)\} \geq deg\,\{y(z)\}$, and thus, by (10.32),

$$\ell \geq deg\,\{g(z)\} + deg\,\{a(z)\} - deg\,\{b(z)\}. \tag{10.33a}$$

For $D(z)$ to be strictly causal [which is perhaps easier to implement in a microcontroller running at a modest clock speed, as seen in (8.24c)], $deg\,\{x(z)\} > deg\,\{y(z)\}$, strict inequality is required in the above expression. Further, if $T(z) = Y(z)/R(z)$ has zero steady-state error to a unit step [that is, if for $r_k = 1$ for $k = 0, 1, 2, \ldots$ and thus $R(z) = z/(z-1)$ it follows that $\lim_{k\to\infty} y_k = 1$], then by the DT final value theorem (Fact 8.4)

$$\lim_{k\to\infty} y_k = \lim_{z\to 1}(z-1)\, Y(z) = \lim_{z\to 1}(z-1)\, T(z)\, \frac{z}{z-1} = T(1) = 1 \quad \Rightarrow \quad g(1) = 1. \tag{10.33b}$$

It is thus clear from (10.32) that either $D(z)$ or $G(z)$ has a pole at $z = 1$.

Minimal-time deadbeat controllers for stable, minimum-phase systems

If $G(z) = b(z)/a(z)$ is strictly causal and both stable and minimum phase [that is, if all of the poles and zeros of $G(z)$ are inside the unit circle], we may simply take $g(z) = 1$ and $\ell = deg\,\{a(z)\} - deg\,\{b(z)\}$ in (10.32); this choice is referred to as the **minimal-time deadbeat controller**, as it results in the fastest-possible complete settling of the output of the DT system. In this case, we may implement the parameterization in (10.27a):

$$T(z) = \frac{1}{z^\ell} \quad \Leftrightarrow \quad D(z) = \frac{1}{G(z)}\, \frac{T(z)}{1 - T(z)} = \frac{a(z)}{b(z)}\, \frac{1}{z^\ell - 1} = \frac{y(z)}{x(z)}. \tag{10.34a}$$

Note that $D(z)$ in this case cancels both the poles and zeros of $G(z)$; this strategy provides internal stability only if $G(z)$ has all its poles and zeros inside the unit circle (see also the discussion in §10.2.1.2, noting that the plant $G(z)$ is generally only known approximately). As discussed in §10.1, the transfer function from the reference r_k to the control u_k, referred to as the control sensitivity, is given in this case by

$$S_u(z) = \frac{U(z)}{R(z)} = \frac{D(z)}{1 + G(z)\, D(z)} = \frac{y(z)\, a(z)}{a(z)\, x(z) + b(z)\, y(z)} = \frac{a(z)\, a(z)}{a(z)\, b(z)\, (z^\ell - 1) + b(z)\, a(z)} = \frac{a(z)}{b(z)\, z^\ell}. \tag{10.34b}$$

Due to the pole/zero cancellations in $[G(z)\,D(z)]$, *the poles of* $S_u(z)$ *are different than the poles of* $T(z)$. [This is not normally the case; when pole/zero cancellations in $[G(z)\,D(z)]$ do not occur, the denominator of both $T(z)$ and $S_u(z)$ is simply $f(z) = a(z)\,x(z) + b(z)\,y(z)$.] As a result, *even though the output* y_k *settles completely after a finite number of timesteps, the control* u_k *does not.* This presents a significant problem: when the DT system considered arises as a result of the application of DT microcontroller to a CT system, the **control oscillations** in u_k caused by this approach often lead to a significant **intersample ripple** in the CT output $y(t)$, even well after the DT *samples* of the output, y_k, settle completely (see Example 10.12). This largely defeats the entire purpose of implementing the deadbeat control design methodology in the first place.

Ripple-free deadbeat controllers for stable, possibly nonminimum-phase systems

Designing a $D(z)$ that cancels the entire dynamics of the plant, as suggested in (10.34), is a heavy-handed approach. A much better approach, called a **ripple-free deadbeat controller**, strives to make both y_k *and* u_k settle after a finite number of timesteps. To see how to do this, assuming that $G(z) = b(z)/a(z)$ is stable but not necessarily minimum-phase (so, all poles of $G(z)$ must be inside the unit circle, but its zeros need not be), we may take $g(z) = b(z)/b(1)$ and $\ell = \textit{deg}\,\{a(z)\}$ in (10.32), and thus [cf. (10.34)]

$$T(z) = \frac{b(z)/b(1)}{z^\ell} \quad \Leftrightarrow \quad D(z) = \frac{1}{G(z)}\,\frac{T(z)}{1-T(z)} = \frac{a(z)/b(1)}{z^\ell - b(z)/b(1)} = \frac{y(z)}{x(z)}, \tag{10.35a}$$

from which it follows that

$$S_u(z) = \frac{U(z)}{R(z)} = \frac{D(z)}{1+G(z)\,D(z)} = \frac{a(z)/b(1)}{[z^\ell - b(z)/b(1)] + b(z)/b(1)} = \frac{a(z)/b(1)}{z^\ell}. \tag{10.35b}$$

In contrast with the minimal-time deadbeat controller (10.34), the ripple-free deadbeat controller (10.35) allows the zeros of $G(z)$ to appear in the numerator of $T(z)$, and by so doing they do *not* appear in the denominator of $S_u(z)$. Thus, *the output* y_k *and the control* u_k *both settle completely* ℓ *timesteps after a step input in* r_k, and the intersample ripple problem mentioned previously is eliminated (see Example 10.13). As the timestep h is made small, deadbeat controllers demand large control inputs; one must thus be careful when following this approach. Guideline 10.1 still applies; to follow it, just don't let the timestep h get too small.

Ripple-free deadbeat controllers for general (possibly unstable and/or nonminimum-phase) systems

For general (possibly unstable and nonminimum-phase) causal DT systems, we may implement the parameterization given (10.28) [see Fact 10.6] which as involves developing $D(z)$ according to the solution $\{x(z), y(z)\}$ of an associated **Diophantine equation** (see §A.7.1)

$$D(z) = \frac{y(z) + a(z)\,\overline{Q}(z)}{x(z) - b(z)\,\overline{Q}(z)} \quad \text{where} \quad a(z)\,x(z) + b(z)\,y(z) = z^\ell, \tag{10.36a}$$

where in the present deadbeat setting we take $f(z) = z^\ell$ and[30] $\overline{Q}(z) = q(z)/z^k$ where $k \geq 0$ and $\textit{deg}\,\{q(z)\} \leq k$ [$q(z)$ is otherwise arbitrary]. Assuming no pole/zero cancellations[31] in $[G(z)D(z)]$, $\ell \geq 2n-1$. Amongst these, the case with $q(z) = 0$ (and thus $\overline{Q}(z) = 0$) and $\ell = 2n-1$ is particularly simple (see Example 10.14), and results in

$$T(z) = \frac{G(z)\,D(z)}{1+G(z)\,D(z)} = \frac{b(z)\,y(z)}{z^\ell} \quad \text{and} \quad S_u(z) = \frac{D(z)}{1+G(z)\,D(z)} = \frac{a(z)\,y(z)}{z^\ell}. \tag{10.36b}$$

[30] To see clearly why this form for $\overline{Q}(z)$ works, see (10.29).

[31] Note that (10.36b) has a minimum of $\ell+1$ powers of z whose coefficients must be matched between the LHS and the RHS. Take $n = \textit{deg}\,\{a(z)\}$ and $m = \textit{deg}\,\{b(z)\}$; since $G(z) = b(z)/a(z)$ is causal, $n \geq m$. Take $i = \textit{deg}\,\{x(z)\}$ and $j = \textit{deg}\,\{y(z)\}$; since $D(z) = y(z)/x(z)$ is causal, $i \geq j$. It follows from (10.36b), assuming no pole/zero cancellations in $[G(z)D(z)]$, that $n+i = \ell$. There are at most $2(i+1)$ free coefficients in $D(z)$. Setting $2(i+1) \geq \ell+1$ for the resulting system to be solvable, it follows that $\ell \geq 2n-1$.

Example 10.12 Minimal-time deadbeat control of a stable plant

Consider the direct digital design setting (Figure 10.35) applied to the stable minimum-phase CT system

$$G(s) = \frac{b(s)}{a(s)} = \frac{1}{(s+1)(s+5)}. \tag{10.37a}$$

Taking $h = 0.2$ and (exactly) converting the DAC – $G(s)$ – ADC cascade to DT [see (8.26)] gives the corresponding stable minimum-phase (with all poles and zeros inside the unit circle) DT transfer function

$$G(z) = \frac{b(z)}{a(z)} = \frac{0.0137\,z + 0.0092}{z^2 - 1.1866\,z + 0.3012}, \quad \text{with} \quad n = deg\{a(z)\} = 2 \quad \text{and} \quad m = deg\{b(z)\} = 1. \tag{10.37b}$$

Applying (10.34) for minimal-time deadbeat control of the stable minimum-phase system (10.37) results in $\ell = n - m = 1$ and

$$D(z) = \frac{a(z)}{b(z)} \cdot \frac{1}{z^\ell - 1} = \frac{z^2 - 1.1866\,z + 0.3012}{0.0137\,z + 0.0092} \cdot \frac{1}{z - 1}; \tag{10.38}$$

the DT and CT responses u_k, $u(t)$, y_k, and $y(t)$ resulting from a unit step input are illustrated in Figure 10.37a; note the significant intersample ripple. △

Example 10.13 Ripple-free deadbeat control of a stable plant

Applying (10.35) for ripple-free deadbeat control of the system (10.37) results in $\ell = n = 2$ and

$$D(z) = \frac{a(z)}{z^\ell - b(z)} = 43.6361 \frac{z^2 - 1.1866\,z + 0.3012}{z^2 - 0.5983\,z - 0.4017}; \tag{10.39}$$

the DT and CT responses u_k, $u(t)$, y_k, and $y(t)$ resulting from a unit step input in this case are illustrated in Figure 10.37b; note that y_k takes one more timestep to settle than the minimal-time deadbeat controller considered in Example 10.12, but the significant intersample ripple is eliminated. △

Example 10.14 Ripple-free deadbeat control of an unstable plant

Consider the direct digital design setting applied to the unstable CT system [cf. (10.37)]

$$G(s) = \frac{b(s)}{a(s)} = \frac{1}{(s-1)(s-5)}. \tag{10.40a}$$

Taking $h = 0.2$ and converting the DAC – $G(s)$ – ADC cascade to DT [see (8.26)] gives the corresponding unstable nonminimum-phase (with two poles and one zero outside the unit circle) DT transfer function

$$G(z) = \frac{b(z)}{a(z)} = \frac{0.0306\,z + 0.0455}{z^2 - 3.9397\,z + 3.3201}, \quad \text{with} \quad n = deg\{a(z)\} = 2 \quad \text{and} \quad m = deg\{b(z)\} = 1. \tag{10.40b}$$

Though the system considered in (10.40) has the same order as that considered in (10.37), neither (10.34) nor (10.35) may be applied in this case, as these approaches are based on pole-zero cancellations between the controller and the plant; since the DT plant, which is only known approximately, has both poles and zeros outside the unit circle, these approaches would certainly fail (see §10.2.1.2).

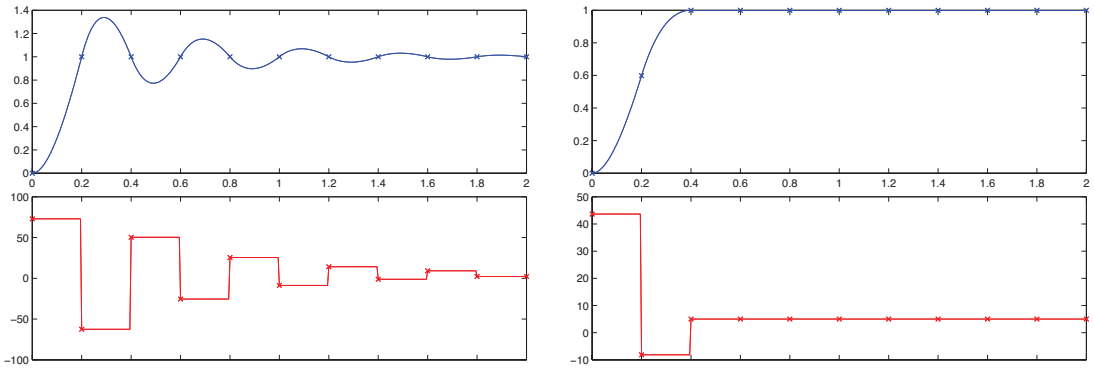


Figure 10.37: (a) Step response of the stable system (10.37) when the **minimal-time deadbeat controller** (10.34) is applied (see Example 10.12). Though the output of the DT system settles completely after $\ell = n - m = 1$ timestep, the output of the underlying CT system exhibits significant **intersample ripples** well after that, due to the control oscillations shown in the bottom subfigure. (b) Step response of the same system with the **ripple-free deadbeat controller** (10.35) applied (see Example 10.13). The output of both the DT and the underlying CT system settle completely after $\ell = n = 2$ timesteps, eliminating the intersample ripple seen previously.

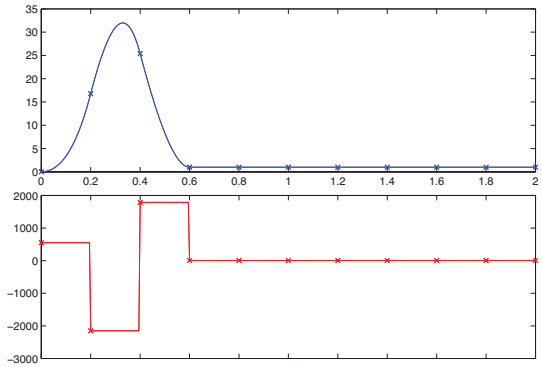


Figure 10.38: Step response of the unstable system (10.40) with the general ripple-free deadbeat controller (10.36) applied, taking $\overline{Q}(z) = 0$ (see Example 10.14). The output of both the DT and the underlying CT system settle completely after $\ell = 2n - 1 = 3$ timesteps.

Instead, applying (10.36) for ripple-free deadbeat control of the unstable nonminimum-phase system (10.40), solving $a(z)\,x(z) + b(z)\,y(z) = z^\ell$ with $\ell = 2n - 1 = 3$ using RR_diophantine results in

$$D(z) = \frac{y(z) + a(z)\,\overline{Q}(z)}{x(z) - b(z)\,\overline{Q}(z)} \quad \text{where} \quad \begin{array}{l} y(z) = 89.589\,z - 87.437, \\ x(z) = z + 1.1983. \end{array} \tag{10.41}$$

Again, all causal ripple-free deadbeat controllers in this case are given by taking $\overline{Q}(z) = q(z)/z^k$ with $k \geq 0$ and $deg\,\{q(z)\} \leq k$ [$q(z)$ is otherwise arbitrary]. The DT and CT responses u_k, $u(t)$, y_k, and $y(t)$ resulting from a unit step input in this case, taking $\overline{Q}(z) = 0$, are illustrated in Figure 10.38; note that y_k takes one more timestep to settle than the ripple-free deadbeat controller considered in Example 10.13, but the deadbeat control strategy used in this case may be applied safely to general (unstable, nonminimum-phase) systems.

The controller derived above has closed-loop transfer function of $T(z) = g(z)/z^\ell$ where $g(z) = b(z)\,y(z)$. Note that $g(1) \neq 1$, and thus the closed-loop system has nonzero steady-state error [see (10.33b)]. The easiest way to fix this is with a prefactor of $P = 1/g(1)$ [see Figure 10.11], as used in Figure 10.38. △

10.5 Describing functions

Another way to approximate nonlinear systems for analysis using transfer functions, useful when characterizing finite-amplitude limit-cycle oscillations of certain nonlinear systems, is **describing functions** (**DFs**).

Recall that an input $v_i = A\sin(\omega t)$ to a linear system $L(\cdot)$ gives an output $v_o = A_1\sin(\omega t + \phi)$. As discussed in §8.4, the gain in amplitude of the response is $A_1/A = |L(\mathrm{i}\omega)|$, and the shift in phase of the response is $\phi = \angle L(\mathrm{i}\omega)$; in the linear setting, neither is a function of the magnitude v_o of the sinusoidal input.

Describing functions extend such analyses to certain nonlinear systems by considering a sinusoidal input

$$v_i = A\sin(\omega t)$$

to a nonlinear system $N(\cdot)$, and characterizing the component of the output v_o at the frequency ω of this input. To proceed with this analysis, we generally assume that the nonlinear system $N(\cdot)$ does not "rectify" [see Example 9.13; meaning that a sinusoidal input to $N(\cdot)$ generates a zero-mean output], and also that $N(\cdot)$ does not introduce any subharmonics (at frequencies some fraction of ω); both assumptions are valid for many nonlinear functions of interest. Subject to these assumptions, we may write the resulting output of $N(\cdot)$ as

$$v_o = N(v_i) = A_1\sin(\omega t + \phi_1) + A_2\sin(2\omega t + \phi_2) + \ldots$$

The describing function $N_D(A,\omega)$ characterizes the component of the response v_o of the nonlinear system $N(\cdot)$ at the oscillation frequency ω of the sinusoidal input $v_i = A\sin(\omega t)$. In general, the DF can depend on both the amplitude A and the frequency ω of this input. The DF is a complex function that characterizes the gain and phase shift of the component of the response v_o at the frequency ω of the input v_i; i.e., $|N_D(A,\omega)| = A_1/A$ and $\angle N_D(A,\omega) = \phi_1$, and thus $N_D = (A_1/A)\,e^{\mathrm{i}\phi_1}$.

To appreciate the value of describing functions in the closed-loop setting, consider the loop illustrated in Figure 10.39a. We will for the moment assume that the signal z in this loop, at a persistent oscillation condition, is essentially a sinusoidal oscillation, with

$$v_i = A\,\sin(\omega t) + \text{"small" harmonics.} \tag{10.42}$$

For this oscillation condition to be persistent, the gain at the oscillation frequency ω when tracing all the way around the loop in Figure 10.39a must be unity, and thus [cf. the corresponding linear expression in (10.3)]

$$L(\mathrm{i}\omega)\,N_D(A,\omega)\,(-1) = 1 \quad \Leftrightarrow \quad L(\mathrm{i}\omega)\,N_D(A,\omega) = -1 \quad \Leftrightarrow \quad L(\mathrm{i}\omega) = -1/N_D(A,\omega). \tag{10.43}$$

A common special class of nonlinear systems $N(\cdot)$ for which characterization with DFs is particularly convenient is those for which the amplitude and phase of the fundamental component of their response to sinusoidal inputs are independent of the input frequency ω; for such nonlinear systems, we denote the corresponding DF as simply $N_D(A)$. Figure 10.39b shows how to find graphically the solution of (10.43) in this convenient case, which establishes the conditions for which a persistent oscillation is possible in the closed-loop setting. Note that the LHS and RHS of (10.43) are complex numbers, each with a magnitude and phase. As shown in Figure 10.39b, we may thus look at (10.43), and plot the magnitude versus the phase of both the LHS (which is just a function of ω) and the RHS (which is just a function of A), then look for the point(s) at which these two curves intersect. The value(s) of ω in $L(\mathrm{i}\omega)$, and corresponding value(s) of A in $[-1/N_D(A)]$, at such intersection point(s) then establish the frequency ω and (finite) amplitude A of potential persistent oscillations.

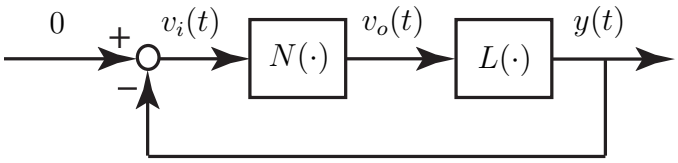


Figure 10.39: Closed loop connection of a nonlinear system $N(\cdot)$ and a linear system $L(s)$. Finite-amplitude limit cycles of such systems may be characterized using the describing function approach of §10.5, representing the response of $N(\cdot)$ to inputs $v_i(t) = A\sin(\omega\, t)$, at the frequency of oscillation ω, as $v_o(t) = |N_D|\sin(\omega\, t + \angle N_D)$.

Example 10.15 DF of an ideal relay

An **ideal relay** is a nonlinear function $N(\cdot)$ which outputs $+1$ for positive inputs and -1 for negative inputs. A sinusoidal input $v_i = A\sin(\omega\, t)$ to such a function, for any ω and A, outputs a square wave of amplitude 1 at frequency ω; this response can be decomposed via its sine decomposition (see *NR* §5) as

$$v_o = \frac{4}{\pi}\Big(\sin\omega\, t + \frac{1}{3}\sin 3\omega\, t + \frac{1}{5}\sin 5\omega\, t + \ldots\Big)$$

The DF in this case is independent of ω, and is written $|N_D(A)| = (4/\pi)/A$ and $\angle N_D(A) = 0$; the first nonzero harmonic is at frequency $3\,\omega$, with amplitude $1/3$ that of the fundamental. △

Example 10.16 DF of a limiter with dead zone

Defining $C = m(\delta_2 - \delta_1)$, a **limiter with dead zone** is a nonlinear function $N(\cdot)$ with output

$$v_o = \begin{cases} 0 & \text{for } |v_i| \le \delta_1, \\ m\cdot\text{sgn}(v_i)\,(|v_i| - \delta_1) & \text{for } \delta_1 < |v_i| < \delta_2, \\ C\cdot\text{sgn}(v_i) & \text{for } |v_i| \ge \delta_2, \end{cases}$$

The DF in this case[32] is again independent of ω, and is written

$$|N_D(A)| = m\Big[f\Big(\frac{\delta_2}{A}\Big) - f\Big(\frac{\delta_1}{A}\Big)\Big] \text{ and } \angle N_D(A) = 0, \text{ where } f = \begin{cases} \frac{2}{\pi}\big(\sin^{-1}\gamma + \gamma\sqrt{1-\gamma^2}\big) & \text{for } 0 \le \gamma \le 1, \\ 1 & \text{for } \gamma > 1, \end{cases}$$

For $\delta_2/A > 1$ and $\delta_1/A \ll 1$, this results in $|N_D(A)| \approx m[1-0] = m$ with negligible harmonics. For significantly smaller δ_2/A and/or larger δ_1/A, $|N_D(A)| < m$, and the harmonics become more significant; a representative case with $\delta_2/A = 0.8$ and $\delta_2/A = 0.2$ is shown in Figure ?c. △

Example 10.17 DF analysis of an ideal relay in closed loop with $L(s) = 1/(s+1)^3$

We now consider an ideal relay, as characterized by its DF in Example 10.15, in a closed loop with a linear system $L(s) = 1/(s+1)^3$, as illustrated in Figure 10.39. △

[32] For complete derivation and tabulation of this and many other useful describing functions, see Gelb & Vander Velde (1968).

Chapter 11

Motion Planning

Chapter 12

Error-Correcting Codes

Contents

As introduced in §1.1.5 and §1.5.3.3, the subject of linear error-correcting codes (ECCs), a.k.a. linear codes (LCs), is at once elegant, intricate, and practical. Efficient LCs may be devised for vectors in $\mathbf{F}_q^n$, with each of their n elements defined over a set of symbols in a *finite field* $\mathbf{F}_q$ of order q, where $q = p^a$ with p prime; cases of particular interest include the *binary field* $\mathbf{F}_2 = \{0, 1\}$, the *ternary field* $\mathbf{F}_3 = \{0, 1, 2\}$, and the *quaternary field* $\mathbf{F}_4 = \{0, 1, \omega, \bar{\omega}\}$, where $\omega = (-1 + \mathrm{i}\sqrt{3})/2$. Codes with $q = 2$, 3, and 4 are called, respectively, linear binary codes (LBCs), linear ternary codes (LTCs), and linear quaternary codes (LQCs). An LC is defined by two matrices, a *basis matrix* V and a *parity-check matrix* H. The use of an LC to communicate data over a noisy channel is straightforward:

- **group** the data into *vectors* (blocks) of length k, with each element defined over an *alphabet* of q symbols;
- **code** each resulting data vector $\mathbf{a} \in \mathbf{F}_q^k$ into a longer codeword $\mathbf{w} \in \mathbf{F}_q^n$, with $n > k$, via $\mathbf{w} = V_{[n,k]_q}\mathbf{a}$;
- **transmit** the corresponding codeword $\mathbf{w}$ over the noisy channel;
- **receive** the (possibly corrupted) message $\hat{\mathbf{w}} \in \mathbf{F}_q^n$ on the other end, and
- **decode** the received message $\hat{\mathbf{w}}$ leveraging $H_{[n,k]_q}$; that is, find the most likely codeword $\mathbf{w}$ corresponding to the received message $\hat{\mathbf{w}}$, and the data vector $\mathbf{a}$ that generated it.

The identification of matrices $\{V_{[n,k]_q}, H_{[n,k]_q}\}$ that define efficient LCs, and streamlined algorithms that can

quickly code and decode messages using such LCs, have a rich history and many remarkable solutions.

On a finite field $\mathbf{F}_q$, addition ($+$) and multiplication ($\cdot$) are closed (that is, they map to elements within the field) and satisfy the usual rules: they are associative, commutative, and distributive, there is a 0 element such that $a+0=a$, there is a 1 element such that $a\cdot 1=a$, for each a there is an element $(-a)$ such that $a+(-a)=0$, and for each $a\neq 0$ there is an element a^{-1} such that $a\cdot a^{-1}=1$. For example, addition and multiplication on $\mathbf{F}_2$, $\mathbf{F}_2$, and $\mathbf{F}_4$ are defined as follows:

$\mathbf{F}_2$:

$+$	0	1
0	0	1
1	1	0

$\cdot$	0	1
0	0	0
1	0	1

$\mathbf{F}_3$:

$+$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

$\cdot$	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

$\mathbf{F}_4$:

$+$	0	1	ω	$\bar{\omega}$
0	0	1	ω	$\bar{\omega}$
1	1	0	$\bar{\omega}$	ω
ω	ω	$\bar{\omega}$	0	1
$\bar{\omega}$	$\bar{\omega}$	ω	1	0

$\cdot$	0	1	ω	$\bar{\omega}$
0	0	0	0	0
1	0	1	ω	$\bar{\omega}$
ω	0	ω	$\bar{\omega}$	1
$\bar{\omega}$	0	$\bar{\omega}$	1	ω

(12.1)

The **Hamming distance** between two vectors in $\mathbf{F}_q^n$ is simply the number of elements that differ between them. An $[n,k]_q$ LC is defined via a set of $k<n$ independent **basis vectors** $\mathbf{v}^i\in\mathbf{F}_q^n$; the q^k distinct valid **codewords** $\mathbf{w}^j\in\mathbf{F}_q^n$ of this LC are given by all *q-ary linear combinations* of the basis vectors $\mathbf{v}^i$ (that is, by all linear combinations with coefficients selected from $\mathbf{F}_q$, with addition and multiplication defined elementwise on $\mathbf{F}_q$). The basis vectors $\mathbf{v}^i$ are generally selected such the **minimum Hamming distance** d of the resulting LC (that is, the minimum distance between any two resulting codewords) is maximized. An LC is often denoted $[n,k,d]_q$, with the minimum distance d of the code specified explicitly.

We denote by $V_{[n,k]_q}$ the $n\times k$ **basis matrix** with the k basis vectors $\mathbf{v}^i$ as columns, and by $W_{[n,k]_q}$ the $n\times q^k$ **codeword matrix** with the q^k codewords $\mathbf{w}^j$ as columns. Without loss of generality, we write $V_{[n,k]_q}$ and a companion $(n-k)\times n$ **parity-check matrix** $H_{[n,k]_q}$ in the **systematic form**[1]

$$H_{[n,k]_q}=\begin{bmatrix}-P_{(n-k)\times k} & I_{(n-k)\times(n-k)}\end{bmatrix},\quad V_{[n,k]_q}=\begin{bmatrix}I_{k\times k}\\ P_{(n-k)\times k}\end{bmatrix},\quad \mathbf{w}^j=\begin{bmatrix}\mathbf{a}^j\\ \mathbf{b}^j\end{bmatrix}=V_{[n,k]_q}\mathbf{a}^j. \tag{12.2}$$

When written in systematic form, each of the codewords $\mathbf{w}^j$ block decomposes into its k **data symbols**[2] $\mathbf{a}^j$ and its $r=n-k$ **parity symbols** $\mathbf{b}^j$; r is sometimes called the *redundancy* of the code. Note that $H_{[n,k]_q}V_{[n,k]_q}=0$ (on $\mathbf{F}_q$)[3], which establishes that each of the basis vectors $\mathbf{v}^i$ so constructed satisfies the **parity-check equation** $H_{[n,k]_q}\mathbf{v}^i=0$ (on $\mathbf{F}_q$), and thus each of the resulting codewords $\mathbf{w}^j$ also satisfies the parity-check equation $H_{[n,k]_q}\mathbf{w}^j=0$ (on $\mathbf{F}_q$). Note further that, for LBCs and LQCs, $P=-P$.

Any $[n,k]_q$ LC C has associated with it an $[n,n-k]_q$ **dual code** $C^\perp$ defined such that[4]

$$C^\perp=\left\{\mathbf{w}\in\mathbf{F}_q^n\ :\ \mathbf{w}\cdot\bar{\mathbf{u}}=0\ \text{for all}\ \mathbf{u}\in C\right\}; \tag{12.3a}$$

the $k\times n$ parity-check matrix $H^\perp$ and $n\times(n-k)$ basis matrix $V^\perp$ of $C^\perp$ may be written in systematic form as

$$H^\perp_{[n,n-k]_q}=\begin{bmatrix}\bar{P}^T & I_{k\times k}\end{bmatrix},\quad V^\perp_{[n,n-k]_q}=\begin{bmatrix}I_{(n-k)\times(n-k)}\\ -\bar{P}^T\end{bmatrix}. \tag{12.3b}$$

[1] In the literature on this subject, it is more common to use a "generator matrix" G to describe the construction of linear codes; the "basis matrix" convention V used here is related simply to the corresponding generator matrix such that $V=G^T$. We find the basis matrix convention to be a bit more natural in terms of its linear algebraic interpretation.

[2] The word "bit", a portmanteau of "binary digit", is reserved for the case with $q=2$; for $q\geq 2$, we use the word "symbol" instead.

[3] The qualifier "on $\mathbf{F}_q$" is used to reinforce the fact that each individual multiplication and addition specified is performed on the finite field $\mathbf{F}_q$, following the rules specified explicitly in (12.1).

[4] Overbar simply denotes (elementwise) complex conjugate; for LBCs and LTCs, $\bar{\mathbf{u}}=\mathbf{u}$ and $\bar{P}=P$.

Note that $\bar{P}^T$ is of order $k \times (n-k)$, and, of course, that $H^{\perp}_{[n,n-k]_q} V^{\perp}_{[n,n-k]_q} = 0$ (on $\mathbf{F}_q$).

If H and V are the parity-check and basis matrices of an $[n,k,d]_q$ LC, with $HV = 0$ (on $\mathbf{F}_q$), then an **equivalent** LC (with the same n, k, and d) may be generated[5] by taking

$$\tilde{H} = R^T H Q, \quad \tilde{V} = Q^T V S, \quad \tilde{\mathbf{w}} = Q^T \mathbf{w} \qquad (\text{on } \mathbf{F}_q), \tag{12.4}$$

where $Q_{n\times n}$ is a permutation matrix, and each column of $R_{(n-k)\times(n-k)}$ and $S_{k\times k}$ is nonzero and independent (on $\mathbf{F}_q$) from the other columns of R and S, respectively[6]. The permutation matrix Q reorders the rows of V and the corresponding columns of H (i.e., it reorders the data symbols and parity symbols in the LC). Each column of S performs a linear combination of the columns of $(Q^T V)$ to form the corresponding column of $\tilde{V}$, while each column of R (that is, each row of R^T) performs a linear combination of the rows of (HQ) to form the corresponding row of $\tilde{H}$; these modifications by S and R leave the set of valid codewords in the LC unchanged.

A **self-dual** code C is an LC for which the dual code $C^{\perp}$ [see (12.3)] is equivalent to C [see (12.4)].

12.1 Characterizing the Hamming distance d of linear codes

Graphically, the codewords of an $[n,k,d]_2$ LBC amount to a carefully chosen subset of 2^k of the 2^n corners on a single n-dimensional unit hypercube, as illustrated for $n = \{3,\ 4,\ 7,\ 8\}$ in Figures $\{$12.1, 12.2, 12.4, 12.5$\}$, where d quantifies the minimum number of edges (i.e., bits) that differ between any two valid codewords. Further:

- An LC with $d = 2$ is **single error detecting** (**SED**) [see, e.g., Figures 12.1a and 12.2a]. In this case, the sum (on $\mathbf{F}_q$) of the symbols in each transmitted codeword is zero, so if it is assumed that at most one symbol error occurred and this sum is nonzero, then a symbol error in transmission occurred, whereas if this sum is zero, then a symbol error did not occur. However, if a symbol error in transmission occurred, the received (invalid) message is generally equidistant from multiple codewords, so it is not possible to correct the error. Two or more symbol errors generally cause the codeword to be misinterpreted.
- An LC with $d = 3$ is **single error correcting** (**SEC**) [see, e.g., Figures 12.1b and 12.4]. In this case, if it is again assumed that at most one symbol error in transmission occured, then if the received codeword is not a codeword, there is only one codeword that is unit Hamming distance away, so the single symbol error may in fact be *corrected*. Again, two or more symbol errors generally cause the codeword to be misinterpreted.
- An LC with $d = 4$ is **single error correcting and double error detecting** (**SECDED**) [see, e.g., Figures 12.2b and 12.5]. In this case, if a single symbol error occurs, the received codeword will be unit Hamming distance away from a single codeword, and thus single symbol errors can be corrected. On the other hand, if two symbol errors occur, the received codeword is generally Hamming distance 2 away from multiple codewords, so double symbol errors can be detected but *not* corrected. In this case, three or more symbol errors cause the codeword to be misinterpreted.

Such labels and their natural extensions (DEC for $d = 5$, DECTED for $d = 6$, TEC for $d = 7$, TECQED for $d = 8$, 4EC for $d = 9$, 4EC5ED for $d = 10$, 5EC for $d = 11$, 5EC6ED for $d = 12$, etc) may be used to quantify the error correction capability of an LC. Alternatively, if error correction is *not* attempted, then:

- an LC with $d = 2$ is single error detecting, with 2 or more symbol errors generally causing misinterpretation,
- an LC with $d = 3$ is double error detecting, with 3 or more symbol errors generally causing misinterpretation,
- an LC with $d = 4$ is triple error detecting, with 4 or more symbol errors generally causing misinterpretation,

etc. Error correcting codes are useful for a broad range of data transmission and data storage applications in which it is difficult or impossible to request that a corrupted codeword be retransmitted; algorithms which

[5] Though many equivalent codes can be transformed between their various representations via (12.4) [see, e.g., conversions between the systematic, cyclic, and syndrome forms of the $[2^m-1, 2^m-1-m, 3]$ binary Hamming codes discussed in §12.4], not all $[n,k,d]_q$ codes that are equivalent may be transformed between their various forms via (12.4), as their parity bits may be defined differently.

[6] For a permutation matrix, $QQ^T = 0$. (In general, matrix inversion on a finite field $\mathbf{F}_q$ can be performed using Cramer's rule.)

Figure 12.1: Codewords of (left) the (SED) $[3, 2, 2]$ LBC; (right) its dual, the (perfect, SEC) $[3, 1, 3]$ LBC.

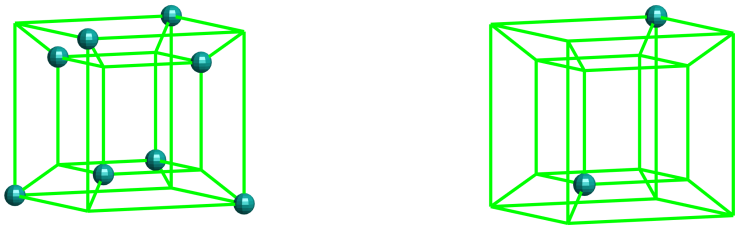


Figure 12.2: Codewords of (left) the (SED) $[4, 3, 2]$ LBC; (right) its dual, the (quasi-perfect, SECDED) $[4, 1, 4]$ LBC.

use such codes for error detection only may be useful when efficient handshaking is incorporated in a manner which makes it easy to request and resend any messages that might be corrupted during transmission.

An $[n, k, d]_q$ LC is said to be **perfect** if, for some integer $t > 0$, each possible n-dimensional q-ary codeword is of Hamming distance t or less from a single codeword (that is, there are no "wasted" vectors that are Hamming distance $t + 1$ or more from the valid codewords, and thus may not be corrected under the assumption that at most t symbol errors have occured). A perfect code has odd $d = 2t + 1 > 1$. A remarkable proof by Tietäväinen (1973) establishes that the *only* nontrivial perfect LCs are the $[(q^r - 1)/(q - 1), (q^r - 1)/(q - 1) - r, 3]_q$ q-ary Hamming codes [§12.4], the $[23, 12, 7]_2$ binary Golay code [§12.5] and the $[11, 6, 5]_3$ ternary Golay code.

An $[n, k, d]_q$ LC is said to be **quasi-perfect** if, for some integer $t > 1$, each possible n-dimensional q-ary codeword is either (a) of Hamming distance $t - 1$ or less from a single codeword, and thus up to $t - 1$ symbol errors may be corrected, or (b) of Hamming distance t from at least one codeword, and thus codewords with t symbol errors may be detected but not necessarily corrected (that is, there are no "wasted" vectors that are Hamming distance $t + 1$ or more from a valid codeword, and thus may not be reconciled under the assumption that at most t symbol errors have occured). A quasi-perfect code has even $d = 2t > 2$; examples include the $[(q^r - 1)/(q - 1) + 1, (q^r - 1)/(q - 1) - r, 4]_q$ extended q-ary Hamming codes [§12.4.1], the $[24, 12, 8]_2$ extended binary Golay code [§12.5], and the $[12, 6, 6]_3$ extended ternary Golay code.

For a high-level view of this general subject, the "best" (largest d) available $[n, k, d]_2$ LBCs, for a range of k data bits and $r = n - k$ parity bits, are illustrated in Figure 12.3, which plots the best available **data rate** k/n versus k for various values of d, from $d = 2$ (SED) to $d = 12$ (5EC6ED), as highlighted by the broken diagonal lines and the color schmeme implemented. The $[3, 1, 3]$, $[4, 1, 4]$, $[3, 2, 2]$, and $[4, 3, 2]$ LBCs depicted visually in Figures 12.1 and 12.2, and the $[7, 4, 3]$ and $[8, 4, 4]$ LBCs depicted visually in Figures 12.4 and 12.5, are indicated near the left of Figure 12.3 (in the $k = \{1, 2, 3, 4\}$ columns). It is seen that, as the number of data bits in the packet, k, is increased, codes with improved data rates k/n generally become available for a given degree of error correction capability d. Many of the most notable codes indicted in this figure (before each break in the diagonal lines) are discussed in §12.3-12.5; codes connected by vertical lines are related by **puncturing** the code below; codes connected by diagonal lines are related by **shortening** the code to the upper-right (see §12.6).

The selection of the "best" code for a given purpose involves a number of tradeoffs. In addition to maximizing the data rate k/n for a given number of data bits k and a given degree of error correction capability d, the existence of fast algorithms to **code** the message (determining its parity bits), and to **decode** the message (checking for errors, and determining the nearest valid codeword) is also quite important. In this regard, LCs with **cyclic forms** (see §12.2) and **syndrome decoding** strategies (see §12.4.0.1) are especially valuable.

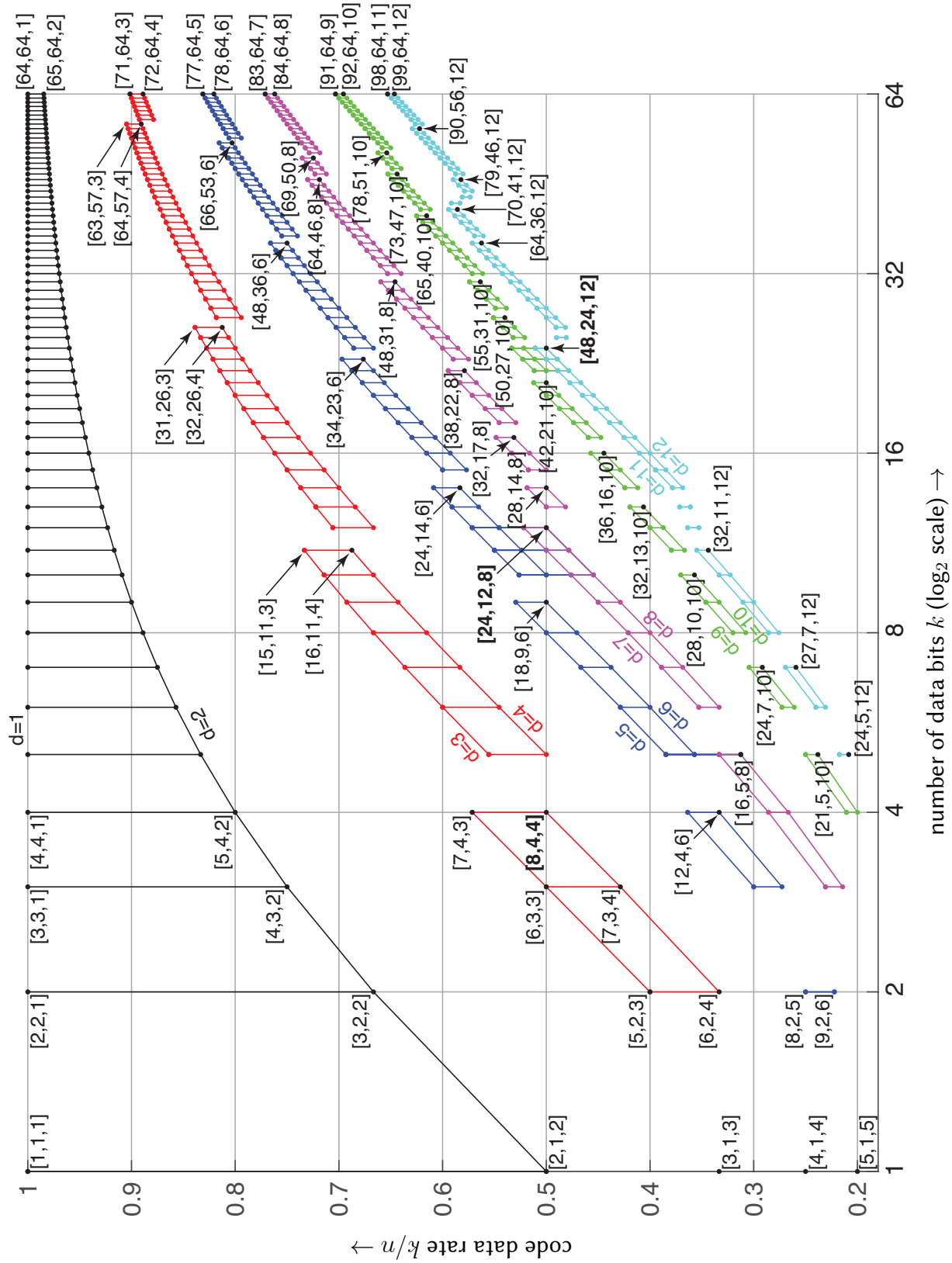


Figure 12.3: The densest known linear binary codes $[n, k, d]_2$ with Hamming distance $1 \leq d \leq 12$.

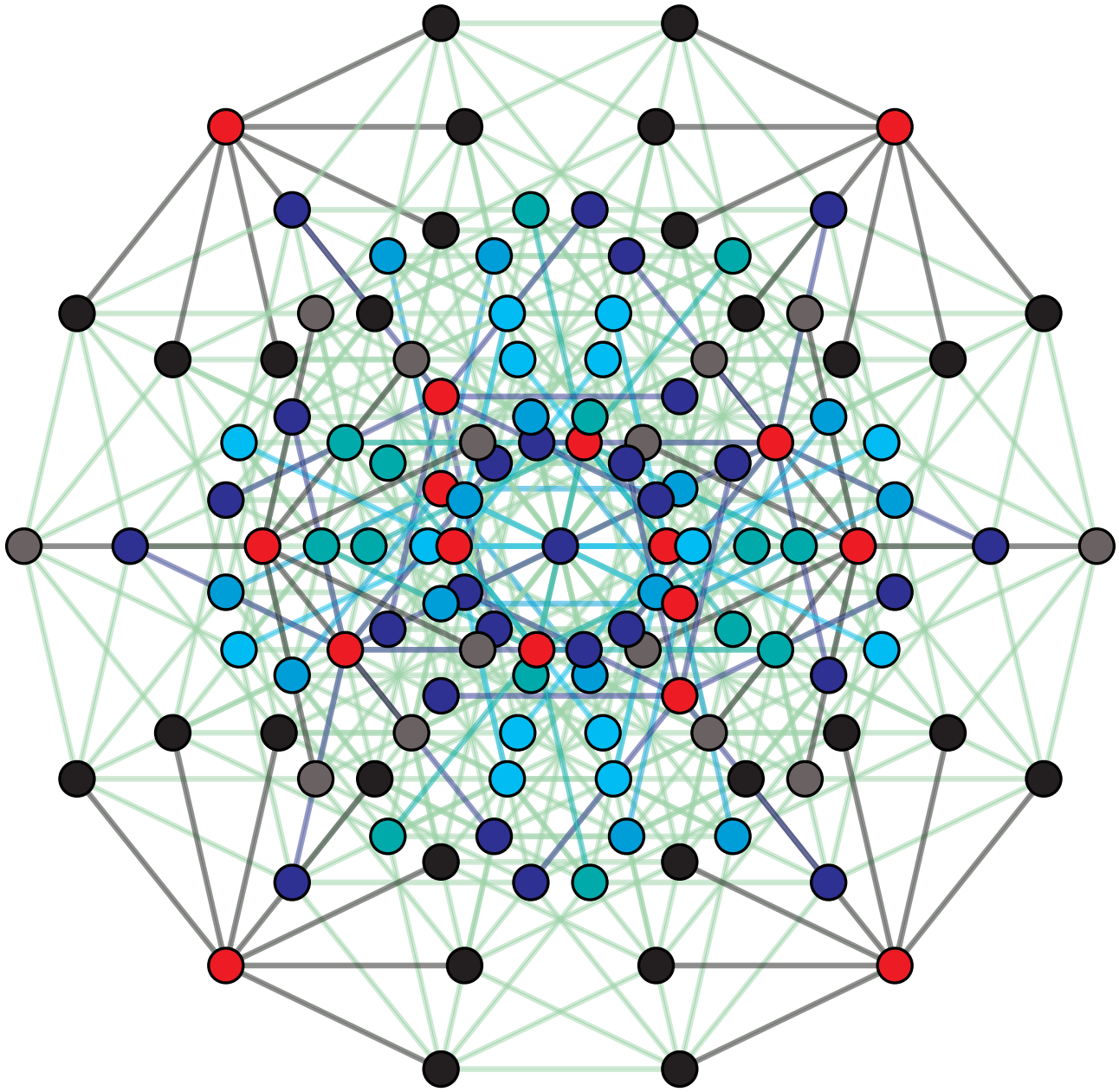


Figure 12.4: The $2^k = 16$ valid codewords (in red) of the (perfect, SEC) $[7, 4, 3]$ LBC (see §12.4 and Figure 1.5a), amongst the $2^n = 128$ possible received messages. Each of the $16 \cdot 7 = 112$ invalid messages is unit Hamming distance from a single valid codeword, thus facilitating single error correction. Note that each valid codeword in the above figure is indeed 3 edges from the nearest valid codeword; in this 2-dimensional orthogonal projection (called a hepteract) of the corners of a 7-dimensional hypercube, some edges overlap. The **data rate** is $k/n = 4/7 = 0.571$ (that is, 4 data bits out of every 7 message bits), which compares favorably to the data rate of 1/3 for the (perfect, SEC) $[3, 1, 3]$ LBC in Figure 12.1b. The (perfect, SEC) $[15, 11, 3]$, $[31, 26, 3]$, $[63, 57, 3]$, $[127, 120, 3]$ binary Hamming codes (see §12.4) have increasingly higher data rates (see Figure 12.3).

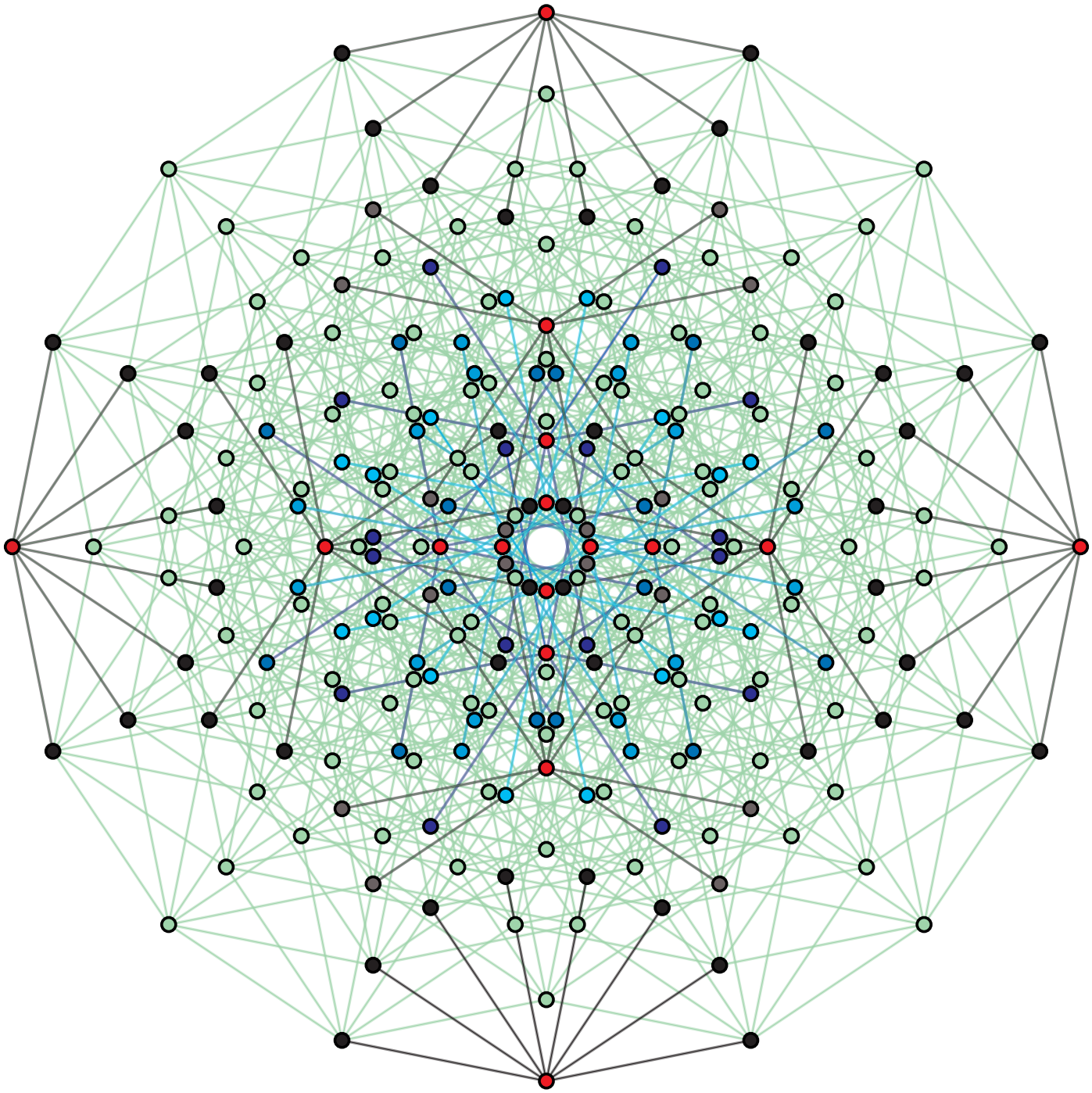


Figure 12.5: The $2^k = 16$ valid codewords (in red) of the (quasi-perfect, SECDED, self-dual) $[8, 4, 4]$ LBC (see §12.4.1), amongst the $2^n = 256$ possible received messages. Of the 240 invalid messages, $16 \cdot 8 = 128$ messages are each unit Hamming distance from a single valid codeword, thus facilitating single error correction, and the remaining $240 - 128 = 112$ messages (in green) are each Hamming distance 2 from multiple valid codewords, thus facilitating double error detection. Each valid codeword in the above figure is 4 edges from the nearest valid codeword; in this 2-dimensional orthogonal projection (called a octeract) of the corners of a 8-dimensional hypercube, some edges overlap. The data rate is $k/n = 4/8 = 1/2$, which compares favorably to the data rate of $1/4$ for the (quasi-perfect, SECDED) $[4, 1, 4]$ LBC in Figure 12.2b (see Figure 12.3). The (quasi-perfect, SECDED) $[16, 11, 4]$, $[32, 26, 4]$, $[64, 57, 4]$, $[128, 120, 4]$ binary Hamming codes (see §12.4.1) have increasingly higher data rates (see Figure 12.3).

12.2 Cyclic form

A cyclic code is an LC that may be written in a **cyclic form** in which the $n \times k$ basis matrix $V^c_{[n,k]_q}$ and the $r \times n$ parity-check matrix $H^c_{[n,k]_q}$, with $n = r + k$, have the form

$$H^c_{[n,k]_q} = \begin{pmatrix} h_k & h_{k-1} & \dots & h_0 & & 0 \\ & h_k & h_{k-1} & \dots & h_0 & \\ & & \ddots & \ddots & \ddots & \ddots \\ 0 & & & h_k & h_{k-1} & \dots & h_0 \end{pmatrix}, \quad V^c_{[n,k]_q} = \begin{pmatrix} v_0 & & & 0 \\ v_1 & v_0 & & \\ \vdots & v_1 & \ddots & \\ v_r & \vdots & \ddots & v_0 \\ & v_r & \ddots & v_1 \\ & & \ddots & \vdots \\ 0 & & & v_r \end{pmatrix}, \tag{12.5a}$$

where $\{h_k, h_0, v_r, v_0\}$ are nonzero with

$$h_k\, v_r + h_0\, v_0 = 0 \quad (\text{on } \mathbf{F}_q), \tag{12.5b}$$

and where [as with the systematic form (12.2)], of course,

$$H^c_{[n,k]_q}\, V^c_{[n,k]_q} = 0 \quad (\text{on } \mathbf{F}_q). \tag{12.5c}$$

A convenient construction which simplifies the math of cyclic LCs is the *cyclic shift* operator z, which is used in a manner akin to the Z-transform analysis of discrete-time linear systems (see §8), with the major difference being that polynomials in z are manipulated here in a cyclic context on $\mathbf{F}_q^n$. That is, arithmetic with polynomials in z and coefficients in $\mathbf{F}_q$ is performed as usual, except that the coefficients of each power of z are combined via the arithmetic rules on $\mathbf{F}_q$ [see (12.1)], and higher powers of z^k are simplified via the *cyclic condition*

$$\boxed{z^n = 1.} \tag{12.6}$$

In the analysis and use of an $[n,k]_q$ cyclic LC, the operator z appears in

the **data polynomials**	$a(z) = a_{k-1}z^{k-1} + a_{k-2}z^{k-2} + \ldots + a_1 z + a_0$	(12.7a)
the **basis polynomial**	$v(z) = v_r z^r + v_{r-1}z^{r-1} + \ldots + v_1 z + v_0$	(12.7b)
the **codeword polynomials**	$w(z) = w_{n-1}z^{n-1} + w_{n-2}z^{n-2} + \ldots + w_1 z + w_0$	(12.7c)
the **received-message polynomials**	$\hat{w}(z) = \hat{w}_{n-1}z^{n-1} + \hat{w}_{n-2}z^{n-2} + \ldots + \hat{w}_1 z + \hat{w}_0,$ and	(12.7d)
the **parity-check polynomial**	$h(z) = h_k z^k + h_{k-1}z^{k-1} + \ldots + h_1 z + h_0.$	(12.7e)

The use of a cyclic LC to communicate data over a noisy channel is again straightforward:

- **group** the data into vectors $\mathbf{a} \in \mathbf{F}_q^k$, and express as polynomials $a(z)$ with the k elements of $\mathbf{a}$ as coefficients;
- **code** each $a(z)$ as a polynomial $w(z)$ with $h(z)\, w(z) = 0$[7], and express $w(z)$ as a vector $\mathbf{w} \in \mathbf{F}_q^n$;
- **transmit** the corresponding codeword vector $\mathbf{w}$ over the noisy channel;
- **receive** the (possibly corrupted) message $\hat{\mathbf{w}} \in \mathbf{F}_q^n$ on the other end, and express as a polynomial $\hat{w}(z)$;
- **decode** $\hat{w}(z)$ leveraging the parity-check polynomial $h(z)$; i.e., find the most likely codeword polynomial $w(z)$ corresponding to $\hat{w}(z)$, and the data polynomial $a(z)$ [and, thus, the data vector $\mathbf{a}$] that generated it.

[7] One way of accomplishing this is simply by defining $w(z) = a(z)\, v(z)$. In practical implementations, a different convention (which is faster to decode), called the **systematic cyclic form**, is commonly used. With this approach, the codeword polynomial $w(z)$ is defined by taking the first k coefficients of $w(z)$ as the coefficients the data polynomial $a(z)$ [see (12.11)]. The remaining coefficients of $w(z)$, which make up the **parity polynomial** $b(z)$, are then defined such that $h(z)\, w(z) = 0$; (12.12) and (12.13) present two equivalent constructions to compute the required $b(z) = b_{r-1}z^{r-1} + \ldots + b_1 z + b_0$ when using this form.

The basis and parity-check polynomials, $v(z)$ and $h(z)$, are constructed by factorization of $(z^n - 1)$ as follows:

$$\boxed{h(z)\,v(z) = [h_k z^k + \ldots + h_1 z + h_0]\,[v_r z^r + \ldots + v_1 z + v_0] = z^n - 1 = 0 \quad (\text{on } \mathbf{F}_q).} \tag{12.8}$$

Note that:

(i) the scalar product of the first row of H^c and the first column of V^c in (12.5a) is precisely the coefficient of z^k in (12.8), once multiplied out, which by construction [see the RHS of (12.8)] is zero;
(ii) the scalar product of the first row of H^c and the last column of V^c is the coefficient of z^1 in (12.8) [that is, $h_1\,v_0 + h_0\,v_1$ (on $\mathbf{F}_q$)], which by construction is also zero;
(iii) ...analogously, the scalar product of any row of H^c with any column of V^c is simply the coefficient of z^j in (12.8), for $1 \leq j \leq k$, which by construction is zero.

Thus, the condition (12.8) enforces (12.5c) [that is, $H^c\,V^c = 0$]. Applying the cyclic condition (12.6), the coefficient of z^0 in (12.8), $h_k\,v_r + h_0\,v_0$, is also zero, and thus (12.5b) is enforced. Indeed, it follows from (12.8) and (12.6) that

$$[z^i\,h(z)]\,[z^j\,v(z)] = z^{i+j}\,[h(z)\,v(z)] = z^{i+j}\,[z^n - 1] = 0 \quad (\text{on } \mathbf{F}_q) \tag{12.9}$$

for any integer i and j and thus, performing arithmetic on $\mathbf{F}_q$ and computing in the cyclic setting with $z^n = 1$, interpreting z as a shift operator, it follows that *any cyclic shift of the parity check polynomial $h(z)$ is orthogonal (on $\mathbf{F}_q^n$) to any cyclic shift of the basis polynomial $v(z)$*. Since it follows that all cyclic shifts of the basis vectors are themselves valid codewords, and noting that all codewords are formed by linear combinations of the basis vectors, it follows that *any cyclic shift of a valid codeword is itself also a valid codeword*, which is indeed how the class of cyclic codes gets its name.

12.2.1 Constructing cyclic codes

Cyclic codes are thus constructed via factorizations of the form (12.8). One factorization of $(z^n - 1)$ on $\mathbf{F}_q$, which exists for any n and q, is

$$z^n - 1 = (z-1)(z^{n-1} + z^{n-2} + \ldots + z + 1); \tag{12.10}$$

this leads to the single parity check code $[n, n-1, 2]_q$ (see §12.3) if one takes $v(z) = (z-1)$ and $h(z)$ equal to the rest, and to the repetition code $[n, 1, n]_q$ (see §12.3.1) if one takes $h(z) = (z-1)$ and $v(z)$ equal to the rest.

If q is not prime, developing other factorizations of $(z^n - 1)$ over $\mathbf{F}_q$ is a bit delicate, as $(z^n - 1)$ does not, in general, factor into the product of unique irreducible forms in this case. As an example, two irreducible factorizations of $(z^5 - 1)$ over $\mathbf{F}_4$ are listed in Table 12.1.

For prime q, however, the development of other factorizations of $(z^n - 1)$ over $\mathbf{F}_q$ is significantly more straightforward, as $(z^n - 1)$ may be factored into the product of unique irreducible forms in this case. A few such factorizations for various values of n are listed in Table 12.2 for $q = 3$ (in which "-1" $= 2$), and in Table 12.3 for $q = 2$ (in which "-1" $= 1$); others are easily found using Matlab or Mathematica. Noting (12.8), these unique irreducible factors of $(z^n - 1)$ may be grouped in different ways to form $v(z)$ and $h(z)$. Most cyclic codes may be constructed via such an approach, only some of which have a favorable minimum distance d and an available simple error correction scheme. A few such codes, with $q = 2$, are listed in Table 12.4.

$z^5 - 1 = (z+1)(z^4 + z^3 + z^2 + z + 1)$
$z^5 - 1 = (z^2 + \omega z + 1)(z^3 + \omega z^2 + \omega z + 1)$

Table 12.1: Two (nonunique!) irreducible factorizations of $(z^5 - 1)$ over $\mathbf{F}_4$ [see (12.1)], with $\omega = (-1 + \mathrm{i}\sqrt{3})/2$.

$$
\begin{aligned}
z^4 - 1 &= (z+2)(z+1)(z^2+1) \\
z^{11} - 1 &= (z+2)(z^5 + 2z^3 + z^2 + 2z + 2)(z^5 + z^4 + 2z^3 + z^2 + 2) \\
z^{13} - 1 &= (z+2)(z^3 + 2z + 2)(z^3 + z^2 + 2)(z^3 + z^2 + z + 2)(z^3 + 2z^2 + 2z + 2)
\end{aligned}
$$

Table 12.2: Some unique irreducible factors of $(z^n - 1)$ over $\mathbf{F}_3$ [see (12.1)] for various values of n.

$$
\begin{aligned}
z^5 - 1 &= (z+1)(z^4 + z^3 + z^2 + z + 1) \\
z^7 - 1 &= (z+1)(z^3 + z + 1)(z^3 + z^2 + 1) \\
z^{15} - 1 &= (z+1)(z^2 + z + 1)(z^4 + z + 1)(z^4 + z^3 + 1)(z^4 + z^3 + z^2 + z + 1) \\
z^{23} - 1 &= (z+1)(z^{11} + z^9 + z^7 + z^6 + z^5 + z + 1)(z^{11} + z^{10} + z^6 + z^5 + z^4 + z^2 + 1)
\end{aligned}
$$

Table 12.3: Some unique irreducible factors of $(z^n - 1)$ over $\mathbf{F}_2$ [see (12.1)] for various values of n.

$[n,k,d]_q$ code	r	$v(z)$	$h(z)$
$[n, n-1, 2]_2$	1	$z+1$	$z^{n-1} + z^{n-2} + \ldots + z + 1$
$[7, 4, 3]_2$	3	$z^3 + z + 1$	$z^4 + z^2 + z + 1$
$[15, 11, 3]_2$	4	$z^4 + z + 1$	$z^{11} + z^8 + z^7 + z^5 + z^3 + z^2 + z + 1$
$[31, 26, 3]_2$	5	$z^5 + z^2 + 1$	$z^{26} + z^{23} + z^{21} + z^{20} + z^{17} + z^{16} + z^{15} + z^{14} + z^{13} + z^9 + z^8 + z^6 + z^5 + z^4 + z^2 + 1$
$[63, 57, 3]_2$	6	$z^6 + z + 1$	Matlab: `mod(deconv([1 zeros(1,62) 1],[1 0 0 0 0 1 1]),2)`
$[127, 120, 3]_2$	7	$z^7 + z^3 + 1$	`mod(deconv([1 zeros(1,126) 1],[1 0 0 0 1 0 0 1]),2)`

Table 12.4: Some small binary cyclic codes, defined such that $v(z)\, h(z) = (z^n - 1)$ over $\mathbf{F}_2$, with $r = n - k$.

12.2.2 Cyclic coding

As mentioned in Footnote 7 a couple of pages back, by convention, practical implementations of cyclic codes usually shift the k data symbols of $a(z)$ to one end of the codeword $w(z)$, for example

$$
\begin{aligned}
w(z) &= z^r a(z) + b(z) \\
&= a_{k-1} z^{n-1} + \ldots + a_1 z^{r+1} + a_0 z^r + b_{r-1} z^{r-1} + \ldots + b_1 z + b_0 \\
&= w_{n-1} z^{n-1} + \ldots + w_1 z^{n-k+1} + w_{n-k} z^{n-k} + w_{n-k-1} z^{n-k-1} + \ldots + b_1 z + b_0,
\end{aligned}
\tag{12.11}
$$

and then determine the parity polynomial $b(z)$ within $w(z)$ in a manner such that $h(z)\, w(z) = 0$.

For $k \lesssim r$ [that is, short $h(z)$ and long $v(z)$], a recursive approach may be used to determine the parity symbols $b(z)$ from $h(z)$ and $a(z)$. By (12.8), (12.9), and (12.6), and the fact that each valid codeword polynomial $w(z)$ is itself a linear combination of the basis polynomials $v(z)$, we have

$$
h(z)\, w(z) \triangleq u_{n-1} z^{n-1} + u_{n-2} z^{n-2} + \ldots + u_1 z + u_0 = 0.
$$

Initializing the first k symbols of $w(z)$ as a_{k-1} through a_0 as in (12.11), noting that $h_k \neq 0$, the remaining symbols of $w(z)$, in $b(z)$, may be determined from the resulting convolution formulae for u_{n-1} through u_k as follows:

$$
\begin{array}{lcl}
u_{n-1} = h_0 w_{n-1} + \ldots + h_k w_{n-k-1} = 0 & \Rightarrow & \boxed{b_{r-1} = w_{n-k-1} = -[h_0 w_{n-1} + \ldots + h_{k-1} w_{n-k} \quad]/h_k,} \\
u_{n-2} = h_0 w_{n-2} + \ldots + h_k w_{n-k-2} = 0 & \Rightarrow & \boxed{b_{r-2} = w_{n-k-2} = -[h_0 w_{n-2} + \ldots + h_{k-1} w_{n-k-1}]/h_k,} \\
\quad \vdots & & \boxed{\qquad\qquad \vdots} \\
u_k = h_0 w_k \quad + \ldots + h_k w_0 \quad = 0 & \Rightarrow & \boxed{b_0 \quad = w_0 \quad = -[h_0 w_k \quad + \ldots + h_{k-1} w_1 \quad]/h_k.}
\end{array}
\tag{12.12}
$$

For $r < k$ [that is, short $v(z)$ and long $h(z)$], a direct polynomial division approach to determine the parity symbols from $v(z)$ and $a(z)$ is more efficient. This may by achieved by writing the shift of the data symbols as some multiple of the basis polynomial $v(z)$ plus a remainder $t(z)$:

$$z^r a(z) = q(z)v(z) + t(z) \quad \Leftrightarrow \quad [z^r a(z)] \bmod v(z) = t(z).$$

Note that $v_r \neq 0$. Calculating $t(z)$ as shown above, taking $b(z) = -t(z)$ and moving this term to the LHS in the above expression, it is seen [cf. (12.11)] that

$$z^r a(z) + b(z) = q(z)v(z) \triangleq w(z),$$

verifying that the $w(z)$ so generated is a valid codeword polynomial, as it is a multiple of $v(z)$, and thus $h(z)\, w(z) = [h(z)\, v(z)]\, q(z) = 0$. Computation of $t(z)$ via polynomial division is straightforward[8]:

$$\begin{array}{l} t(z) = z^r a(z) \quad \% \text{ Initialize } t(z) \text{ as the } (n-1)\text{'th order polynomial } z^r a(z) \\ \text{for } i = n : -1 : r+1 \qquad \% \text{ Zero the } i\text{'th coefficient in } t(z) \text{ by subtracting multiples of } v(z) \\ \quad t(z) = t(z) - z^{i-r-1}\, v(z)\, [t_i / v_r] \qquad \% \text{ Note: arithmetic must be performed on } \mathbf{F}_q! \\ \text{end} \qquad \% \text{ Returns remainder } t(z) \text{ when } v(z) \text{ is divided into } z^r a(z) \end{array} \tag{12.13}$$

A binary ($q = 2$) implementation of (12.13) is given in Algorithm 12.1. Defining the $(r-1)$'th order parity polynomial $b(z) = -t(z)$ computed via this approach, the remaining symbols of $w(z)$ in (12.11) are determined.

12.2.3 Cyclic decoding

Using cyclic codes (12.5) constructed in the form (12.11) [leveraging (12.12) or (12.13) to compute $b(z)$], received codewords are easy to check for errors, and trivial to decode. To check for errors, two approaches are possible:

a) multiply each of the $r = n - k$ rows of H [which by (12.5a) are each simple shifts of the k symbols of the parity check polynomial $h(z)$, padded with zeros] times each received message $\hat{\mathbf{w}}$ [corresponding to the n symbols of the polynomial $\hat{w}(z)$], noting that the received message is error free [i.e., $\hat{w}(z) = w(z)$] if each of these scalar products is zero; or
b) recompute the check bits from the data bits in the received message $\hat{w}(z)$ using (12.13), and compare with the check bits in $\hat{w}(z)$; the received message is error-free if they match.

To decode, note by (12.11) that the k symbols of the data polynomial $a(z)$ are just the first k symbols of $w(z)$.

The polynomial multiplications and divisions involved in the cyclic coding and decoding algorithms described above may be calculated efficiently in either an application-specific integrated circuit (ASIC) or field-programmable gate array (FPGA), as discussed in §1.5.3, in which repeated computations with shifted data may be performed, in parallel, remarkably quickly. The reduced storage associated with the vector representation of the basis matrix and the parity-check matrix in cyclic form help to facilitate such implementations.

An extended summary of the subject of linear binary ($q = 2$), ternary ($q = 3$), and quaternary ($q = 4$) codes with a broad range of k and d (including a summary of the longer turbo codes), and the connection between these codes and n-dimensional lattice packings, is provided in *RP*, and the references therein. The remainder of this chapter focuses on a few families of **linear binary codes** (LBCs) with $q = 2$ and rather small k and d (see Figure 12.3), usually denoted simply as $[n, k]$ or $[n, k, d]$ (dropping the q subscript), which are useful in embedded applications. We also, in §12.7, discuss the use of binary **cyclic redundancy check** (CRC) codes, defined again by a basis polynomial $v(z)$ as laid out above, but for which, in application, the number of data bits k to be transmitted is *variable*, based on the communication needs of the system and the noise on the transmission line, and is determined on the fly.

[8] Note that the quotient $q(z)$ of this polynomial division is not actually needed, just the remainder $t(z)$.

12.3 Binary single parity check codes and their dual forms

The $[n, n-1, 2]$ *binary*[9] *single parity-check codes* are SED, and include $[2, 1, 2]$ (self-dual), $[3, 2, 2]$, $[4, 3, 2]$, etc. Using such a code, for each $k = n - 1$ data bits to be sent, a single ($r = 1$) *parity bit* is generated such that the sum (on $\mathbf{F}_2$) of the data bits and parity bit is 0; when decoding, an error is flagged if this sum (on $\mathbf{F}_2$) is 1. The $[3, 2, 2]$ code illustrated in Figure 12.1a, with $P = \begin{pmatrix} 1 & 1 \end{pmatrix}$ [see (12.2)] and $2^k = 4$ valid codewords, is given by

$$H_{[3,2,2]} = \big(\underbrace{1 \quad 1}_{P} \quad 1 \big), \quad V_{[3,2,2]} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}, \quad W_{[3,2,2]} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}. \tag{12.14}$$

The $[4, 3, 2]$ code illustrated in Figure 12.2a, with $P = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$ and $2^k = 8$ valid codewords, is given by

$$H_{[4,3,2]} = \big(\underbrace{1 \quad 1 \quad 1}_{P} \quad 1 \big), \quad V_{[4,3,2]} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad W_{[4,3,2]} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}. \tag{12.15}$$

Other binary single parity-check codes have a partity submatrix P [see (12.2)] of similar form (a row of 1's).

In cyclic form [see (12.5)], binary single parity-check codes are generated by $h(z) = z^{n-1} + z^{n-2} + \ldots + z + 1$ and $v(z) = z + 1$. For example, for $n = 4$, the H and V matrices for a binary single parity check code in cyclic form is given below left; these two matrices may be transformed back to the systematic form in (12.15) via (12.4) by taking $R = I$, $Q = I$, and (for any n) S as defined below right:

$$H^c_{[4,3,2]} = \begin{pmatrix} 1 & 1 & 1 & 1 \end{pmatrix}, \quad V^c_{[4,3,2]} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}; \qquad S = \begin{pmatrix} 1 & & 0 \\ \vdots & \ddots & \\ 1 & \ldots & 1 \end{pmatrix}. \tag{12.16}$$

A single parity-check code (binary or otherwise), with $d = 2$, can detect but not correct an error in an unknown position. However, it can correct an **erasure** (i.e., the loss of data from a known position). A common application of this capability is in a **RAID 5** system (i.e., *Redundant Array of Independent Disks*) for data storage. In such a system, data is striped across n hard disk drives using a single parity check code; if any single drive fails (which is unfortunately fairly common, relative to the expected shelf life of certain data), the failed drive can be swapped out, and data on it can be recovered simply by achieving parity with the other disks.

12.3.1 Binary repetition codes

The dual [see (12.3b)] of the binary single parity-check codes are the $[n, 1, n]$ *binary repetition codes*, which include $[2, 1, 2]$ (SED, self-dual), $[3, 1, 3]$ (SEC, perfect), $[4, 1, 4]$ (SECDED, quasi-perfect), $[5, 1, 5]$ (DEC, perfect), etc. This family of codes repeats a single ($k = 1$) data bit n times (i.e., $r = n - 1$); when decoding, if an error is detected (i.e., if $H\hat{\mathbf{w}} \neq 0$), one may find which of the two valid codewords that the received message is closest to simply by majority vote. The $[3, 1, 3]$ code illustrated in Figure 12.1b, with $P = \begin{pmatrix} 1 & 1 \end{pmatrix}^T$, is given by

$$H_{[3,1,3]} = \begin{pmatrix} 1 & 1 & 0 \\ \underbrace{1}_{P} & 0 & 1 \end{pmatrix}, \quad V_{[3,1,3]} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad W_{[3,1,3]} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}. \tag{12.17}$$

The $[4, 1, 4]$ code illustrated in Figure 12.2b, with $P = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}^T$, is given by

$$H_{[4,1,4]} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ \underbrace{1}_{P} & 0 & 0 & 1 \end{pmatrix}, \quad V_{[4,1,4]} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad W_{[4,1,4]} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}. \tag{12.18}$$

[9] As suggested previously, for the remainder of §12, the q subscript is suppressed for notational clarity when $q = 2$.

Other repetition codes have a partity submatrix P of similar form (a column of 1's); note that $P_{[n,1,n]} = P^T_{[n,n-1,2]}$.

In cyclic form, binary repetition codes are generated by $h(z) = z + 1$ and $v(z) = z^{n-1} + z^{n-2} + \ldots + z + 1$. For example, for $n = 4$, the H and V matrices for a binary single parity check code in cyclic form is given below left; these two matrices may be transformed back to the systematic form in (12.18) via (12.4) by taking $Q = I$, $S = I$, and (for any n) R as defined below right:

$$H^c_{[4,1,4]} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}, \quad V^c_{[4,1,4]} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}; \qquad R = \begin{pmatrix} 1 & \ldots & 1 \\ & \ddots & \vdots \\ 0 & & 1 \end{pmatrix}. \tag{12.19}$$

12.4 Binary Hamming codes and their extended and dual forms

The $[2^r - 1, 2^r - (r+1), 3]$ *binary Hamming codes* are perfect and SEC, and include $[3, 1, 3]$, $[7, 4, 3]$, $[15, 11, 3]$, $[31, 26, 3]$, $[63, 57, 3]$, $[127, 120, 3]$, $[255, 247, 3]$, etc. For a given $k = 2^r - (r+1)$ data bits to be transmitted, each of the r parity bits is generated such that the sum (on $\mathbf{F}_2$) of a particular subset of the data bits plus that parity bit is 0. The parity-check matrix H of a binary Hamming code has as columns all nonzero binary vectors of length $r = n - k$; when expressed in systematic form, the r columns of H corresponding to the identity matrix are shifted to the end, and the remaining k columns of H, in arbitrary order (often, binary order is used[10]), make up the parity submatrix P. For example, the $[7, 4, 3]$ code (see Figure 12.4), with four data bits $\{a_1, a_2, a_3, a_4\}$, three parity bits $\{b_1, b_2, b_3\}$, and $2^k = 16$ valid codewords $\mathbf{w}^i$, is given in systematic form by

$$H_{[7,4,3]} = \left(\underbrace{\begin{matrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{matrix}}_{P} \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} \right), \quad V_{[7,4,3]} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix}. \tag{12.20a}$$

$$W_{[7,4,3]} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

Other binary Hamming codes are built in the same manner.

In cyclic form, $[2^r - 1, 2^r - (r+1), 3]$ binary Hamming codes are generated by selecting $v(z)$ as an r'th-order polynomial that is a root of $(z^n - 1)$, where $n = 2^r - 1$, and by taking $h(z) = (z^n - 1)/v(z)$ on $\mathbf{F}_2$; for $r = 3$ through 7, $v(z)$ and $h(z)$ are listed in Table 12.4. For example, the $[7, 4, 3]$ code may be written in cyclic form (12.5) by taking $v(z) = z^3 + z + 1$ and $h(z) = z^4 + z^2 + z + 1$, and thus H and V take the form given below:

$$H^c_{[7,4,3]} = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}, \quad V^c_{[7,4,3]} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{12.20b}$$

[10] In (12.20a), binary order using big endian convention on the individual bits is used on the columns of the parity submatrix P, with the msb in the top row and the lsb in the bottom row; little endian convention may also be used [see §1.1.3 for further discussion].

By defining Q, R, and S in (12.4) such that

$$Q^c = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \qquad R^c = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \qquad S^c = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix},$$

it is seen that $H_{[7,4,3]} = (R^c)^T H^c_{[7,4,3]} Q^c$ and $V_{[7,4,3]} = (Q^c)^T V^c_{[7,4,3]} S^c$, thus demonstrating the equivalence of these two LBCs. In fact, it may be shown (see [Hill, 1986]) that Q, R, and S may always be found such that the cyclic and systematic forms of any $[2^r - 1, 2^r - (r+1), 3]$ LBC may be related by (12.4).

A binary Hamming code, with $d = 3$, can correct a single error in an unknown position (see §12.4.0.1). However, it can correct up to *two* erasures (cf. §12.3). A common application of this capability is in a **RAID 6** system for storage of large amounts of critical data. In such a system, data may be striped across n hard disk drives using a binary Hamming code; if any single drive fails, the data on it can be recovered using an appropriate parity check equation (that is, one of the parity check equations that takes that bit into account). If (while rebuilding the information on that disk, which might take a while) a *second* drive fails, then two useful equations may be derived (by linear combination on $\mathbf{F}_q^n$) from the r parity check equations: one that takes failed disk A into account but not failed disk B, and one that takes failed disk B into account but not failed disk A. By restoring parity in these two derived equations, the information on *both* drives may be rebuilt.

12.4.0.1 Syndrome based error correction of binary Hamming codes

The $r = n - k$ parity bits of a $[2^r - 1, 2^r - (r+1), 3]$ binary Hamming code may be used in a remarkably simple fashion to determine not only *whether* or not a received message $\hat{\mathbf{w}}$ has a single bit error (which is true, as usual, if $H\hat{\mathbf{w}} \neq 0$) but if it does, *which* bit contains the error. To see this, consider a code equivalent to a binary Hamming code in systematic form, but permuted such that the columns of the modified parity check matrix H^s appear in binary order. For example, the $[7, 4, 3]$ binary Hamming code (12.20a) may be permuted into this non-systematic order (where superscript s denotes syndrome form) as

$$H^s_{[7,4,3]} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}, \quad V^s_{[7,4,3]} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{w}^s = \begin{pmatrix} b_3 \\ b_2 \\ d_1 \\ b_1 \\ d_2 \\ d_3 \\ d_4 \end{pmatrix}. \tag{12.20c}$$

By examining the relation $\mathbf{w} = Q^s \mathbf{w}^s$, it is seen immediately that the $[7, 4, 3]$ code defined in systematic form in (12.20a) may be transformed into syndrome form in (12.20c) using (12.4) with a permutation matrix of

$$Q^s = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \Rightarrow \quad H^s_{[7,4,3]} = H_{[7,4,3]} Q^s, \quad V^s_{[7,4,3]} = (Q^s)^T V_{[7,4,3]}, \quad \mathbf{w}^s = (Q^s)^T \mathbf{w}, \tag{12.21}$$

thus demonstrating its equivalence. Note that each row of H^s so constructed adds to 0 (on $\mathbf{F}_2$), and of course that each row of H^s is orthogonal (on $\mathbf{F}_2$) to each column of V^s. Now define the product (on $\mathbf{F}_2$) of the matrix

H^s times any (possibly corrupted) received message $\hat{\mathbf{w}}^s$, arranged in the corresponding order (see above), as the *syndrome vector*

$$\mathbf{s} = H^s\hat{\mathbf{w}}^s = H^s(Q^s)^T\hat{\mathbf{w}} = H\hat{\mathbf{w}}. \tag{12.22}$$

If $\mathbf{s} = 0$, the message $\hat{\mathbf{w}}^s$ is interpreted as being error free (i.e., $\hat{\mathbf{w}}^s = \mathbf{w}^s$). If $\mathbf{s} \neq 0$, we may interpret $\mathbf{s}$ as an r-bit binary representation of a number called the *syndrome*, denoted s, of the received message $\hat{\mathbf{w}}^s$. Remarkably, as a direct result of the structure of H^s used in this construction [see, e.g., (12.20c)], the syndrome s identifies precisely which bit of the n-bit received message, $\hat{\mathbf{w}}^s$, must be flipped in order to determine the nearest codeword, thereby performing single error correction (SEC). This correction may be written as a correction vector $\mathbf{e}^s$, which is taken as 1 in the s element and zero in all others.

In practice, the systematic form [see, e.g., (12.20a)] of a binary Hamming code may still be preferred to code and check each data vector $\mathbf{d}$, due to its simplicity in decoding, thus transmitting a message $\mathbf{w} = V\mathbf{d}$ and receiving a corresponding (possibly corrupted) message $\hat{\mathbf{w}}$. In this case, by (12.22), the syndrome s is calculated from the received message $\hat{\mathbf{w}}$ using the original systematic form [in (12.2)] of the parity check matrix, H. The interpretation of the corresponding syndrome s, however, remains the same: that is, as identifying, with $\mathbf{e}^s$, which bit of received message $\hat{\mathbf{w}}^s$, written in syndrome form [see, e.g., (12.20c)], must be flipped to perform single error correction. This interpretation must thus be permuted (via $\mathbf{e} = P\mathbf{e}^s$) to determine a correction vector $\mathbf{e}$ for the corresponding received message $\hat{\mathbf{w}}$ written in systematic form.

Alternatively, (and, in many cases, faster, as discussed in the second paragraph of §12.2.3) the cyclic form (12.5a) of a binary Hamming code may be preferred to code and check each data vector $\mathbf{d}$. If the received message is found to have an error (that is, if $H^c\hat{\mathbf{w}}^c \neq 0$), one may simply permute the received message (in cyclic form) to syndrome form form via $\hat{\mathbf{w}}^s = Q^{sc}\hat{\mathbf{w}}^c$, where $Q^{sc} = (Q^s)^T(Q^c)^T$ is built from the permutation matrices Q^s and Q^c discussed above, compute the syndrome $\mathbf{s} = H^s\hat{\mathbf{w}}^s$ and the corresponding correction vector $\mathbf{e}^s$ in syndrome form as discussed above, then permute this correction vector back to cyclic form via $\mathbf{e}^c = (Q^{sc})^T\mathbf{e}^s$. In fact, these vector permutations can be hard wired into the implementation of the error correction code itself, and thus may be performed at essentially zero computational cost.

12.4.1 Extended binary Hamming codes

The $[2^{r-1}, 2^{r-1} - r, 4]$ *extended binary Hamming codes* are quasi-perfect and SECDED, and include $[4, 1, 4]$, $[8, 4, 4]$ (self-dual), $[16, 11, 4]$, $[32, 26, 4]$, $[64, 57, 4]$, $[128, 120, 4]$, $[256, 247, 4]$, etc. These codes may be formed simply as binary Hamming codes (see §12.4) with an additional overall parity bit b_p; that is, with an extra row in V such that each column of V adds to zero, and an extra row in H that checks the overall parity of the received message. To illustrate, the $[8, 4, 4]$ code (see Figure 12.5) is given in syndrome form by

$$H^s_{[8,4,4]} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \quad V^s_{[8,4,4]} = \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}, \quad \mathbf{w}^s = \begin{pmatrix} b_3 \\ b_2 \\ d_1 \\ b_1 \\ d_2 \\ d_3 \\ d_4 \\ b_p \end{pmatrix}. \tag{12.23a}$$

Other extended binary Hamming codes may be constructed similarly, by extending a $[2^r - 1, 2^r - (r + 1), 3]$ binary Hamming code in syndrome form with an extra row in V such that each column of V adds to zero, and an extra row in H that checks the overall parity of the received message.

The parity check of each received message is block decomposed into the syndrome $\mathbf{s}$, defined exactly as in (12.22) [neglecting the new parity bit b_p], and the overall parity calculation p such that

$$\begin{bmatrix} \mathbf{s} \\ p \end{bmatrix} = H^s \hat{\mathbf{w}}^s \quad (\text{on } \mathbf{F}_q).$$

Assuming no more than two bit errors, the syndrome $\mathbf{s}$ and overall parity p are interpreted as follows:

- If $\mathbf{s} = 0$ and $p = 0$, the received message is error free.
- If $\mathbf{s} \neq 0$ and $p = 0$, the received message has two bit errors, which can not be uniquely corrected.
- If $p = 1$, the received message has a single bit error, which if $\mathbf{s} = 0$ is simply in the parity bit, and if $\mathbf{s} \neq 0$ can be located within the rest of the message using $\mathbf{s}$, precisely as laid out in §12.4.0.1.

This algorithm thus efficiently performs single error correction and double error detection (SECDED).

Note that the syndrome form (with additional overall parity) of an extended binary Hamming code, as in (12.23a), can be converted back to systematic form by replacing the last row of H^s with the sum (on $\mathbf{F}_q$) of all of the rows of H^s, then permuting the first $(n-1)$ columns of H^s and the first $(n-1)$ rows of V^s consistent with §12.4.0.1. For example, using Q^s as defined in (12.21), the $[8,4,4]$ extended binary Hamming code given in (12.23a) may be transformed back to systematic form via (12.4) as follows

$$\bar{Q} = \begin{pmatrix} (Q^s)^T & 0 \\ 0 & 1 \end{pmatrix}, \quad \bar{R} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \Rightarrow \quad H_{[8,4,4]} = \bar{R}^T H^s_{[8,4,4]} \bar{Q}, \quad V_{[8,4,4]} = \bar{Q}^T V^s_{[8,4,4]}, \quad \mathbf{w} = \bar{Q}^T \mathbf{w}^s,$$

which gives

$$H_{[8,4,4]} = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ \underbrace{1 & 1 & 1 & 0}_{P} & 0 & 0 & 0 & 1 \end{pmatrix}, \quad V_{[8,4,4]} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}. \tag{12.23b}$$

12.4.2 Binary simplex codes

The dual of the binary Hamming codes are the $[2^k - 1, k, 2^{k-1}]$ *binary simplex codes*, which include $[3,2,2]$ (SED), $[7,3,4]$ (SECDED), $[15,4,8]$ (TECQED), $[31,5,16]$ (5EC6ED), etc. The codewords of these LCs form a simplex with 2^k vertices in $2^k - 1$ dimensions; this simplex is regular in the $[3,2,2]$ case, but irregular in the other cases. The $[3,2,2]$ code is illustrated in Figure 12.1a; the $[7,3,4]$ code is given by

$$H_{[7,3,4]} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ \underbrace{1 & 1 & 1}_{P} & 0 & 0 & 0 & 1 \end{pmatrix}, \quad V_{[7,3,4]} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}. \tag{12.24}$$

Other binary simplex codes have a partity submatrix given similarly by the transpose of the corresponding binary Hamming code.

12.4.3 Binary biorthogonal codes

The dual of the extended binary Hamming codes are the $[2^m, m+1, 2^{m-1}]$ *binary biorthogonal codes* (a.k.a. *Hadamard codes*), and include $[4,3,2]$ (SED; see §12.3), $[8,4,4]$ (SECDED, quasi-perfect, self-dual; see §12.4.1), $[16,5,8]$ (TECQED), $[32,6,16]$ (5EC6ED), etc. The $[32,6,16]$ code was used on the Mariner 9 spacecraft. The codewords of these LCs are mutually orthogonal [that is, $\mathbf{w}^i \cdot \mathbf{w}^j = 0$ (on $\mathbf{F}_2$) for $i \neq j$]. The binary biorthogonal codes each have a partity submatrix that is simply the transpose of the parity submatrix of the corresponding extended binary Hamming code.

12.5 Binary quadratic residue codes

The $[n, (n+1)/2, d]$ *binary quadratic residue codes* are defined for all prime n for which there exists an integer $1 < x < n$ such that $x^2 = 2 \pmod{n}$ [equivalently, for all prime n of the form $n = 8m \pm 1$ where m is an integer], and include $[7,4,3]$ (SEC, perfect; see §12.4), $[17,9,5]$ (DEC), $[23,12,7]$ (TEC, perfect, a.k.a. the *binary Golay code*), $[31,16,7]$ (TEC), $[41,21,9]$ (4EC), $[47,24,11]$, etc. Adding an overall parity bit to these codes, the $[n_0+1, (n_0+1)/2, d+1]$ *extended binary quadratic residue codes* include $[8,4,4]$ (SECDED, quasi-perfect, self-dual; see §12.4.1), $[18,9,6]$ (DECTED), $[24,12,8]$ (TECQED, quasi-perfect, self-dual, a.k.a. the *extended binary Golay code*), $[32,16,8]$ (TECQED), $[42,21,10]$ (4EC5ED), $[48,24,12]$ (5EC6ED, self-dual), etc. The $[24,12,8]$ extended binary Golay code, used by the Voyager 1 & 2 spacecraft, is given by

$$H_{[24,12,8]} = \begin{bmatrix} P_{12\times 12} & I_{12\times 12} \end{bmatrix}, \quad V_{[24,12,8]} = \begin{bmatrix} I_{12\times 12} \\ P_{12\times 12} \end{bmatrix}, \quad P_{12\times 12} = \begin{pmatrix} 0&1&1&1&1&1&1&1&1&1&1&1 \\ 1&1&1&0&1&1&1&0&0&0&1&0 \\ 1&1&0&1&1&1&0&0&0&1&0&1 \\ 1&0&1&1&1&0&0&0&1&0&1&1 \\ 1&1&1&1&0&0&0&1&0&1&1&0 \\ 1&1&1&0&0&0&1&0&1&1&0&1 \\ 1&1&0&0&0&1&0&1&1&0&1&1 \\ 1&0&0&0&1&0&1&1&0&1&1&1 \\ 1&0&0&1&0&1&1&0&1&1&1&0 \\ 1&0&1&0&1&1&0&1&1&1&0&0 \\ 1&1&0&1&1&0&1&1&1&0&0&0 \\ 1&0&1&1&0&1&1&1&0&0&0&1 \end{pmatrix}. \tag{12.25}$$

Note that P is symmetric. The $[23,12,7]$ binary Golay code may be obtained by *puncturing* the $[24,12,8]$ code listed above; that is, by eliminating any row of P (typically, the last).

12.6 Puncturing/extending, augmenting/expurgating, and shortening/lengthening

For a given alphabet q, there are six essential operations that may be applied to any $[n, k, d]_q$ LC, with $r = n-k$:

Puncturing:	fix k, reduce r,	reduce n	(eliminate check symbols)
Extending:	fix k, increase r,	increase n	(add check symbols)
Augmenting:	fix n, reduce r,	increase k	(eliminate check symbols and add data symbols)
Expurgating:	fix n, reduce k,	increase r	(eliminate data symbols and add check symbols)
Shortening:	fix r, reduce k,	reduce n	(eliminate data symbols)
Lengthening:	fix r, increase k,	increase n	(add data symbols)

A code obtained by eliminating a check symbol, and thus reducing both r and n by 1, is said to be a **punctured** code, and (conversely) a code obtained by adding a check symbol is said to be an **extended** (a.k.a. **expanded**) code. If starting from a perfect (or, nearly perfect) LC, puncturing it by 1 symbol will typically also reduce d by 1. For example, as seen in §12.4-12.4.1 and §12.5, the (perfect) binary Hamming and binary Golay codes may be extended to quasi-perfect codes by adding an overall parity bit, thereby increasing n by 1 (and, in the case of these specific codes, increasing d by 1 as well); conversely, the corresponding quasi-perfect codes may be punctured to create the corresponding perfect codes by removing the overall parity bit.

In contrast, a code obtained by simultaneously eliminating a check symbol and adding a data symbol is said to be an **augmented** code, and (conversely) a code obtained by simultaneously eliminating a data symbol and adding a check symbol is said to be an **expurgated** code. An example of this is the [7,3,4] code, which may be augmented to form the [7,4,3] code; conversely, the [7,4,3] code may be expurgated to form the [7,3,4] code.

Finally, a code obtained by eliminating a data symbol, and thus reducing both k and n by 1, is said to be a **shortened** code, and (conversely) a code obtained by adding a data symbol, and thus increasing k and n by 1, is said to be a **lengthened** code. Shortening an LC leaves both $r = n - k$ and d unchanged, but reduces the data rate k/n. Shortening is useful for developing LBCs for error-correcting memory systems, in which the data comes naturally in blocks of 8, 16, 32, 64, 128, or 256 bits (see §1.2). In particular,

- starting from $[15, 11, 3]$ or $[16, 11, 4]$, eliminating 3 data bits creates the shortened $[12, 8, 3]$ or $[13, 8, 4]$ LBCs,
- starting from $[31, 26, 3]$ or $[32, 26, 4]$, eliminating 10 data bits creates the $[21, 16, 3]$ or $[22, 16, 4]$ LBCs,
- starting from $[63, 57, 3]$ or $[64, 57, 4]$, eliminating 25 data bits creates the $[38, 32, 3]$ or $[39, 32, 4]$ LBCs,
- starting from $[127, 120, 3]$ or $[128, 120, 4]$, eliminating 56 data bits creates the $[71, 64, 3]$ or $[72, 64, 4]$ LBCs,
- starting from $[255, 247, 3]$ or $[256, 247, 4]$, eliminating 119 data bits creates the $[136, 128, 3]$ or $[137, 128, 4]$ LBCs,
- starting from $[511, 502, 3]$ or $[512, 502, 4]$, eliminating 246 data bits creates the $[265, 256, 3]$ or $[266, 256, 4]$ LBCs.

Most ECC memory and RAID 6 storage systems are based on one of these shortened LBCs[11], which are SEC (if $d = 3$) or SECDED (if $d = 4$), and are both simple and fast to use.

The operations of shortening and puncturing are perhaps the most important of the six operations discussed above; as highlighted in Figure 12.3 for $1 \leq d \leq 12$ and $1 \leq k \leq 64$, the densest LBCs available may be obtained simply by shortening and/or puncturing a few exemplary LBCs, many of which are defined in §12.3-12.5.

A shortened LC can be coded, decoded, and (if necessary) corrected using essentially the same algorithms available to code, decode, and correct the LC before it was shortened, with the unused data symbols simply set to zero (and, thus, not actually transmitted over the channel), and the corresponding symbols on the other end of the channel asserted to be zero (and, error-free).

[11] Note that the perhaps peculiar name "shortened extended binary Hamming code" means precisely a binary Hamming code that has been *extended* by adding an overall parity bit, and then *shortened* by removing some data bits.

Algorithm 12.1: Main loop of a binary CRC code implementing (12.13); full code available at RR_CRC_encode.m.

```
t=bitshift(a,r);  % initialize t corresponding to t(z) = z^r * a(z)
for  i=n:-1:r+1   % zero coefficient i in t(z) by subtracting shift of v(z)
  if bitget(t,i), t=bitxor(t,bitshift(v,i-r-1)); end
end
b=dec2bin(t,r); whos
```

12.7 Binary Cyclic Redundancy Check (CRC) codes

As laid out in §12.2, LCs in systematic cyclic form are defined by basis polynomials $v(z) = v_r z^r + \ldots + v_1 z + v_0$, and/or corresponding parity check polynomials $h(z) = h_k z^k + \ldots + h_1 z + h_0$, defined mutually such that $h(z)\,v(z) = z^n - 1 = 0$, for a given r, k, and n such that $r + k = n$.

Binary cyclic codes are also commonly implemented in a cyclic redundancy check (CRC) setting, in which the number of data bits k to be transmitted is actually *variable*. In standardized applications (USB, bluetooth, ethernet, etc), r and $v(z)$ are predefined by the particular version of the standard being used (see §4), and k is restricted to particular values within a certain range (e.g., from 1 B to 16 KiB). Binary cyclic redundancy check codes are commonly denoted CRC-rA, where $r = n - k$ is the number of parity bits used (typically, $r = 4$ to 32), and A is a set of letters and numbers used to identify that CRC code.

A CRC code is defined by its basis polynomial $v(z)$. For example, the CRC-32 code used by ethernet is

$$v(z) = z^{32} + z^{26} + z^{23} + z^{22} + z^{16} + z^{12} + z^{11} + z^{10} + z^8 + z^7 + z^5 + z^4 + z^2 + z + 1.$$

It is helpful to use a shorthand[12] to define the polynomial $v(z)$. Treating the coefficients of all but the constant $[z^0]$ term as a binary number, we can write these coefficients as $1000\,0010\,0110\,0000\,1000\,1110\,1101\,1011_2$ in this case, or in hexadecimal as 0x82608edb, as listed in the corresponding (802.3) entry in Table 12.5. Additional shorthand examples and their corresponding basis polynomials are given in the first 8 rows of Table 12.5.

The CRC approach takes a block of data $a(z)$ [with k bits], and applies a binary version of (12.13), leveraging the basis polynomial $v(z)$ implemented, to determine the $b(z)$ [with r bits] of the corresponding codeword $w(z)$ [with $n = k + r$ bits] to be transmitted [see (12.11)]. The r bits of $b(z)$ may be determined by the (easy-to-read) code illustrated in Algorithm 12.1; in practice, this algorithm should be implemented either in an ASIC (§1.5.3.3), or with a carefully-tuned (but more difficult to read) low-level (C or assembly) code. For example, applying this approach to the data vector $1101\,1010\,0011\,1101_2$ = 0xda3d using the CRC-5-USB [aka HAM(31,26,3)] code, $v(z) = z^5 + z^2 + 1$ [aka 0x12], each nontrivial step of the binary XOR in this calculation is:

t = 11011010001111010000
v = 10010100000000000000
t = 01001110001111010000
v = 01001010000000000000
t = 00000100001111010000
v = 00000100101000000000
t = 00000000101111010000
v = 00000000100101000000
t = 00000000000100100000
v = 00000000000100101000
t = 00000000000000001000

The $r = 5$ parity bits to be transmitted [i.e., the binary coefficients of $b(z)$] in this case are thus 01000.

[12] The version of the shorthand notation used in this text is referred as **implicit+1** or **reversed reciprocal** notation.

r	nickname	shorthand	basis polynomial $v(z)$	maximum k for a given d $d=3$ (SEC),	4 (SECDED),	5 (DEC),	6,	7
4	HAM(15, 11, 3)	0x9	z^4+z+1	11				
5	HAM(31, 26, 3)	0x12	z^5+z^2+1	26				
	CRC-5-ITU	0x15	z^5+z^3+z+1	→	10			
8	HAM(255, 247, 3)	0x8e	$z^8+z^4+z^3+z^2+1$	247	13	6		
	CRC-8F/3	0xe7	$z^8+z^7+z^6+z^3+z^2+z+1$	247	19	→	→	1
	CRC-8P	0x83	z^8+z^2+z+1	→	119			
	CRC-8F/5	0xeb	$z^8+z^7+z^6+z^4+z^2+z+1$	→	→	9	2	1
	BCH(15, 7, 5)	0xe8	$z^8+z^7+z^6+z^4+1$	→	→	7		

r	nickname	shorthand	maximum k for a given Hamming distance d $d=3$ (SEC),	4 (SECDED),	5 (DEC),	6,	7,	8,	9,	10
16	HAM(65535, 65519, 3)	0x8016	65519	551	40					
	CRC-16F/3	0x8d95	65519	1149	62	19	9	→	5	
	CRC-16K/5	0x9627	65519	390	119	15	11	→	4	
	CRC-16K/6	0x86f2	65519	83	53	40	→	6	5	
	CRC-16K/7.2	0xb82d	65519	221	22	→	18	→	2	
	C5	0xd175	→	32751	→	93	→	11	→	2
	C3	0xac9a	→	→	241	35	10	8	3	
	BCH(255, 239, 5)	0xb7b1	→	→	239	14	13	→	→	2
	C1	0x9eb2	→	→	→	135	→	6	→	4
	C4	0x808d	→	28642	→	99				
24	HAM(2^{24}–1, 2^{24}–25, 3)	0x80000d	16777191	5815	509					
	CRC-24K/3.2	0x8f90e3	16777191	22868	599	47	37	33	12	10
	CRC-24K/3.3	0x93cb4f	16777191	8880	1060	52	39	19	→	9
	CRC-24K/3.4	0xa2e4ce	16777191	2562	457	248	70	30	11	9
	CRC-24K/4	0x9945b1	→	8388583	→	822	→	37	→	12
	CRC-24K/5.1	0x98ff8c	→	→	4073	228	13	→	→	9
	BCH(4095, 4071, 5)	0xa0efce	→	→	4071	121	54	→	24	9
	CRC-24K/6.2	0xbd80de	→	4074	→	2026	→	59	→	12
	BCH(255, 231, 7)	0xddd0da	→	→	→	→	231	9	8	2
	CRC-24K/6sub8	0x80009a	→	8388583	→	667				
32	HAM(2^{32} – 1, 2^{32} – 33, 3)	0x8C000001	4294967263	76947	2210					
	CRC-32 (802.3)	0x82608edb	4294967263	91607	2974	268	171	91	57	34
	CRC-32K/3.1	0xad0424f3	4294967263	427053	817	522	149	63	27	19
	CRC-32K/3.2	0x9d9947fd	4294967263	146826	8162	361	145	60	56	33
	CRC-32K/3.3	0x8e2371ef	4294967263	118103	2438	1199	201	66	45	38
	CRC-32K/4.2	0xc9d204f5	→	2147483615	→	6167	→	148	→	44
	CRC-32/5.1	0xd419cc15	→	→	65505	1060	81	58	→	27
	BCH(65535, 65503, 5)	0x80af10a3	→	→	65503	501	157	62	55	26
	CRC-32K/6.4	0x9960034c	→	65506	→	32738	→	193	→	31
	CRC-32K/6sub8	0x80000072	→	1761607438	→	4113				
36	HAM(2^{36} – 1, 2^{36} – 37, 3)	0x800000400	68719476699							
	BCH(262143, 262107, 5)	0x820843820	→	→	262107	1114	→	36	6	
	BCH(4095, 4059, 7)	0xa21e33520	→	→	→	→	4059	191	78	29
	BCH(511, 475, 9)	0xe615cc4d0	→	→	→	→	→	→	475	46

Table 12.5: Performance of various cyclic codes. Entries indicated with '→' represent no improvement over the entry for next larger value of d. Additional columns, for $d > 10$ and reduced k, are (to save space) not shown. Tabulated data and related software (used for calculating d for HAM and BCH codes) courtesy of Koopman.

Algorithm 12.2: Binary code for computing $h(z)$ from $v(z)$; full code available at RR_CRC_h_from_v.m.

```
t=bitshift(1,n)+0b1u64; h=0b0u64; % initialize t=100...001 corresponding to t(z)=z^n-1
for i=n+1:-1:r+1                  % calculate  h=t/v on F2
  if bitget(t,i)
    h=h+bitshift(1,i-r-1); t=bitxor(t,bitshift(v,i-r-1));
  end
end
```

Table 12.5 quantifies the performance (that is, the maximum number of data bits k for a given Hamming distance d, up to[13] $d = 10$) of various cyclic codes. Maximum values of k highlighted in red in the Table are optimal amongst all CRC codes for that value of r and d; amongst all CRC codes with the indicated (in red) optimal k values for that r and d, tuned secondary k values (for different d) are highlighted in blue. Note that k values highlighted in green are optimal, for $d = 6$ and that value of r, amongst all 6sub8 CRC codes (that is, cyclic codes with a basis polynomial $v(z)$ that is zero except for its leading bit and its last 8 bits, which significantly streamlines certain numerical implementations).

Noting Table 12.5 and Koopman, for a given $r \leq 32$, the highest data-rate (largest k) CRC codes are given,

in the $d = 3$ (SEC)	case for $r \geq 4$,	by $[2^r - 1, 2^r - (r+1), 3]$ binary Hamming codes [see §12.4],
in the $d = 4$ (SECDED)	case for $r \geq 4$,	by $[2^{r-1} - 1, 2^{r-1} - (r+1), 4]$ LBCs [cf. §12.4.1],
in the $d = 5$ (DEC)	case for even $r \geq 8$,	by $[2^{r/2} + 1, 2^{r/2} - (r-1), 5]$ LBCs, and
in the $d = 6$ (DECTED)	case for $r \geq 17$,	by $[2^{r_2} - 2\,r_2 + r, 2^{r_2} - 2\,r_2, 6]$ LBCs where $r_2 = \lfloor (r-1)/2 \rfloor$.

Remarkably, as seen in Table 12.5, there is sufficient freedom in the ordering of a CRC LBC design in cases that are optimal in the SEC setting (as highlighted in red in the $d = 3$ column), which are equivalent to binary Hamming codes[14], to simultaneously tune its performance at some higher d (as highlighted in blue), for situations in which data packets with substantially shorter k, but using the same $v(z)$, are also to be encountered in practice.

Often, CRC codes are used for error *detection* only; the same ASIC or low-level code may be used on the receiving end of the transmission to recalculate the r parity bits from the k data bits received, and an error is flagged (and, retransmission of that particular packet requested) if the received and recalculated parity bits don't match[15]; this approach can detect up to $d - 1$ random bit errors. Alternatively, error *correction* may be performed by determining the closest valid codeword to the received transmission. As discussed in §12.1, for odd d, an error correcting code can correct up to $(d-1)/2$ bit errors, and, for even d, an error correcting code can both correct up to $d/2 - 1$ bit errors and (simultaneously) detect but not correct $d/2$ bit errors.

Error correction in the CRC setting can be performed using the cyclic form of the parity check matrix H [see (12.5a)] as before, which is built on the coefficients of $h(z)$, which is determined from $v(z)$ such that $h(z)\,v(z) = z^n - 1$, as shown in Algorithm 12.2.

[13] Additional columns are less important when selecting a CRC scheme for practical use with $r \geq 16$, which typically use $k > r$.

[14] Note that binary Hamming codes are perfect, which implies that they really can't be tweaked very much without reducing the value of d (from $d = 3$) for a given r and a given optimal value of k. However, they can still be modified according to (12.4), which can have a significant effect on the maximum k for a given $d > 3$ when a reduced number of bits are to be transmitted.

[15] Equivalently, the (possibly corrupted) received message $\hat{w}(z)$, with the parity bits $b(z)$ (instead of r zeros) appended [see (12.11)] may be fed directly into this ASIC or low-level code, with an error being flagged if a zero is not returned.

12.8 Binary BCH Codes

The Bose, Chaudhuri, Hocquenghem (BCH) family of cyclic LCs have $n = 2^m - 1$, $r \leq mt$, and $d \geq 2t + 1$, where $m \geq 3$ and $1 \leq t < 2^{m-1}$. We focus on binary ($q = 2$) BCH codes in the discussion that follows, though the BCH family generalizes to other q. The SEC ($t = 1$, $d = 3$, $r = m$) binary BCH codes are equivalent to the cyclic $[2^r - 1, 2^r - (r + 1), 3]$ binary Hamming codes discussed previously.

Binary BCH codes generalize binary Hamming codes to higher d [see (12.27)] while maintaining a cyclic form. Their construction is presented in §12.8.1. The 32 smallest redundancy [$r \leq 36$, perhaps the "most useful"] (n, k, d) binary BCH codes with $t > 1$, which do not correspond to codes discussed previously in this chapter, are listed below in the same implicit+1 shorthand format introduced in §12.7:

- the DEC ($t = 2$) BCH codes, $(2^m - 1, 2^m - 1 - 2m, 5)$ for $m = \{4, 5, \ldots, 18\}$, $r = \{8, 10, \ldots, 36\}$:
  (15,7,5)=0xe8, (31,21,5)=0x3b4, (63,51,5)=0xa9c, (127,113,5)=0x2a3e, (255,239,5)=0xb7b1, (511,493,5)=0x24ae4, (1023,1003,5)=0x80c3b, (2047,2025,5)=0x2482d8, (4095,4071,5)=0xa0efce, (8191,8165,5)=0x26a8aa5, (16383,16355,5)=0x92dfcf5, (32767,32737,5)=0x21080632, (65535,65503,5)=0x80af10a3, (131071,131037,5)=0x20006003b, (262143,262107,5)=0x820843820;
- the TEC ($t = 3$) BCH codes for $m = \{4, 5, 6, 7, 8, 9, 10, 11, 12\}$, $r = \{10, 15, 18, 21, 24, 27, 30, 33, 36\}$:
  (15,5,7)=0x29b, (31,16,7)=0x47d7, (63,45,7)=0x3c167, (127,106,7)=0x14980d, (255,231,7)=0xddd0da, (511,484,7)=0x6b095bc, (1023,993,7)=0x28548889, (2047,2014,7)=0x137c5373e, (4095,4059,7)=0xa21e33520;
- the 4EC ($t = 4$) BCH codes for $m = \{6, 7, 8, 9\}$, $r = \{24, 28, 32, 36\}$:
  (63,39,9)=0xed93bb, (127,99,9)=0xc52bc9f, (255,223,9)=0xf72da17e, (511,475,9)=0xe615cc4d0;
- the 5EC ($t = 5$) BCH codes for $m = \{5, 6, 7\}$, $r = \{20, 27, 35\}$:
  (31,11,11)=0xb136a, (63,36,11)=0x4374089, (127,92,11)=0x708e54dab;
- the 6EC ($t = 6$) BCH code for $m = \{6\}$, $r = \{33\}$:
  (63,30,13)=0x1be6875b3.

Note that the maximum data rate k/n of these codes *increases* quickly as m (and thus both r and $k = n - r$) are increased for a given t, but the maximum data rate k/n *decreases* as t (and thus the Hamming distance d) is increased for a given m. Following the constructive process in §12.8.1, efficient binary BCH codes with large n and t may easily be generated; for example,

- the $m = 10$ BCH codes for $t = \{3, 4, 5, 6, 12, 26, 57, 106\}$, $r = \{30, 40, 50, 60, 120, 255, 510, 765\}$:
  (1023,993,7)=0x28548889, (1023,983,9), (1023,973,11), (1023,963,13), (1023,903,25), (1023,768,53), (1023,573,101), (1023,258,213).

Note also that the performance (that is, the maximum k for a given Hamming distance d) for the 8 codes highlighted above in magenta is provided in full in Table 12.5. Many other BCH codes with large n and t may also be generated (see, e.g., Table 6.1 of Lin & Costello 1983).

The reason binary BCH codes are interesting is that, as generalizations of Hamming codes, fast error correction algorithms may be constructed for them.

As seen in Table 12.5 and comparing with the corresponding entries highlighted in red, the performance of BCH codes are seen to be nearly optimal at the Hamming distances for which they were designed. This fact is quite remarkable, given the convenient structure of BCH codes from the perspective of error correction.

In summary, at $d = 3$, cyclic binary Hamming codes are perfect, and error correction may be performed easily if desired (see §12.4.0.1). For higher d using a cyclic LBC, if performing error *detection* only, one of the optimal entries highlighted in red in Table 12.5 should be selected, whereas if performing error *correction*, a BCH code, such as one of those listed above, should generally be selected instead.

12.8.1 Construction of binary BCH codes†

As described in §12.8, a binary BCH code has $n = 2^m - 1$, $r \leq mt$, and $d \geq 2t + 1$, where m and t are small positive integers. In this section, we describe the framework used to identify the binary BCH codes.

12.8.1.1 Primitive polynomials

To begin, we first define an **irreducible** polynomial as a polynomial that can not be factored as the product of other (lower-order) polynomials with coefficients of the same type (in this case, binary).

We then define a **primitive** polynomial $f(z)$ as an irreducible polynomial of degree m such that, for any root α of $f(z)$, the first $n = 2^m - 1$ powers of α (namely, $\beta_0 = \alpha^0$ to $\beta_{n-1} = \alpha^{n-1}$) generate all n of the n'th roots of unity (that is, this process generates all of the distinct β such that $\beta^n - 1 = 0$).

The test for whether or not an irreducible polynomial is primitive is perhaps best illustrated by two examples. In each example, we denote α as a zero of the polynomial considered, and simplify the expressions of the first n powers of α using the polynomial itself. For primitive polynomials like $f(z) = z^4 + z + 1$ (restricting our attention here, for simplicity, to the case with binary coefficients, with $m = 4$ and thus $n = 15$), all n'th roots of unity are reached via this process; for non-primitive polynomials like $g(z) = z^4 + z^2 + 1$, this fails to be true:

primitive example: $f(z) = z^4 + z + 1$		**non-primitive example**: $g(z) = z^4 + z^2 + 1$	
α^p mod $f(\alpha)$	binary representation	α^p mod $g(\alpha)$	binary representation
$\alpha^0 \rightarrow 1$	0001	$\alpha^0 \rightarrow 1$	0001
$\alpha^1 \rightarrow \alpha$	0010	$\alpha^1 \rightarrow \alpha$	0010
$\alpha^2 \rightarrow \alpha^2$	0100	$\alpha^2 \rightarrow \alpha^2$	0100
$\alpha^3 \rightarrow \alpha^3$	1000	$\alpha^3 \rightarrow \alpha^3$	1000
$\alpha^4 \rightarrow \alpha + 1$	0011	$\alpha^4 \rightarrow \alpha^2 + 1$	0101
$\alpha^5 \rightarrow \alpha(\alpha^4) = \alpha^2 + \alpha$	0110	$\alpha^5 \rightarrow \alpha(\alpha^4) = \alpha^3 + \alpha$	1010
$\alpha^6 \rightarrow \alpha(\alpha^5) = \alpha^3 + \alpha^2$	1100	$\alpha^6 \rightarrow \alpha^4 + \alpha^2 = 1$	0001
$\alpha^7 \rightarrow \alpha^4 + \alpha^3 = \alpha^3 + \alpha + 1$	1011	$\alpha^7 \rightarrow \alpha \cdot \alpha^6 = \alpha$	0010
$\alpha^8 \rightarrow \alpha^4 + \alpha^2 + \alpha = \alpha^2 + 1$	0101	$\alpha^8 \rightarrow \alpha^2$	0100
$\alpha^9 \rightarrow \alpha^3 + \alpha$	1010	$\alpha^9 \rightarrow \alpha^3$	1000
$\alpha^{10} \rightarrow \alpha^4 + \alpha^2 = \alpha^2 + \alpha + 1$	0111	$\alpha^{10} \rightarrow \alpha^4 = \alpha^2 + 1$	0101
$\alpha^{11} \rightarrow \alpha^3 + \alpha^2 + \alpha$	1110	$\alpha^{11} \rightarrow \alpha^5 = \alpha^3 + \alpha$	1010
$\alpha^{12} \rightarrow \alpha^4 + \alpha^3 + \alpha^2 = \alpha^3 + \alpha^2 + \alpha + 1$	1111	$\alpha^{12} \rightarrow \alpha^6 = 1$	0001
$\alpha^{13} \rightarrow \alpha^4 + \alpha^3 + \alpha^2 + \alpha = \alpha^3 + \alpha^2 + 1$	1101	$\alpha^{13} \rightarrow \alpha \cdot \alpha^{12} = \alpha$	0010
$\alpha^{14} \rightarrow \alpha^4 + \alpha^3 + \alpha = \alpha^3 + 1$	1001	$\alpha^{14} \rightarrow \alpha^2$	0100
$\alpha^{15} \rightarrow \alpha^4 + \alpha = 1$	0001	$\alpha^{15} \rightarrow \alpha^3$	1000

(12.26)

The process of computing α^p mod $f(\alpha)$ is automated in RR_Binary_Field_Mod.m.

Note next that, for example, the polynomial $z^n + 1$ (recall that "-1" is 1 in binary) for $m = 6$ and thus $n = 63$ may be expressed as the product [on GF(2^m)] of the following 13 irreducible polynomial factors (calculated using the Cantor-Zassenhaus algorithm):

$$z^{63} + 1 = (z+1)(z^2+z+1)(z^3+z+1)(z^3+z^2+1)(z^6+z^3+1)(z^6+z^4+z^2+z+1)(z^6+z^5+z^4+z^2+1)\cdot$$
$$(z^6+z+1)(z^6+z^4+z^3+z+1)(z^6+z^5+1)(z^6+z^5+z^2+z+1)(z^6+z^5+z^3+z^2+1)(z^6+z^5+z^4+z+1)$$

The last 6 terms in this factorization, highlighted in red and blue, are the $m = 6$ primitive polynomials. The other 3 polynomials of order 6 in this factorization are *not* primitive, as they fail the test illustrated in (12.26). For $m = \{1, 2, 3, 4, 5, 6, 7, \ldots\}$, the number of primitive polynomials follows the OEIS A011260 sequence: $\{1, 1, 2, 2, 6, 6, 18, 16, 48, 60, 176, 144, 630, 756, 1800, 2048, \ldots\}$. For BCH code generation, at any m, it is customary to use a primitive polynomial with the smallest number of terms and, amongst these, that polynomial with the smallest powers in the terms after the first, as highlighted in red above. Such a primitive polynomial is said to be **minimal**, and may be found by exhaustive search. The first 50 minimal primitive polynomials for binary fields GF(2^m) are listed in Table 12.6; for practical BCH code development, these should be all you need.

$z+1$	z^2+z+1	z^3+z+1	z^4+z+1	z^5+z^2+1
z^6+z+1	z^7+z+1	$z^8+z^4+z^3+z^2+1$	z^9+z^4+1	$z^{10}+z^3+1$
$z^{11}+z^2+1$	$z^{12}+z^6+z^4+z+1$	$z^{13}+z^4+z^3+z+1$	$z^{14}+z^5+z^3+z+1$	$z^{15}+z+1$
$z^{16}+z^5+z^3+z^2+1$	$z^{17}+z^3+1$	$z^{18}+z^7+1$	$z^{19}+z^6+z^5+z+1$	$z^{20}+z^3+1$
$z^{21}+z^2+1$	$z^{22}+z+1$	$z^{23}+z^5+1$	$z^{24}+z^4+z^3+z+1$	$z^{25}+z^3+1$
$z^{26}+z^8+z^7+z+1$	$z^{27}+z^8+z^7+z+1$	$z^{28}+z^3+1$	$z^{29}+z^2+1$	$z^{30}+z^{16}+z^{15}+z+1$
$z^{31}+z^3+1$	$z^{32}+z^{28}+z^{27}+z+1$	$z^{33}+z^{13}+1$	$z^{34}+z^{15}+z^{14}+z+1$	$z^{35}+z^2+1$
$z^{36}+z^{11}+1$	$z^{37}+z^{12}+z^{10}+z^2+1$	$z^{38}+z^6+z^5+z+1$	$z^{39}+z^4+1$	$z^{40}+z^{21}+z^{19}+z^2+1$
$z^{41}+z^3+1$	$z^{42}+z^{23}+z^{22}+z+1$	$z^{43}+z^6+z^5+z+1$	$z^{44}+z^{27}+z^{26}+z+1$	$z^{45}+z^4+z^3+z+1$
$z^{46}+z^{21}+z^{20}+z+1$	$z^{47}+z^5+1$	$z^{48}+z^{28}+z^{27}+z+1$	$z^{49}+z^9+1$	$z^{50}+z^{27}+z^{26}+z+1$

Table 12.6: Minimal primitive polynomials for binary fields GF(2^m) for orders $m = 1$ through 50.

12.8.1.2 Minimal polynomials with $\beta_p = \alpha^p$ as a root

Given a primitive polynomial $f(z)$ of order m, and denoting α as a root of this polynomial, we next seek to find the polynomial of lowest order with $\beta_p = \alpha^p$ as a root, dubbed the **minimal polynomial** $\phi_p(z)$, for various powers p between 0 and $n-1$. Of course, since $f(z)$ is irreducible, the minimal polynomial for $p = 1$ is just the primitive polynomial itself, $\phi_1(z) = f(z)$. For larger p, the corresponding minimal polynomial may be found by expressing β_p^q, for $q = 1$ to m, in terms of the α^i for $i = 0$ to $m-1$, following the general approach demonstrated in (12.26) [using, e.g., RR_Binary_Field_Mod.m], then using the results to rewrite the equation

$$c_0 + c_1\beta_p + c_2\beta_p^2 + \ldots + c_{m-1}\beta_p^{m-1} + c_m\beta_p^m = 0$$

in terms of the α_i (for $i = 0$ to $m-1$). Setting the coefficients of like powers of α equal to zero in the resulting equation leads to a homogeneous set of m linear equations in the $m+1$ unknowns $\mathbf{x} = \{c_0, c_1, \ldots, c_m\}$, which may be written in the form $A\mathbf{x} = 0$. This underdetermined binary system has one or more nontrivial solutions, the smallest of which may be found by transforming A to reduced echelon form R [see *NR* §2.6] using binary arithmetic, then setting the coefficient c_i corresponding to the first non-pivot column of equal to 1, and the coefficients corresponding to all other non-pivot columns (if any) equal to 0, and solving the remaining nonsingular system for the remaining coefficients c_i. The resulting minimal polynomial may then be written as

$$\phi_p(z) = c_0 + c_1 z + c_2 z^2 + \ldots + c_{m-1} z^{m-1} + c_m z^m,$$

and has β_p as a root by construction. This process is automated in RR_Binary_Field_Mod.m.

12.8.1.3 Putting it all together

Given a primitive polynomial $f(z)$ of order m [see §12.8.1.1; often, the minimal primitive polynomial from Table 12.6 is used], denoting α as a root of $f(z)$, and given the minimal polynomials $\phi_p(z)$ with $\beta_p = \alpha^p$ as roots [see §12.8.1.2] for $p = 3, 5, 7, \ldots$, denoting $LCM[\cdot, \cdot, \ldots]$ as the least common multiple of the polynomials indicated, a binary BCH code with $n = 2^m - 1$ and $r = n - k \leq mt$ may then be constructed as follows:

$$\begin{array}{ll}
\text{for } t = 1,\ d \geq 3 \text{ (i.e., binary Hamming)}, & \text{take } v(z) = f(z), \\
\text{for } t = 2,\ d \geq 5 \text{ (for } m \geq 4), & \text{take } v(z) = LCM[f(z), \phi_3(z)], \\
\text{for } t = 3,\ d \geq 7 \text{ (for } m \geq 4), & \text{take } v(z) = LCM[f(z), \phi_3(z), \phi_5(z)], \\
\text{for } t = 4,\ d \geq 9 \text{ (for } m \geq 6), & \text{take } v(z) = LCM[f(z), \phi_3(z), \phi_5(z), \phi_7(z)], \\
\text{for } t = 5,\ d \geq 11 \text{ (for } m \geq 5), & \text{take } v(z) = LCM[f(z), \phi_3(z), \phi_5(z), \phi_7(z), \phi_9(z)], \\
\text{for } t = 6,\ d \geq 13 \text{ (for } m \geq 6), & \text{take } v(z) = LCM[f(z), \phi_3(z), \phi_5(z), \phi_7(z), \phi_9(z), \phi_{11}(z)],
\end{array} \tag{12.27}$$

etc. All of the BCH codes summarized in the introduction to §12.8 were generated in this manner; the process is automated in RR_BCH_Constructor.m.

12.9 Soft decision decoding

The type of decoding discussed thus far, in which the received vector $\hat{\mathbf{w}}$ is assumed to be in $\mathbf{F}_q^n$ [see (12.1)], is known as **hard decision decoding**.

Another formulation of the decoding problem, which we describe here in the binary case $q = 2$, assumes again that $\mathbf{w} \in \mathbf{F}_2^n$ (that is, that the symbols being transmitted are binary, and in this formulation are usually rescaled to be ± 1), but that $\hat{\mathbf{w}} \in \Re^n$ (that is, that the received data is real). In this setting, missing bits in known positions (aka erasures) are assigned a value of zero, and bits received only "weakly" are assigned numeric values closer to zero than to ± 1. The decoding problem in this case, called **soft decision decoding**, is similar to that considered before (again, to find the most likely valid codeword $\mathbf{w}$ corresponding to the received vector $\hat{\mathbf{w}}$, and the original data vector $\mathbf{a}$ that generated it), but is now based on finding the valid codeword $\mathbf{w}$ that minimizes the Euclidian distance to $\hat{\mathbf{w}}$, rather than that which minimizes the Hamming distance.

Implementation of soft decision decoding is somewhat difficult in many applications, as it requires some sort of ADC (analog-to-digital converter) in order to quantify the (real) value associated with the "certainty" of each received bit. This approach is thus not often used in embedded computing today. However, when the data is difficult to obtain but easy to process (e.g., signals received from an interplanetary space probe by NASA's Deep Space Network, or signals received by a wall-powered server from an ultra-low-power embedded IoT device), a soft decision decoding approach may indeed be warranted.

Part III

System Design, Development, and Integration

Chapter 13

Open vs Proprietary Development Models

13.1 Patent protection

Article I, Section 8, Clause 8 of the United States Constitution, often referred to as the Patent and Copyright Clause, grants congress the specific power "to promote the progress of science and useful arts, by securing for limited times to authors and inventors the exclusive right to their respective writings and discoveries."

Patent protection is thus an essential aspect of mechanical and electrical engineering and associated fields in the U.S., specifically in the design of creative new products leveraging ideas developed in such fields. Title 35 of the United States Code (a.k.a. 35 U.S.C.) establishes and governs manners related to the operation of the United States Patent and Trademark Office (USPTO) regarding U.S. patents, including even clear stipulations (in §105) about patentable ideas that are developed in outer space (sorry Elon!). Aspects of U.S. patent law that are not spelled out explicitly in 35 U.S.C. are established via various federal court precedents.

Most other countries with large economies have closely related patent laws, and the Patent Cooperation Treaty (PCT) coordinates patent protections between over 150 member countries. Inventions can be protected in the Europe Union (EU) either by individual national patents, or by European patents granted centrally by the European Patent Office. The discussion below is focused on the relevant US laws, upon which the associated frameworks of patent laws in many other countries, the EU, and eventually Mars (?) are based.

13.1.1 Writing and filing a patent on your invention

As a developer of mechatronic and cyber-physical systems, you will likely be motivated at some point to protect your clever idea(s) with a patent. There are several steps to go through in the patenting process, including:

- evaluating the commercial potential of your idea (is it even worth attempting to patent?),
- determining whether or not the key components of your idea are patentable, including a thorough patent search, as well as a close examination of existing prior art, for related ideas[1],
- understanding the applicable patent law and due process in the countries/planets you want to file your patent,
- (optional) filing a provisional patent application, in order to set an official USPTO date stamp on the original date of your key invention ideas,
- preparing an official patent application, complete with all figures in the required format,
- responding promptly to questions posed by USPTO patent examiners, amending the application as necessary,
- enforcing and maintaining the patent once issued, and

[1] Attempting to patent ideas the essence of which are already covered by existing patents, are previously described in the published literature, or are already available as commercial products is at best a foolish waste of money. At worst, a bogus patent (if granted) is a legal liability, and will make your startup look especially dubious when considered by a larger company as a target for acquisition.

• (optional) marketing and licensing the patent to other companies.

Most people and organizations work with attorneys that specialize in patent law, to make certain that these several important steps are properly followed. However, if on a tight budget, it is entirely possible to *Patent It Yourself*; the several steps highlighted above are covered in detail in, e.g., Pressman & Blau (2020) and Lo & Pressman (2019), to which the reader is referred for step-by-step instructions and extensive helpful, practical guidance[2]. We thus only provide a brief overview of this intricate subject below.

13.1.2 Patent structure

First and foremost, a patent is formed around a set of **claims**, generally written to be as broad as possible in order to cover as many possible variations and implementations of the essential ideas of the patent that one could possibly imagine/foretell, together with a **specification** section that contains a detailed description of one or more **preferred embodiments** that illustrate how the key ideas claimed by the patent may be successfully applied, as you currently envision it.

Some of the claims in a patent are independent (stand on their own), and others are dependent (building on earlier independent claims). There is an important role for each type of claim;

The goal of a set of patent claims is generally to cast as broad a net as possible over the application of the key ideas of the patent. Thus, for example, if the preferred embodiment of the claimed idea builds on a stable frame with 4 legs, instead of stating explicitly "a frame with 4 legs", the phrase "a frame with 3 or more legs" should generally be used in the claims instead. If the preferred embodiment of the claimed idea uses a 100:1 gearbox, instead of stating "a set of four gears providing a 100:1 gear reduction", the phrase "a gearbox with a plurality of gears providing an appropriate gear reduction" should be used, and this provision might further be placed in a dependent claim, leaving the possibility in the main claim for a direct drive arrangement with no gearbox.

That said, the phrasing of claims in a patent should, to the maximum extent possible, use the ordinary and customary meaning (to a POSITA in the relevant field) of the words used, thus leaving no ambiguity of both the structure and function of the various components in the invention when read carefully. In application, the patent stands„ based on the phrasing of the claims alone, and if the claims are written well the specification section of the patent is essentially irrelevant.

13.1.3 Patent litigation

Two key sections of 35 U.S.C., which lay out the requirements for an invention to be patentable, are

• §102, which states that the key ideas of the patent must be "not anticipated" (that is, "novel"), and
• §103, which states that the key ideas of the patent must be "not obvious".

For an idea to be patentable, it needs to be established to the USPTO, based on the state of the related prior art, that both of these characterizations apply to a POSITA (that is, to a "Person of Ordinary Skill In The Art") in the associated field at the effective filing date (that is, the date that the key ideas of the patent application were originally filed by the inventor), not at the time the patent is litigated, which is often several years later (most good ideas, it may be argued, are "obvious", or at least "natural", with the benefit hindsight).

Two U.S. supreme court cases[3], Graham and KSR, set important precedents clarifying the modern standards upon which §103 is now applied. The **Graham framework** establishes three specific things that need to be examined in order to establish nonobviousness:

[2] Even if you plan to file your patent with the help of an attorney, ...

[3] These two U.S. Supreme Court decisions may be cited formally as *Graham v. John Deere Co., 383 U.S. 1 (1966)*, and *KSR Int'l Co. v. Teleflex Inc, 550 U.S. 398 (2007)*.

- the scope and content of the prior art,
- the differences between the claimed invention and the prior art's disclosures, and
- the requisite skill in the art that would have been required to derive the invention at the effective filing date.

The Graham framework also recognizes the pertinence of various "secondary" objective considerations, a **nexus** (that is, a preponderance) of which can be used to establish further prevailing evidence of nonobviousness at the relevant time. Such secondary considerations might include:

- long recognized but previously unsolved needs in the general area,
- failure of other approaches to address the same problem that is solved by the patent,
- commercial success of products leveraging the patent by the inventor,
- licenses granted on the patent to others,
- copycat (possibly infringing) products made by others using the key ideas of the patent, etc.

The **KSR** precedent further establishes that a rational and logical basis needs to be articulated for arguing obviousness via a combination of references in disparate application areas.

Petitions by patent owners may be filed with the USPTO to assert **patent infringement** (that is, that certain competing products incorporate key ideas protected by a patent owner), and petitions may be filed by others to challenge the **validity and scope** of a patent (often based on arguments of lack of novelty and obviousness, based on the state of the prior art on . Such patent litigations are thus based on the various criteria outlined above, and often conclude with substantial financial settlements (payable from the losers to the winners) to compensate for related damages.

In the process of patent litigation, it is the **claims** of the patent that matter most; in the filing of a petition challenging a patent's validity, it is the articulated **grounds** for dismissal that matter most.

In such litigations, **claim construction** (that is, the interpretation of the scope of meaning of each claim, based on careful dissection and analysis of the words used) is the key issue to be resolved.

In the case that words in the patent

if disputed, the preferred embodiments described in the specification are sometimes valuable to resolve various ambiguities.

means for

§112(f)

As a successful developer of mechatronic and cyber-physical systems, the reader of this text might in due course be called upon as an **expert witness** in the litigation of patents in related fields. This opportunity is akin to jury duty. Besides an opportunity for being well renumerated for such efforts, you can learn a lot about patent law, and how it is practically applied, by pursuing such opportunities.

As an expert witness, you will generally be required to write (with the assistance of counsel) and submit a written **declaration** of your opinions, including a brief introduction to your understanding of the relevant patent law (paraphrasing some of the background material provided above might help). At a

Objections to opposing counsel's question that your counsel might file during your deposition as an expert witness include the following:

- **Objection - mischaracterizes testimony**. The question formulated effectively puts words in your mouth.
- **Objection - compound question**. Multiple questions were asked at once.
- **Objection - assumes facts not in evidence**. Implies there may be a false premise in the question.
- **Objection - cause for improper hypothetical**. .
- **Objection - hearsay**. .
- **Objection - cause for speculation**. Causes you to guess at someone else's intentions.

To get accustomed to such objections, so they don't trip you up during your own deposition, you might find it fun to practice these objections around the house with young children (in the author's experience, ample

opportunities often arise to use all of them).

13.2 Open hardware designs

Out innovate the competition

13.3 Community-supported open software design

13.4 Repo maintenance

LibreOffice

13.4.1 Case study: RR library

References

Pressman, D, & Blau, D (2022) *Patent It Yourself: Your Step-by-Step Guide to Filing at the U.S. Patent Office.* 21st Edition. NOLO.
Lo, J, & Pressman, D (2019) *How to Make Patent Drawings.* Eighth Edition. NOLO.

Chapter 14

Crowd Funding vs Venture Capital

Chapter 15

Computer Aided Design & Manufacturing (CAD/CAM)

15.1 Mechanical CAD (MCAD)

15.1.1 Case study: Onshape

15.2 Electrical CAD (ECAD)

15.2.1 Case study: Altium

15.3 Hybrid CAD programs

As of 2021, attempts to fuse mechanical and electrical CAD programs are still in their infancy...

15.4 Additive vs subtractive manufacturing

15.4.1 3D printers

15.4.2 Mills and lathes

15.4.3 Injection molding

Chapter 16

Design Paradigms

16.1 Wheeled designs

elder care, concierge, toys, …

16.1.1 Case study: eduRover

16.1.2 Case study: SCR (Stair Climbing Robot)

16.2 Reaction-wheel and CMG-based designs

16.2.1 Case study: iceCube

16.3 Legged walking, running, and hopping

16.3.1 Case study: iHop

16.4 Drones

16.4.1 Case study: eduMAV

16.5 Tensegrity structures

16.5.1 Case study: control of a 3D stage

16.5.2 Case study: control of a multiply tethered balloon

16.6 Origami and kiragami

16.7 Biomimetic and bioinspired designs

16.8 Soft robotics

fruit picking, surgery

16.9 Industrial robotics

16.9.1 Assembly lines

Paolo's conveyer belt.

16.9.2 Case study: Creator burger machine in SF

16.9.3 Robot arms

16.9.4 Case study: Flippy at LA Caliburger

16.10 Pick and place machines

Delta mechanism

Chapter 17

Case study: myMiP

Appendix A

Matlab programming

A useful definition of a **small-scale numerical problem** is a calculation that takes longer to code and debug than it does to run. By this definition, many calculations that you will perform in science, engineering, and other disciplines, both as a student and in industry, are indeed small-scale numerical problems.

Matlab (a **portmanteau word**[1] formed from **mat**rix **lab**oratory) is a powerful high-level programming language marketed by MathWorks. Though expensive[2], Matlab has become something of a de facto industry standard for many classes of small-scale numerical problems, in areas such as linear algebra, data analysis & visualization, control design, system identification, and optimization.

GNU Octave is a powerful, free, community-developed alternative to Matlab that is almost entirely compatible with the (well-established and documented) Matlab syntax. You are assured free access to a legal copy of Octave for any computer platform that you might use in the future, so it is a good idea to test all of the major codes that you develop in Matlab syntax in both Matlab and Octave, as we have attempted to do in this text[3], so that you can be assured that you will be able to run them without difficulty in the future.

Both Matlab & Octave[4] provide an interactive, user-friendly environment in which the plotting of simulation results is especially simple. These programming environments are thus quite useful as intuitive testing grounds in which one can experiment with small-scale numerical problems on a laptop or desktop computer. There are two main directions one can go from there:

(1) embedding numerical algorithms efficiently into inexpensive low-power microcontrollers (see §1.5) for controlling robotic systems, a class of problems that we focus on in particular throughout this text, *RR*, and
(2) designing numerical algorithms that efficiently scale to much larger numerical problems which tax the capabilities of the largest and most modern computational platforms that you can afford to use, a class of problems that we focus on in the companion text, *NR*.

In both cases, low-level compiler-based languages [such as C, Fortran, and many others; see §2] are strongly preferred, as they give the programmer much more precise control over both the memory usage and the parallelization of the numerical algorithm. Conversion of numerical algorithms from Matlab syntax to the syntax of such lower-level languages is generally straightfoward, as discussed further in, e.g., §11.4 of *NR*.

[1] A portmanteau word is formed out of parts of other words, which is common in the naming of computer hardware and software. For example, **Fortran** is a portmanteau word formed from **for**mula **tran**slation, **codec** from **co**der/**dec**oder, **voxel** from **vo**lumetric pi**xel**, etc. Such words are often formed informally as new technology is developed, then become established through usage.

[2] Note that special Matlab For Students deals are available on many college and university campuses.

[3] Please contact the author if you encounter errors running any of the algorithms presented in this text, and the associated codebase, in recent versions of either Matlab or Octave.

[4] A few popular alternatives to Matlab & Octave well suited for both small-scale numerical problems (working with floating-point numbers) as well as symbolic computations (that is, software-based manipulation of mathematical expressions for solving algebra and calculus problems) include Python/NumPy/SciPy/SymPy, R, Julia, Scilab, Maxima, Mathematica/Alpha, and CPL.

A.1 Fundamentals of both Matlab and Octave

Once you get Matlab or Octave up and running, you will likely find that no manual or formal classroom instruction on either language itself is even necessary. Most of the basic constructs available in such languages [primarily, basic arithmetic on floating-point numbers, **for** loops, **if** statements, and **function** calls] can generally be understood easily simply by examining sample codes, such as those developed throughout *RR* and *NR*.

It is helpful to recognize that complex problems are solved efficiently on modern computers simply by sequencing appropriately basic arithmetic, for loops, if statements, function calls, and data storage and retrieval together, using logic which is admittedly sometimes subtle. It is generally the logic itself, and the judgement and reasoning involved in organizing it, that makes The Art of Computer Programming a skill that takes years to master; the syntax of the language best suited for the job (Matlab, C, or something else) is generally something that is quite easy to pick up, or convert to, by examining a handful of well-written example codes.

Convenient built-in command names in Matlab/Octave are all intuitive (**sin** for computing the sine, **eig** for computing eigenvalues/eigenvectors, etc.), and extensive **help** for all commands is readily available in both Matlab and Octave, at the >> prompt in the command window, simply by typing, for example,

```
>> help eig
```

Even more information is available online. These help pages also point you to several related commands, which can be used to learn what you need to know about any given aspect of Matlab or Octave very quickly.

To help get you off to a fast start, we now introduce some of the fundamental constructs used in Matlab and Octave, then explain some of their more subtle features. To begin, Matlab or Octave can function as an ordinary calculator. At the command prompt, try typing[5]

```
>> 1+1
```

Matlab or Octave should reassure you that the universe is still in good order. Note that you can always scroll back to see the preliminary definitions and calculations that led to a particular result using the up arrow on your keyboard. To enter a matrix, type

```
>> A=[1 2 3; 4 5 6; 7 8 0]
```

Note that matrix elements are separated by commas or (where it can be done without ambiguity) spaces, and a semicolon indicates the end of each row of the matrix. Matlab/Octave responds with

```
A =
     1     2     3
     4     5     6
     7     8     0
```

By default, Matlab/Octave operates in a **verbose** mode[6] in which the results generated by any given command will be printed on the screen as soon as they are calculated. Once a code segment is debugged, such a verbose behavior quickly becomes tedious, and slows the computer down. To suppress this behavior, simply type a semicolon after any command that would otherwise dump output to the screen; the use of semicolons after calculations or function calls also allows several commands to be included on a single line, such as

```
>> A=[1 2 3; 4 5 6; 7 8 0]; x=5;
```

To put multiple commands on a single line without suppressing echo mode, separate the commands by commas (try it!). Three periods in a row means that the present command is continued on the following line, as in:

```
>> A=[1 2 3;  ...
       4 5 6;  ...
       7 8 0];
```

[5] To get maximum value from this appendix, we recommend copying/pasting (or, retyping) the commands following the >> prompts in the text into your own Matlab/Octave window, modifying it a bit, and checking that the output generated makes sense.

[6] This verbose behavior can be further augmented by toggling on the **echo** command, which displays the actual statements encountered during execution of a script (see §A.2.1). This command, which apparently evolved from the TRON command of the 1980s vintage BASIC programming language, prints so much information to the screen that its practical utility is actually somewhat limited.

The legibility of your code can be substantially improved by aligning long expressions in a natural fashion using spaces or tabs, as illustrated above. Elements of a matrix can also be arithmetic expressions, such as `3*pi`, etc.; when doing this, it often necessary to separate matrix elements by commas to remove any possible ambiguity.

Matlab syntax has control flow statements, such as **for** loops, similar to other programming languages. Note that each **for** must be matched by an **end**. To illustrate, the commands

```
>> for j=1:10, a(j)=j^2; end, a, b=[0:2:10]
```

build row vectors (try it!), whereas the commands

```
>> for j=1:10, c(j,1)=j^2; end, c, d=[0:2:10]'
```

build column vectors. In most cases, you want the latter, not the former. *The most common mistake made in Matlab syntax is to build a row vector when you intend to build a column vector*, as their use in Matlab/Octave is usually not interchangeable; thus, pay especially close attention to this issue if your code is misbehaving.

The format of a **while** statement is similar to that of **for**, but exits at the control of a logical condition:

```
>> m=0; while m<7, m=m+2; end, m
```

An **if** statement may be used as follows[7]:

```
>> n=7; if n>0, sgn=1, elseif n<0, sgn=-1, elseif n==0, sgn=0, else disp('undefined'), end
```

The (related) **switch**/**case** construction allow one to check a single variable against several possible conditions

```
>> switch n, case -1, c='N', case 0, c='Z', case 1, c='P', otherwise, c='?', end
```

A column vector `y` can be premultiplied by a matrix `A` and the result stored in a column vector `z` with, e.g.,

```
>> A=[1 2 3; 4 5 6; 7 8 0], y=[1 2 3]', z=A*y
```

Subsequent multiplication of the vector `z` by a scalar, like the predefined constant **pi**, may be accomplished with

```
>> w=pi*z
```

A 3×3 matrix A with complex random entries, each with real and imaginary parts uniformly distributed between 0 and 1[8], its conjugate $B = \overline{A}$, its transpose $C = A^T$, and its conjugate transpose $D = A^H = \overline{A^T}$ (see §1 of *NR*), may be generated as follows

```
>> A=rand(3,3)+sqrt(-1)*rand(3,3), B=conj(A), C=A.', D=A'
```

The inverse of a square matrix (that is, the matrix B such that $BA = I$, if it exists) may be obtained by typing

```
>> B=inv(A), check=B*A
```

For pedagogical purposes, this inverse command is rewritten in §2 of *NR*. As mentioned there, you should never actually compute a matrix inverse[9] in a **production code** (that is, in a code designed to run at the maximum possible speed, without failure), though it is sometimes convenient to compute a matrix inverse in a **test code** (that is, in a code used for demonstration purposes only, on small-scale numerical problems).

A 5×5 identity matrix may be constructed with

```
>> E=eye(5)
```

Tridiagonal matrices (see §1.2.7 and §2.2.5 of *NR*) may be constructed by, e.g., the following command:

```
>> m=5, x=randn(m-1,1), T=1*diag(ones(m-1,1),-1)-2*diag(ones(m,1),0)+diag(x,1)
```

[7] When in the Matlab command window, using the up and down arrows allows you to scroll back and forth through recently executed commands. For example, after copy/pasting/running the commands shown here, scroll back to it using the up arrow, walk to the left of the line with the left arrow, change `n=7` to `n=NaN` or **Inf**, hit enter, and see if the new answer makes sense (see §1.1.4).

[8] Note that the command `A=randn(2,3)` generates a 2×3 matrix A with real, random, independent, normally-distributed entries, each sampled from a Gaussian probability distribution with zero mean and standard deviation 1 (see §6 of *NR*), and the command `A=randi(13,2,3)` generates a 2×3 matrix with positive integer entries uniformly distributed between 1 and 13.

[9] The algorithm to compute a matrix inverse is computationally very expensive, as discussed in §2 of *NR*, and destroys any known sparsity structure (that is, sets of elements known to be zero) in the original matrix, as discussed in §1 of *NR*.

There are two "matrix division" commands in Matlab/Octave, **mldivide**, also written as \, and **mrdivide**, also written as /; if A is a nonsingular square matrix, then A\B and B/A correspond formally to left and right multiplication of B (which must be of the appropriate size that the product is well defined) by the inverse of A [i.e., **inv**(A)*B and B***inv**(A), respectively]. However, the commands A\B and B/A obtain these answers directly via **Gaussian elimination with pivoting**, as developed from scratch (again, for pedagogical reasons) in §2 of *NR*, while leaving the matrix A intact, *without* computing **inv**(A) along the way (which, as mentioned in Footnote 9 above, is prohibitively expensive for large matrices). Thus, to solve a system A*x=b for the unknown vector x, one may simply type, for example,

```
>> A=[1 2 3; 4 5 6; 7 8 0]; b=[5 8 -7]'; x=A\b
```

which results in

```
x =
    -1
     0
     2
```

To check this result, just type

```
>> A*x
```

which verifies, as expected, that

```
ans =
     5
     8
    -7
```

Starting with the innermost group(s) of operations nested in parentheses and working outward, the usual precedence rules are observed by Matlab/Octave. First, all the exponentials are calculated. Then, all the multiplications and divisions are calculated. Finally, all the additions and subtractions are calculated. In each of these three catagories, the calculation proceeds from left to right through the expression. Thus

```
>> a=5/5*3, b=5/(5*3)
```

gives a=3 and b=0.3333. If in doubt, use parentheses to ensure the order of operations is as you intend.

Matrix sizes must be such that the requested linear algebra operation is well defined (see §1 of *NR*), or an error will be thrown. For example, suppose we have two column vectors x and y and wish to perform the component-wise product of each element of x with the corresponding element of y. Such a component-wise multiplication may be accomplished in Matlab syntax as, for example,

```
>> x=[1:5]';  y=[6:10]';  z=x.*y
```

Note that z=x*y throws an error, since this implies a linear algebra operation that is undefined (that is, a column vector times a column vector). In contrast, a row vector times a column vector is well defined, so z=x'*y is successful, generating the inner product of x and y (try it!).

The period generally distinguishes matrix operations from component-wise operations, for example (try it!)

```
>> A=[1 2; 3 4], B=[5 6; 7 8]
>> C1=A^2,  D1=A*B,  E1=A/B   % Matrix operations!
>> C2=A.^2, D2=A.*B, E2=A./B  % Component-wise operations!
```

Typing **whos** lists all variables you have created up to that point, and typing **clear** removes these variables from your workspace. Typing **clc** clears the command window.

The **format** command toggles the number of significant figures printed to the screen, for example,

```
>> format long, pi, format short, pi
```

Various self-explanatory Matlab/Octave functions include: **factorial**, **abs**, **conj**, **real**, **imag**, **sin**, **cos**, **tan**, **asin**, **acos**, **atan**, **atan2**, **sinh**, **cosh**, **tanh**, **asinh**, **acosh**, **atanh**, **exp**, **log**, **log10**; some predefined constants[10] include **pi**, i, j,

[10] Note that Inf is actually signed in Matlab (see §1.1.4); try typing, e.g., 10^310 and -10^310.

eps, **Inf**, **NaN**. Your code can actually change such constants[11] (it is particularly common to use i and j as indexing variables); be careful if you do this, and later need to use these constants as originally defined!

Matlab and Octave are distributed with many special additional functions to aid in linear problem-solving, control design, etc. Many of these advanced built-in functions are themselves just prewritten m-files (see below) that can be opened and accessed by the user for examination with, for example, a command such as **open bode**. In most cases, *RR* and *NR* avoid most of these convenient **black-box** functions, instead working up the core of many of them from scratch, to remove the mystery that might otherwise be associated with their use.

Sometimes, Matlab or Octave will suspend the printing of text to the command window, or the drawing of a plot to a figure window, until later computations are finished or a **pause** statement is reached. The **fflush**(stdout) command in Octave, and the **drawnow**('update') command in Matlab, can be used to force this output to be printed or drawn. This is one of the few little (yet, annoying) differences between Matlab and Octave.

All of Matlab's "random" number functions, including the **rand**, **randi**, and **randn** commands mentioned above, draw values from a shared pseudorandom number generator (PRNG; see §2.7). Matlab actually has several deterministic algorithms implemented for pseudorandom number generation with good statistical properties implemented; it uses the Mersenne Twister by default. Matlab's PRNG algorithm has a "seed" that is reset every time Matlab is restarted (thereby generating the exact same sequence of pseudorandom numbers every time Matlab is restarted). This seed can be manually reset to the default initial state with the **rng**('default') command, or can be manually set to a "random" initial state, based on the current time, using the **rng**('shuffle') command.

A.2 Matlab programming procedures: stay organized!

As an alternative to interactive mode, you can also save a series of Matlab/Octave commands in m-files, which are just ASCII (aka plain text) files with descriptive filenames, ending in .m, containing a sequence of commands listed exactly as you would enter them if running interactively. When working on Matlab/Octave problems that take more than one line to express (that is, essentially, all the time, even when using Matlab/Octave as a simple calculator!), *it is imperative to write and run m-files rather than working in interactive mode.* By so doing, it is self evident which calculation follows from which. Further, following this approach, the several commands typically required to perform a given calculation do not need to be retyped when the calculation or simulation needs to be modified and rerun, which is generally much more often than one would care to admit. *Staying organized with different versions of your m-files as a project evolves (even a fairly simple project!) is essential. Create new directories and subdirectories as appropriate for each problem you work on to stay organized, and to keep from accidentally overwriting previously written and debugged codes.*

To execute the commands in a script named[12] **foo**.m, type **foo** at the >> prompt. Any text editor (see §2.4.1.2) may be used to edit m-files. A % symbol in such a file indicates that the rest of that line is a comment. Typing **help foo** prints the first set of commented lines of **foo**.m to the screen; **type foo** prints the entire **foo**.m code to the screen. *Comment all m-files clearly, sufficiently, and succinctly*, focusing specifically on its inputs and outputs, so that you can come back to the code later and understand how it works. You can also print nicely to the screen (using, e.g., the **disp** or **fprintf** commands) to update the user on the code's progress as it runs; several codes discussed in this text use a RR_VERBOSE flag to turn such updates on or off. It also helps to use descriptive variable names within any code. There is an important tradeoff between succinctness and readability; this tradeoff should be made deliberately, don't go overboard on one side or the other.

[11] If you're bored, try redefining **pi**=3.2 and see how much it messes things up; be glad this wasn't established by legislative fiat!

[12] Almost all texts describing computer programming, dating back to the 1978 classic by Kernighan and Ritchie, make reference to expository codes named **foo** and **bar**. Following this convention is a small way one can pay respect to those greats in the field of computing who came before us.

A.2.1 The distinction between scripts and functions

There are two distinct types of m-files: **scripts** and **functions**.

A **script** is a set of Matlab/Octave commands that run just as if you had typed in each command interactively. A script has access to all previously-defined variables (that is, if called from the interactive window, it inherits the **base workspace**), and all variables that it defines are available for later inspection in that workspace, which is sometimes useful when debugging. In order to make a test script run the same way every time (repeatability is usually strongly desired in numerical calculations!), it is generally a good idea to put a **clear** command at the beginning of all of your scripts, so they always run "from scratch"[13].

A **function**, on the other hand, is a set of commands that begins with a **function** declaration that defines that function's inputs and outputs; for example,

```
function [output1,output2] = bar(input1,input2,input3)
```

A function so defined may then be **called** (as in a compiler-based programming language) with the command

```
[c,d] = bar(a,b,c)
```

Note that some variables (in the above example, c) may be used as both inputs and outputs.

When a function is running, it can only reference those external variables that are transferred in via the input list with which it was called; in the present example, the function is only "aware" of the external variables {a,b,c}, which this function refers to internally as {input1,input2,input3}. A notable exception to this rule is those variables that are declared as **global** in multiple functions and (usually) the base workspace; in this case, a single copy of the variable so declared is shared, but only to those functions that include its global declaration. Global variables are usually denoted with both all capital letters and especially descriptive variable names (e.g., **global** FUNCTION_EVALUATION_COUNTER), to keep these special variables from being accidentally overwritten in the several different places that they might be used. Global variables should be used only sparingly.

The special variables **nargin** and **nargout** identify the number of input and output arguments, respectively, that are actually used when any given function is called. Input and output arguments are assigned from the left to the right, so any missing arguments in this call are necessarily those at the right end of each list. If some of the input arguments, often tagged as "optional", are omitted in the function call, logic may be implemented in the function to set these variables to certain default values; if this logic is not implemented properly, the function will crash when these variables (if left undefined by the function call) are first referenced. If some of the output arguments are omitted in the function call, logic may be implemented in the function to avoid explicit computation of these omitted output variables in order to reduce execution time.

After a function finishes running, the only variables that are modified in the workspace that called the function, as compared to before the function was run, are those (**nargout**) variables in the function's output list that are paired with corresponding variables in the command that called the function, in addition (possibly) to some of the variables declared as global. In the above example, if called as c2=**bar**(a,b,c1), the function **bar** only modifies[14] the single (that is, **nargout**=1) variable c2, which **bar** refers to internally as {output1}.

Functions are much more easily embedded as smaller parts of larger programs than scripts, as functions make crystal clear, via their input/output argument list, what information used in, and returned by, the called function. In complicated codes, unintentionally assigning a minor variables (like the index i or k) with different meanings in different scripts that call each other can lead to a bug that is nearly impossible to find. The proper use of functions, and the associated passing of only the relevant data back and forth (known as **handshaking**), goes a long way towards preventing such insidious bugs from appearing in your production codes.

On the other hand, short test codes, such as those provided with many of the functions developed in *RR* and *NR*, are often convenient to write as scripts, so that the variables defined by the test script may be checked (for debugging purposes) after the test script is run.

[13] See also the discussion at the end of in §A.1 about resetting the PRNG seed, which is also sometimes useful in test scripts.

[14] That is, in addition to those variables defined as global, as mentioned above.

A.2.2 Pass by value versus pass by reference†

There is extra overhead associated with function calls, as they usually **allocate** (that is, assign) new memory locations to receive the variables passed in when called, then **deallocate** (release) this memory upon exit. This general approach is referred to as **pass by value**. In addition to the allocation and deallocation of memory for each function call, the pass by value approach must also copy the values from the input array(s) into the new memory location(s) allocated by the function call, then pass the resulting values from the output array(s) back into the appropriate array(s) in the global workspace. For scalars and short vectors, this overhead is minimal; it is actually beneficial in certain settings, especially when using a compiler, as it identifies which variables are about to be heavily referenced next, and should thus be stored in a high-speed memory cache (see §1.3).

For large arrays, however the overhead associated with the pass by value approach can be unacceptable. There are generally two approaches to avoid this overhead. The first approach is simply to use arrays declared with the **global** identifier (see previous page). Though computer scientists generally frown upon this approach, as it can lead to confusing functions that do not clearly identify their inputs and outputs up front, and can thus be somewhat difficult to reuse in a black box fashion on other problems, engineers who write very-large scale, special-purpose numerical codes often find this approach to be the fastest[15].

The second approach is **pass by reference**. This is often the best compromise between the overhead associated with the pass by value approach, and the raw speed of the approach leveraging globally-defined arrays. The pass by reference approach involves, during the function call, simply the passing of a **pointer** to the starting-point location of an array in memory [in addition, somehow, to the dimensions of the array in question], rather than the cumbersome allocation/copying/deallocation process associated with the pass by value approach. The pass by reference approach is the default for array passing in Fortran, and is also quite common in C.

Matlab's default is, usually, a pass-by-value approach, with (quite cleverly) the memory allocation and copying of data associated with the pass-by-value approach actually deferred until the first time a given array is modified within the function itself (and, thus, a new memory location for that array is actually warranted).

However, a pass-by-reference mode is initiated in Matlab (automatically, without any intervention by the user, which is also rather clever) whenever a code calls a function with an identical argument in both its input and output lists (in both the code that makes the call, as well as the function that is called). This subtle feature should not be ignored, as it can lead to **significant performance improvements** in the execution speed of Matlab codes applied to large problems. As an example, if a Matlab function defined as

```
function [array1,array2,array4] = foo(array1,array2,array3)
```

is called with the command

```
[a,e,c] = foo(a,b,c)
```

then the a array (referred to inside **foo** as array1) is handled with a pass-by-reference approach, but all other arrays are handled with a pass by value approach (as the corresponding array names don't match in both the input and output lists, in either the code that makes the call, or the function that is called). Passing very large arrays back and forth to functions in the manner that a is passed into array1, and then back out to a, in this example can substantially improve the execution speed of your code.

It is sometimes convenient to group related parameters together into a structure array, aka **struct** [such as p.linestyle1 ='k-', p.linestyle2 ='b-.', p.omega_min=0.01, p.omega_max=0.10, p.K=20, ...], then simply passing the name of the variable assigned to this structure array [in this example, p], which conveniently contains all of the (appropriately grouped) optional parameters, thus avoiding the use of global variables, while also avoiding the listing of each and every minor parameter in the list of input arguments of your functions.

[15] Also, for functions that you plan to call a lot (like, every timestep in a long simulation), declaring "scratch" arrays that are only used internally in certain functions with the **persistent** identifier prevents those arrays from being allocated and deallocated every time that function is called, which can substantially accelerate code execution. As opposed to using the **global** identifier, variables tagged with the **persistent** identifier can not be accidentally overwrite in other functions.

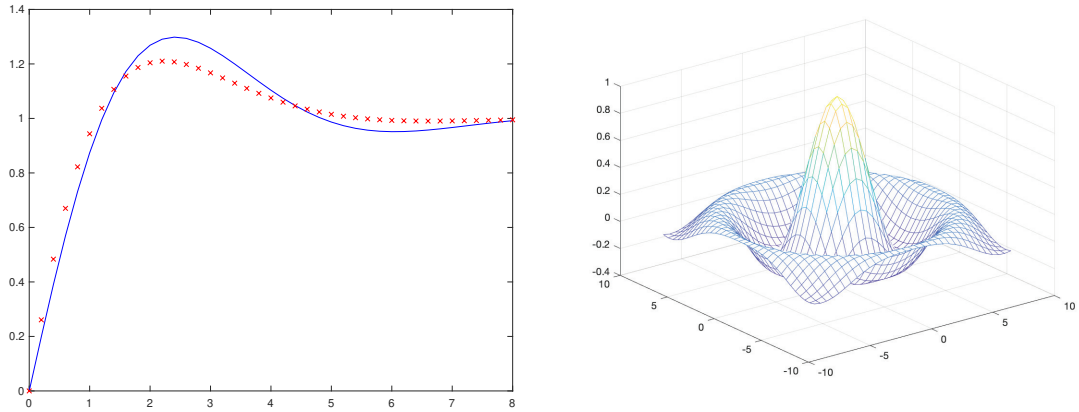


Figure A.1: Sample 2D and 3D plots: (left) the step responses 2nd-order continuous-time (CT) and discrete-time (DT), as developed in Examples 8.1 and 8.2, and (right) $\mathrm{sinc}(r) = \sin(r)/r$ where $r = \sqrt{x^2 + y^2}$.

A.3 Plotting

Both 2D and 3D plots are easy to generate in Matlab and Octave, as shown below:

A sample 2D plot

```
>> t=[0:.2:8]; z=[.5; .7];
>> s=z; o=sqrt(1-z.^2); d=z./o;
>> y=exp(-s*t).*(-cos(o*t)+d.*sin(o*t))+1;
>> plot(t,y(1,:),'b-',t,y(2,:),'rx')
```

A sample 3D plot

```
>> [x,y]=meshgrid(-8:.5:8,-8:.5:8);
>> R=sqrt(x.^2 + y.^2)+eps;
>> Z=sin(R)./R;
>> mesh(x,y,Z)
```

The code segments listed above produce the figures shown in Figure A.1. Linear versus log axes, titles, axis labels, etc, can be controlled with **loglog**, **semilogx**, **semilogy**, **title**, **xlabel**, **ylabel**, and related commands (see the corresponding help pages); **axis**([0 5 -1.1 1.1]) zooms a 2D plot (like that above left) to the region $[0, 5]$ on the horizontal axis and $[-1.1, 1.1]$ on the vertical axis, **axis** equal sets the aspect ratio so that equal tick mark increments on the various axes are equal in size, and **axis** square makes the current axis box square. Try it!

Commands like those above produce plots in figure windows. Once a plot is as you like it, you will often want to save it, so you can email it, include it in a talk, post it on social media (you are, after all, an engineer!), make printouts of it, and/or include it in a paper or textbook you are writing. For scientific writing, LaTeX invariably produces the best results, though you may find that **what-you-see-is-what-you-get** (wysiwyg) word processors like Microsoft Word or Apple Pages are, at least initially, somewhat easier to use[16]. The recommended format to save your figures for such purposes is encapsulated postscript (denoted with a **.eps** suffix). Encapsulated postscript is a robust, platform-independent standard for both vector graphics files (representing lines as lines) and bitmaps (collections of pixels); vector graphics look especially sharp when printed out or zoomed in. All major typesetting programs, including LaTeX, Word, and Pages, can import **.eps** bfiles.

To produce a color **.eps** file, execute the **print** command after setting up the plot as you like it[17]:

```
print -vector -depsc sinc.eps
```

Once you have created an **.eps** file, you may view it with any (free) eps viewer, such as ghostscript and gv. Adobe Illustrator is a good commercial software package for making edits to **.eps** files (changing line types, font sizes, etc.), which is often necessary when preparing scientific documents. The psfrag package is especially powerful for replacing tags (characters) in **.eps** files with mathematical expressions generated by LaTeX, thus seamlessly integratings complex figures and equations into your documents (see, e.g., Figure 10.7).

[16] That is, until you begin to care about how well the equations are typeset, at which point your best option is to switch to LaTeX.

[17] Above a certain number of elements in a given figure, Matlab automatically switches from the (highest-quality) **–vector** renderer to the (bitmapped) **–image** renderer. Manually forcing the former, as done here, ensures the (higher-quality) vector output. Note that, prior to fall 2021, the **–vector** and **–image** options to the **print** command were called **–painters** and **–opengl**, respectively.

In Matlab, the contents of a figure window may also be saved with the command **saveas**(1,'foo.fig'), and later reopened and edited further with **open foo**.fig. The author does not typically recommend this approach, however, as it substantially limits further modification. Instead, to send a figure via email or to include it in a paper, use the **.eps** format discussed previously, and manually downgrade it to a **.jpeg**, if necessary, to reduce file size. If you want the option (you usually will!) to edit the figure later in Matlab or Octave, your flexibility is maximized by saving, in an m-file, the entire list of commands that generated the figure, thus allowing you to tweak these commands in the future, and regenerate the figure of interest from scratch.

Printouts of the text appearing in the Matlab or Octave command window after a code is run is best achieved simply by copy/pasting this text into the editor of your choosing, then printing (or generating a pdf) from there.

A.4 Source code repositories: Github and its alternatives

Source code repositories (aka repos) are like Google Docs, but for numerical codes. They:

- provide a backup of all of your most important coding work,
- sync codebases between different computers that you might use during the week,
- provide version control, allowing you to revert to a previous version of a code if a new edit breaks things,
- allow developers to fork a codebase from its mainline into a private branch for code development, and
- allow repo owners to merge new code, once debugged, from private branches back to the mainline,
- thus facilitating the simultaneous collaboration of many developers on a large set of interacting codes, while
- enabling repo owners to distribute (and, keep updated) the mainline of a big codebase to a large set of users.

Note that forking is easy, and happens at the push of a button; merging (that is, reconciling possibly conflicting code updates in different branches) is where substantial care is sometimes required. As of this writing, Github is the dominant standard for source code repositories; alternatives to Github include:

GitLab, Bitbucket, GitBucket, Sourceforge, AWS CodeCommit, and Google Cloud Source Repositories.

The many codes developed in both *Renaissance Robotics* and *Numerical Renaissance* are maintained at the Renaissance Repository (https://github.com/tbewley/RR).

You may not realize it now, but as an engineering student interested in robotics and numerical methods, you will both use large codebases and, ultimately, write lots of codes yourself (in Matlab, C, Python, Fortran, and many other languages). To become successful in this endeavor, you should thus become familiar with the proper use of source code repositories early on. Further, one of the most valuable things you can include in your resume, when looking for academic or industry jobs in the fields of robotics and numerical methods, is a link to your Github page. Even if it just has codes from various classes and small projects that you have worked on thus far, a well organized Github page demonstrates clearly to a potential employer your coding style and clarity of thought, and well showcases the major engineering skills that you have developed thus far.

After opening an account at Github[18], it is also convenient to download Github Desktop[18] onto the (Mac or Windows) computers that you plan to use to write code, which provides a convenient graphical interface to:

- clone a repo (like the Renaissance Repository) that you want to use,
- update your local clone of a repo with recent improvements from the mainline version of the repository,
- sync/merge your own daily local code developments back into your own repositories online,
- fork someone else's repo into a private branch that you can work on yourself, as a developer, and, ultimately,
- submit a pull request to suggest to a repo owner that they merge your new codes into their mainline.

To get started, (a) open a Github account, (b) download Github Desktop, (c) use Github Desktop to clone the Renaissance Repository to your computer, and (d) create and use your own repos for your current classes and

[18] Complete instructions for using Github and Github Desktop are available at the corresponding websites. Choose your Gihub username carefully, because you will most likely want to use this account for the rest of your life!

projects.

A.5 Navigating your path

Akin to both unix/Linux/Mac shells (see Table 2.1) and the Windows command prompt, the **cd** command changes directory (that is, the current Matlab working directory, for saving new files), the command **dir** lists the files in that working directory, and the command **pwd** prints the working directory to the screen. If the necessary m-files to run your code are stored in more than the current working directory, which is generally the case if you are staying well organized (please do!), the command **path** can be used to view the set of directories that Matlab will look within to find the additional m-files that it may need, and the command **addpath**(dir1,dir2), where dir1 and dir2 are strings containing complete path names (e.g., dir1='/Users/bewley/classes/MAE144', etc.), may be used to add directories to this path.

In particular, once you have cloned (using Github Desktop, as discussed in the previous section) the Renaissance Repository to your computer, you will want to add all of the directories containing your local copy of the codes it contains to your path. This is best done automatically, when firing up Matlab. A convenient script, RR_path_init.m, is provided to help with this.

To use this path initialization code, you should set up your computer to call it automatically when firing up Matlab. This may be done by appending a call to RR_path_init in the **startup**.m file of your default Matlab **userpath** directory.

In short, fire up Matlab, and type the command **userpath**, which will return the name of the default directory that Matlab starts up in on your machine. Within that directory (important!), edit the file **startup**.m (or, create a new file of this name if one doesn't already exist); this file should contain, at least, the following lines:

```
RRbase='/Users/bewley/RR'; cd(RRbase); RR_path_init
```

Note: replace the directory name in single quotes above with the full path to the location that you have installed the Renaissance Repository on your computer. Note that forward slashes, /, as shown above, are used on Macs, whereas backslashes, \, are used in Windows; on a Windows machine, the full path to this directory might look something like C:\Users\bewley\RR. Note you can also put other commonly needed Matlab initialization commands in your **startup**.m file; in particular, you will probably want to add the paths to your personal Matlab project directories that you use often (see the last sentence of the first paragraph of this section), and perhaps also (at the end of the **startup**.m file) a **cd** to the directory that you plan to be working in most in the foreseeable future. You may also include a host of other actions, such as calling a routine like RR_physical_constants, to define some of the physical constants that you might often need in your line of work.

Many other getting-started functions are available in the Renaissance Robotics Appendix A section of the repository. Rather than listing those simple functions here, the reader is asked to, well, **cd**(strcat(RRbase,'/chapAA')) in your Matlab command window at the >> prompt, then **dir**, then **type** RR_double_factorial, **type** RR_swap, **type** RR_permute, etc, for each command that you find. By studying these sample codes, and running the test commands embedded in their comments, I reckon you'll digest them quite quickly, and find your own way forward from there.

A.6 Advanced prepackaged numerical routines

Matlab and Octave have a ton of very useful advanced prepackaged numerical routines built in, including **inv**, **lu**, **qr**, **cond**, **eig**, **schur**, **svd**, **norm**, **rank**, **pinv**, **tf**, **minreal**, **bode**, **rlocus**, **impulse**, **step**, **c2d**, **lyap**, **dlyap**, **icare**, **idare**, etc. As you progress through the *RR* and *NR* texts, these routines will quickly become useful for you, as you learn *what* they compute. However, the purpose of the *RR* and *NR* texts is actually not simply to catalog these prepackaged routines, but rather to flush out *when* and *where* you might use them, *why* they are the tools of choice for certain problems, and *how* they actually work. With this knowledge, the reader will be able to select and use such routines with much greater understanding and forethought. We thus avoid using almost all such prepackaged routines in these texts, opting instead to rewrite many of them from scratch.

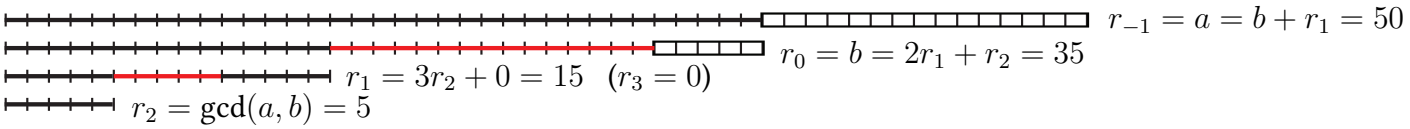


Figure A.2: Euclid's algorithm: Take $r_{-1} = a$ and $r_0 = b < a$. For $k \geq 1$, subtract the largest integer multiple of r_{k-1} from r_{k-2} that leaves a remainder $r_k \geq 0$, until some $k = n$ for which $r_n = 0$; then, $\gcd(a, b) = r_{n-1}$.

A.7 Keeping it classy with object-oriented programming

The modern notion of object-oriented programming (**OOP**), which defines **classes** of objects, and valid **operations** on objects that are members of these classes, can be valuable[19] for the development of **reusable code**. The RR codebase illustrates the practical use of OOP by example, by defining and using three object classes:

- integers (see RR_int.m), with Euclidean division giving an integer quotient and remainder,
- polynomials (see RR_poly.m), with Euclidean division giving a polynomial quotient and remainder, and
- transfer functions (see RR_tf.m), represented by rational functions of s (see §8.2.3) or z (see §8.3.3).

We focus specifically, in §A.7.1, on the use of the RR_int and RR_poly classes on the Diophantine problem, which highlights well the versatility of the OOP approach. The codes defining these classes, as well as the fundamental operations ($+, -, \times, \div, \ldots$) on objects in these classes, should at this point be mostly self explanatory; some special syntax used in these definitions is described in detail at the related Matlab help pages.

A.7.1 Euclid's algorithm and the Diophantine equation

The **greatest common divisor** (**GCD**, aka greatest common factor, GCF) of two[20] positive integers a and b, denoted $\gcd(a, b)$, is the largest positive integer g such that a/g and b/g are both integers. The GCD may be computed using the **Euclidean algorithm** (aka **Euclid's algorithm**): define $r_{-1} = a$ and $r_0 = b < a$, and perform **integer division** of r_{k-2}/r_{k-1} to determine the quotient and remainder $\{q_k, r_k\}$ [i.e., find the largest positive integer q_k and associated positive integer r_k that solve $r_{k-2} = q_k\, r_{k-1} + r_k$] for $k = 1, 2, \ldots$ until some $k = n$ for which $r_n = 0$; then, $r_{n-1} = g$. Graphical interpretation is given in Figure A.2. Note also the following:

Fact A.1 (Bézout's identity) *If $g = gcd(a, b)$, then $g = a\, x_0 + b\, y_0$ for two integers $\{x_0, y_0\}$.*

Proof: The **extended Euclidean algorithm** determines $\{x, y\}$ in Bézout's identity (thus providing a constructive proof of its validity) by running through the quotients q_k computed in Euclid's algorithm in reverse order: defining $z_{n+1} = 0$, $z_n = 1$ and computing $z_k = z_{k+2} - q_k\, z_{k+1}$ for $k = n-1, n-2, \ldots, 1$, it follows that $x_0 = z_2$ and $y_0 = z_1$. This fact may be verified by writing out the Euclidean algorithm with $g = r_{n-1}$:

$$a = q_1\, b + r_1, \quad b = q_2\, r_1 + r_2, \quad r_1 = q_3\, r_2 + r_3 \quad \rightarrow \quad r_{n-4} = q_{n-2}\, r_{n-3} + r_{n-2}, \quad r_{n-3} = q_{n-1}\, r_{n-2} + g,$$

(note that $r_{n-2} = q_n\, g + 0$), then working backwards through this algorithm, solving each step for its last term:

$$g = r_{n-3} - q_{n-1}\, r_{n-2}, \quad r_{n-2} = r_{n-4} - q_{n-2}\, r_{n-3} \quad \rightarrow \quad r_3 = r_1 - q_3\, r_2, \quad r_2 = b - q_2\, r_1, \quad r_1 = a - q_1\, b.$$

Starting with the first expression above, substituting in the second to eliminate r_{n-2}, substituting in the next to eliminate r_{n-3}, etc., ultimately leads to $g = x_0\, a + y_0\, b$, where $x_0 = z_2$ and $y_0 = z_1$ are linear combinations (as formulated previously) of the integers q_i appearing in Euclid's algorithm. □

[19] The OOP paradigm also has certain weaknesses for which it is sometimes criticized. Principal among these criticisms is the substantial code overhead associated with its implementation, which tends to prevent self-optimizing compilers from generating maximally fast executable code for narrowly targeted applications, like cryptography and high-performance computing.

[20] Note that $\gcd(a, b, c) = \gcd(a, \gcd(b, c)) = \gcd(b, \gcd(c, a)) = \gcd(c, \gcd(a, b))$.

The above two algorithms extend easily,

- from integer rings, endowed with Euclidean division giving an integer quotient and remainder,
- to polynomial rings, endowed with Euclidean division giving an polynomial quotient and remainder.

The code RR_gcd.m implements Euclid's algorithm, working over integer or polynomial objects using the *same* object-oriented code, to find the GCD g of $\{a, b\}$ [this code also returns the q_i and n generated in the process]. Using these results, the code RR_bezout.m then implements the extended Euclidean algorithm [again, working over integer or polynomial objects] to find an $\{x_0, y_0\}$ pair solving Bézout's identity $g = a\,x_0 + b\,y_0$.

The general solution of the Diophantine equation [again, working over integer or polynomial objects],

$$f = a\,x + b\,y, \tag{A.1a}$$

as implemented in RR_diophantine.m, is then given, assuming f is evenly divisible by g, as follows:

$$\text{Define: } c = f/g,\ r = b/g,\ t = a/g. \quad \text{Then: } x = c\,x_0 + k\,r \text{ and } y = c\,y_0 - k\,t \text{ for any } k. \tag{A.1b}$$

Note that the solution $\{x, y\}$ for $k = 0$ above is just the scaling (by c) of $\{x_0, y_0\}$ in Bézout's identity in Fact A.1, as determined by the extended Euclidean algorithm derived above. Taking $k \neq 0$ in the solution for $\{x, y\}$ both adds and subtracts the term $k\,a\,b/g$ to the RHS of (A.1a). The "best" solution with the smallest y (that is, with the smallest integer y, or the polynomial y of smallest degree if applying this entire algorithm to polynomials) is then given by taking $k = c\,y_0/t$, which effectively makes y as small as possible in (A.1b).

A.7.2 Operator overloading for easy manipulation of transfer functions

The definition of basic arithmetic operations ($+$, $-$, $\times$, $\div$, ...) for objects that are not simple integer, real, or complex numbers (e.g., row vectors representing the coefficients of polynomials, as implemented in RR_poly.m) is referred to as **operator overloading**. Leveraging classes that define various basic operations on such objects, writing bug-free code for higher-level problems becomes significantly easier.

The definition of the transfer function class in RR_tf.m, formed as a rational function of numerator and denominator polynomial objects from the polynomial class RR_poly.m, allows one to perform basic arithmetic operations on objects in this class on the computer. Thus, defining a CT plant $G(s)$ and controller $D(s)$ (see §8.2.3 and §8.3.3), important equations like

$$G(s) = \frac{b(s)}{a(s)}, \quad D(s) = \frac{y(s)}{x(s)}, \quad T(s) = \frac{G(s)\,D(s)}{1 + G(s)\,D(s)} = \frac{b(s)\,y(s)}{a(s)\,x(s) + b(s)\,y(s)} = \frac{g(s)}{f(s)}. \tag{A.2}$$

can be computed quite simply in code. For example, taking $G(s) = 1/(s+1)$ and $D(s) = (s+z)/(s+p)$ where z and p are not yet specified, we can simply write

```
>> syms z p, G=RR_tf(1,[1 1]), D=RR_tf([1 z],[1 p]), T=G*D/(1+G*D)
```

rather than writing out several complicated function calls or computing by hand, reducing the chance of error.

Further, given some plant $G(s)$ and some desired target $f(s)$ specifying the poles of the closed-loop transfer function $T(s)$ in (A.2), the derivation in §A.7.1 gives the denominator $x(s)$ and numerator $y(s)$ of the controller $D(s)$ such that, when used in closed loop, we achieve this desired target, which is accomplished simply by solving the polynomial Diophantine equation $a(s)\,x(s) + b(s)\,y(s) = f(s)$ for the unknowns $x(s)$ and $y(s)$, and selecting from all possible solutions of this problem that solution with the lowest order $y(s)$.

Becoming skilled at implementing and extending such class definitions yourself, rather than relying on existing implementations, allows you to include features you deem important. For example, the implementation of the polynomial and transfer function classes given in RR_poly.m and RR_tf.m facilitate the use of symbolics in the definition of polynomials and transfer functions, which as illustrated in the example above is convenient; this valuable feature is currently lacking in Matlab's built-in **tf** class.

Appendix B

Assorted mathematical foundations

B.1 Complex arithmetic

Complex arithmetic is based on the mathematical construct[1] $\mathrm{i} \triangleq \sqrt{-1}$. A **complex number**[2] $z = a + b\,\mathrm{i}$ is a number with both a **real part**, $a = \Re(z)$, and an **imaginary part**, $b = \Im(z)$. If $b = 0$, z is said to be **real**; if $a = 0$, z is said to be **imaginary** (aka **pure imaginary**). Complex numbers may also be written in **polar form** $z = Re^{\mathrm{i}\theta}$, where $R = |z| = \sqrt{a^2 + b^2} \geq 0$ is referred to as the **magnitude** or **modulus** of z and[3] $\theta = \measuredangle z = \operatorname{atan2}(b, a) \in (-\pi, \pi]$ is referred to as the **phase** of z. Note that, for any integer k, $e^{\mathrm{i}\theta} = e^{\mathrm{i}(\theta+2\pi k)} = \cos\theta + \mathrm{i}\sin\theta$ (**Euler's formula**); thus, $e^{\mathrm{i}\pi} + 1 = 0$ (**Euler's identity**). Complex numbers near the origin are best understood in the **complex plane**, as illustrated in Figure B.1a. Note that complex numbers are added, subtracted, multiplied, and divided as if i were a normal algebraic variable, then simplified by leveraging the fact that $\mathrm{i}^2 = -1$. For example, $(a + b\,\mathrm{i})(c + d\,\mathrm{i}) = (ac - bd) + (ad + bc)\mathrm{i}$.

The n**'th roots** of a complex number z (written as $z = Re^{\mathrm{i}\theta}$ for real R and θ) are given analytically by:

$$\lambda^n = z \quad \Rightarrow \quad \lambda_j = R^{1/n} e^{\mathrm{i}(\theta+2\pi j)/n} \quad \text{for } j \in [0, \ldots, n-1].$$

This formula is illustrated graphically by example in Figure B.1b. The **principal** n**'th root** of a real number z, denoted $\sqrt[n]{z}$, is defined here as the (positive) real root with $j = 0$ in the above formula if $z > 0$, and as the (negative) real root with $j = \frac{n-1}{2}$ in the above formula if both $z < 0$ and n is odd.

Note that, if $z_{k+1} = \sigma z_k$, then $z_k = \sigma^k z_0$ and thus $|z| \xrightarrow[k\to\infty]{} \infty$ if $|\sigma| > 1$, whereas $z \xrightarrow[k\to\infty]{} 0$ if $|\sigma| < 1$. If $dz(t)/dt = \lambda z(t)$, then $z(t) = e^{\lambda t} z(0)$ and thus $|z| \xrightarrow[t\to\infty]{} \infty$ if $\Re(\lambda) > 0$, whereas $z \xrightarrow[t\to\infty]{} 0$ if $\Re(\lambda) < 0$.

---

[1] Half of the scientific literature refers to this construct as i, the other half calls it j. We call it i.

[2] Complex numbers are generalized by **quaternions**, which are discussed in detail in §6.3.2.2.

[3] Noting that $\operatorname{atan} x \in [0, \pi/2)$ if $x \geq 0$, the following definition is useful to remove ambiguity (see Figure B.1a):

$$\operatorname{atan2}(b, a) \triangleq \begin{cases} \operatorname{atan}|b/a| \cdot \operatorname{sgn} b & \text{if } b \neq 0,\ a > 0, \\ \pi/2 \cdot \operatorname{sgn} b & \text{if } b \neq 0,\ a = 0, \\ (\pi - \operatorname{atan}|b/a|) \cdot \operatorname{sgn} b & \text{if } b \neq 0,\ a < 0, \\ 0 & \text{if } b = 0,\ a \geq 0, \\ \pi & \text{if } b = 0,\ a < 0, \end{cases} \quad \text{where} \quad \operatorname{sgn} b = \begin{cases} -1 & \text{if } b < 0, \\ 0 & \text{if } b = 0, \\ 1 & \text{if } b > 0; \end{cases} \tag{B.1}$$

note that $\operatorname{atan2}(b, a) \in (-\pi, \pi]$.

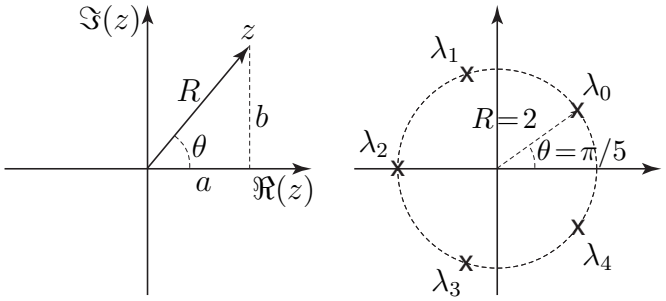


Figure B.1: (a) The relationship between the real part a and imaginary part b of the complex number $z = a + b\,\mathrm{i}$ and its polar components R and θ in the complex plane. (b) The fifth roots of $z = -32$: $\lambda_0 = 2e^{\mathrm{i}\pi/5}$, $\lambda_1 = 2e^{\mathrm{i}3\pi/5}$, $\lambda_2 = 2e^{\mathrm{i}\pi} = -2$, $\lambda_3 = 2e^{\mathrm{i}7\pi/5}$, $\lambda_4 = 2e^{\mathrm{i}9\pi/5}$; $\lambda_2 \triangleq \sqrt[5]{-32}$ is referred to as the principal root.

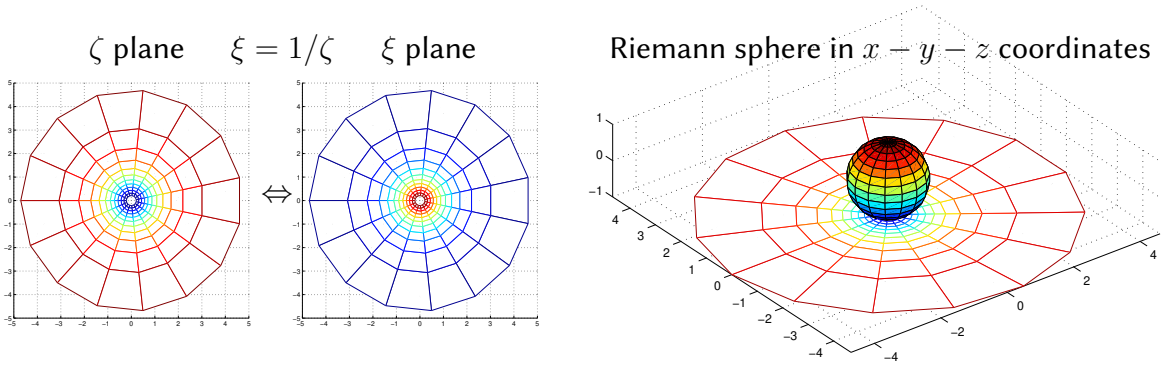


Figure B.2: (left) Two related complex planes which together introduce the concept of the **extended complex plane** (that is, the complex plane ζ together with a **point at infinity**, which maps to the origin in the ξ plane). (right) The **Riemann sphere** obtained by mapping the complex plane ζ (illustrated by a "web" in the plane $z = -1$) onto the unit sphere; note that, with the origin of the ζ plane mapping to the south pole, the point at infinity maps to the north pole, and the "web" maps to corresponding longitude and latitude lines on the sphere. Alternatively, the Riemann sphere may be constructed by mapping from the complex plane ξ, with the origin of the ξ plane mapping to the north pole and its "point at infinity" mapping to the south pole.

The relationship between complex numbers far from the origin in the complex plane $\zeta = \zeta_r + \mathrm{i}\,\zeta_i = Re^{i\theta}$ is best understood by considering a related complex plane $\xi = \xi_r + \mathrm{i}\,\xi_i = re^{i\phi}$, where all points except the origin in the two planes are related by the **transition map** $\zeta = 1/\xi$, and thus $r = 1/R$ and $\phi = -\theta$. Taking $R \to \infty$ for any θ in the ζ plane is equivalent to taking $r \to 0$ for $\phi = -\theta$ in the ξ plane. This introduces the notion of the **extended complex plane**: that is, the complex plane ζ together with the **point at infinity** (referred to in the singular), identified as the origin in the ξ plane, to which ζ maps as $R \to \infty$ for any θ (a point which one might humorously identify as the location of the **restaurant at the end of the universe**).

A geometric interpretation of the entire extended complex plane (Figure B.2) is given by mapping the ζ and ξ planes to and from the **Riemann sphere**, with $x^2 + y^2 + z^2 = 1$, via the **stereographic projections**[4]

$$\zeta = (x + \mathrm{i}y)/(1 - z) \quad \Leftrightarrow \quad z = (|\zeta|^2 - 1)/(|\zeta|^2 + 1), \text{ with } x = \zeta_r(1 - z) \text{ and } y = \zeta_i(1 - z), \tag{B.2a}$$

$$\xi = (x - \mathrm{i}y)/(1 + z) \quad \Leftrightarrow \quad z = (1 - |\xi|^2)/(1 + |\xi|^2), \text{ with } x = \xi_r(1 + z) \text{ and } y = -\xi_i(1 + z). \tag{B.2b}$$

Approaching the origin in the ζ plane (that is, approaching the south pole of the Riemann sphere by taking $x \to 0$, $y \to 0$, and $z \to -1$) corresponds to moving toward infinity in the ξ plane, whereas approaching the origin in the ξ plane (that is, approaching the north pole of the Riemann sphere by taking $x \to 0$, $y \to 0$, and $z \to 1$) corresponds to moving toward infinity in the ζ plane.

[4] From the formulas at left, together with $x^2 + y^2 + z^2 = 1$, it may be shown that $|\zeta|^2 = \zeta_r^2 + \zeta_i^2 = (1 + z)/(1 - z)$, and that $|\xi|^2 = \xi_r^2 + \xi_i^2 = (1 - z)/(1 + z)$, from which the formulas at the right follow easily.

Points with $|\zeta|^2 < 1$ and thus $|\xi|^2 > 1$ correspond to the southern hemisphere of the Riemann sphere, whereas points with $|\zeta|^2 > 1$ and thus $|\xi|^2 < 1$ correspond to the northern hemisphere of the Riemann sphere.
The injective forward and inverse mappings in (B.2a), relating any finite complex ζ to a distinct $\{x, y, z\}$ on the unit sphere excluding the north pole ($z = 1$), reveal that this stereographic projection is **bijective**, with a one-to-one correspondence between the complex plane ζ and the unit sphere excluding the north pole.
The injective forward and inverse mappings in (B.2b), relating any finite complex ξ to a distinct $\{x, y, z\}$ on the unit sphere excluding the south pole ($z = -1$), reveal that this stereographic projection is also bijective, with a one-to-one correspondence between the complex plane ξ and the unit sphere excluding the south pole.

Example B.1 To infinity, and beyond. Consider a curve (actually, a line segment) s in the complex plane ζ, starting from $\zeta = -1$ and proceeding to the right along the negative real axis, through the origin, and further along the positive real axis, ending at the point $\zeta = 1$.

Now define $t = 1/s$. The corresponding curve t proceeds from $\zeta = -1$, along the negative real axis to the *left*, all the way out to "$\zeta = -\infty$", then suddenly jumps all the way over to "$\zeta = +\infty$", and continues to proceed along the positive real axis, to the left, finally ending at the point $\zeta = 1$.

Without the helpful interpretations, above, of the transition map $\zeta = 1/\xi$ and the stereographic projection $\zeta = (x + iy)/(1 - z)$, the above-described behavior of what should be (?) the simple curve t might, at first, seem somewhat bizarre. With these helpful interpretations, however, this behavior is much easier to visualize and explain. In the complex plane ξ, the curve t proceeds from $\xi = -1$ to the right, through the origin, ending at $\xi = 1$ [identical to the behavior of the curve s in the complex plane ζ].

Even more revealing, on the Riemann sphere, the curve s proceeds:

- from a starting point on the $0°$ latitude line (aka the equator) and the $180°$ longitude line,
- down the $180°$ longitude line and through the $-90°$ latitude point (aka the south pole), and finally
- back up the $0°$ longitude line (aka the prime meridian) to a terminal point on the equator.

Analogously, the curve t proceeds:

- from the same starting point on the equator and the $180°$ longitude line,
- *up* the $180°$ longitude line and through the $+90°$ latitude point (aka the north pole), and finally
- back *down* the prime meridian to the same terminal point on the equator.

The symmetry of this interpretation is self evident. In the setting of the Riemann sphere, it is seen that the point at the north pole is "just another point" representing a complex number; on the Riemann sphere, a curve passes through the north pole just as smoothly as a curve passes through the south pole, or any other point. The interpretation that there is really just only one "point at infinity" (that is, the point corresponding to the north pole on the Riemann sphere) is thus entirely reasonable, and reconciling of your intuition about the behavior of (what should be) simple curves as they move far from the origin in the complex plane. △

Further interpretation of the behavior of complex numbers far from the origin given in the discussion of zeros at the infinity point in root locus plots (see §11).

B.1.1 Counting zeros minus poles inside a contour: Cauchy's argument principle

Another result relating to complex arithmetic that we use in this book is Cauchy's argument principle, which is now stated and proved.

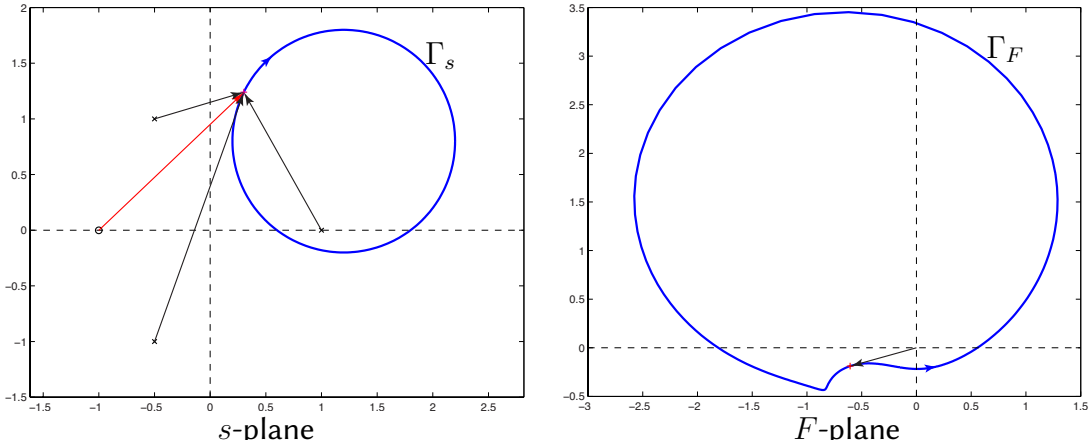


Figure B.3: Cauchy's argument principle (Fact B.1). Each point $\overline{F}$ on the contour Γ_F in the F-plane is found by applying the transform $F(s)$ to the corresponding point $\overline{s}$ on the contour Γ_s in the s-plane.

Fact B.1 (Cauchy's argument principle) *If a clockwise, closed contour Γ_s in the s-plane encircles Z zeros and P poles of a rational function $F(s)$, then the corresponding contour Γ_F in the F-plane [related to the s-plane by the mapping $F(s)$] makes $N = Z - P$ clockwise encirclements of the origin. Defining $L = F - 1$, the corresponding contour Γ_L in the L-plane makes $N = Z - P$ clockwise encirclements of the point $L = -1$.*

Proof: Consider first the change in the phase of $F(\overline{s})$ as $\overline{s}$ traverses the contour Γ_s (in the s-plane) one full circuit in the clockwise direction. Writing $F(s)$ [a **conformal mapping** from the s-plane to the F-plane] as

$$F(s) = K\frac{(s-z_1)(s-z_2)\cdots(s-z_m)}{(s-p_1)(s-p_2)\cdots(s-p_n)},$$

the phase of $F(\overline{s})$ at any given point $\overline{s}$ on the contour Γ_s is given by the sum of the phases of the vectors[5] from the m zeros to the point $\overline{s}$ minus the sum of the phases from the n poles to the point $\overline{s}$:

$$\angle F(\overline{s}) = [\angle(\overline{s}-z_1) + \angle(\overline{s}-z_2) + \ldots + \angle(\overline{s}-z_m)] - [\angle(\overline{s}-p_1) + \angle(\overline{s}-p_2) + \ldots + \angle(\overline{s}-p_n)].$$

As $\overline{s}$ traverses the contour Γ_s one full circuit in the clockwise direction, the contribution to $\angle F(\overline{s})$ from each zero and pole which are outside the closed contour Γ_s returns to its original value (before the circuit was begun). However, as $\overline{s}$ traverses the contour Γ_s one full circuit in the clockwise direction, the contribution to $\angle F(\overline{s})$ from each of the Z zeroes inside the closed contour Γ_s *decreases* by $360°$, whereas the contribution to $\angle F(\overline{s})$ from each of the P poles inside the closed contour Γ_s *increases* by $360°$. Thus, $\angle F(\overline{s})$ after the circuit is $(P-Z)\cdot 360°$ larger than $\angle F(\overline{s})$ before the circuit.

Now consider the change in the phase of $\overline{F} = F(\overline{s})$ as $\overline{F}$ traverses the contour Γ_F (in the F-plane) one full circuit as $\overline{s}$ traverses Γ_s one full circuit in the clockwise direction. The phase of the point $\overline{F}$ (in the F-plane) is measured simply by the angle of the vector from the origin of the F-plane to the point $\overline{F}$. This angle decreases [resp., increases] by $360°$ for every clockwise [resp., counterclockwise] encirclement of the origin executed by the contour Γ_F. Thus, $\angle\overline{F}$ after the circuit is $-N \cdot 360°$ larger than $\angle\overline{F}$ before the circuit, where N is the number of clockwise encirclements of the origin executed by the contour Γ_F.

Thus, for the phase of $F(\overline{s})$ as $\overline{s}$ traverses the clockwise, closed contour Γ_s in the s-plane to change by the same amount as does the phase of $\overline{F}$ as $\overline{F}$ traverses the closed contour Γ_F in the F-plane, it follows that the contour Γ_F makes $N = Z - P$ clockwise encirclements of the origin in the F-plane, which corresponds immediately to $N = Z - P$ clockwise encirclements of the point $L(s) = -1$ in the plane $L(s) = F(s) - 1$. □

[5]The phase of each vector is measured counterclockwise from horizontal, and is assumed to vary smoothly as the contour is traversed (that is, no mod $360°$ command is used to keep the angles in a prespecified range).

B.2 Polynomials and their roots

B.2.1 Roots of a quadratic polynomial

Complex arithmetic facilitates the determination of the roots of a quadratic polynomial analytically. That is,

$$\lambda^2 + Q\lambda + R = 0 \quad \Rightarrow \quad \lambda_\pm = \frac{-Q \pm \sqrt{-D}}{2} \quad \text{where} \quad D = 4R - Q^2,$$

as may be verified by substitution. The quantity D, referred to as the **discriminant**, characterizes the nature of the solution. For real Q and R: if $D > 0$, there are two complex-conjugate roots, if $D = 0$, there are two identical, real roots, and if $D < 0$, there are two distinct, real roots.

B.2.2 Roots of a cubic polynomial

Similarly, we may also determine the roots of a cubic polynomial analytically. Starting with the **normal form**

$$\lambda^3 + P\lambda^2 + Q\lambda + R = 0,$$

we first define $\lambda = x - P/3$ and substitute, giving

$$x^3 + qx + r = 0,$$

where $q = Q - P^2/3$ and $r = R + 2P^3/27 - PQ/3$. This **reduced form** is solved first for its roots x_j for $j \in [1, 2, 3]$, after which the roots $\lambda_j = x_j - P/3$ solving the corresponding equation in normal form may be determined immediately. The **discriminant** in this case, $d = r^2/4 + q^3/27$, again characterizes the nature of the solution. For real q and r,

- if $d > 0$, there is one real and two complex-conjugate roots,
- if $d = 0$, there are three real roots, at least two of which are identical, and
- if $d < 0$, there are three distinct, real roots.

As may be verified by substitution, in the first two of these cases (with $d \geq 0$), the roots are given by **Cardano's formula** (note that the imaginary part of the complex roots goes to zero as $d \to 0$)

$$x_1 = u_+ + u_-, \quad x_{2,3} = -\frac{u_+ + u_-}{2} \pm \mathrm{i}\sqrt{3} \cdot \frac{u_+ - u_-}{2} \quad \text{where} \quad u_\pm = \sqrt[3]{-r/2 \pm \sqrt{d}}.$$

In the third case (with $d < 0$), the so-called **casus irreducibilis**, the three distinct, real roots are given by

$$x_j = v\,\cos(\theta/3 + 2\pi j/3) \quad \text{where} \quad v = 2\sqrt{-q/3}, \quad \theta = \operatorname{acos}\left(\frac{-r/2}{\sqrt{-q^3/27}}\right).$$

B.2.3 Roots of higher-order polynomials

A closed-form solution of the roots of a quartic polynomial, due originally to Ferrari, also exists, though it is complicated. Unfortunately, a general closed-form solution for quintic and higher-order polynomials does not exist, and must instead be found iteratively [for example, by finding the eigenvalues of a corresponding matrix in companion form, as discussed in §4.4 of *NR*.] In fact, the lack of a closed-form expression for solving the roots of the characteristic polynomial of a matrix (for $n \geq 5$) is the reason that the matrix eigenvalue problem must, in general, be solved iteratively.

B.2.4 Factoring polynomials: the fundamental theorem of algebra

Proofs of the fundamental theorem of algebra have a long and rich history. The first complete proof is often attributed to Gauss, though this is a matter of some debate (for a historical summary, see Fine & Rosenberger 1997). The proof provided below, summarized in Santos (2007), is particularly elementary.

Fact B.2 (The fundamental theorem of algebra) *Any n'th-order monic polynomial $p(s)$ with complex coefficients c_k has exactly n complex roots (including multiplicities), and thus may be written*

$$p(s) = s^n + c_{n-1}s^{n-1} + \ldots + c_1 s + c_0 = (s-p_1)(s-p_2)\cdots(s-p_{n-1})(s-p_n).$$

Proof: Let $P_m(s) = s^m + a_{m,m-1}s^{m-1} + \ldots + a_{m,1}s + a_{m,0}$, where $m \geq 2$, be an m'th-order monic polynomial with m complex coefficients $a_{m,k}$ for $k = 0, \ldots, m-1$. We first prove that $P_m(s)$ has at least one root, which we call p_m, and thus may be written $P_m(s) = (s-p_m)(s^{m-1} + a_{m-1,m-1}s^{m-2} + \ldots + a_{m-1,1}s + a_{m-1,0}) = (s-s_m)P_{m-1}(s)$. Starting with the n'th-order monic polynomial $P_n(s) = s^n + a_{n,n-1}s^{n-1} + \ldots + a_{n,1}s + a_{n,0}$, repeated application of this fact $n-1$ times, on the successively lower-order monic polynomials $P_{n-1}(s)$, $P_{n-2}(s)$, etc., thus proves Fact B.2.

To prove that $P_m(s) = s^m + a_{m,m-1}s^{m-1} + \ldots + a_{m,1}s + a_{m,0}$ has a root p_m, we simply assume that it doesn't (that is, that $P_m(s) \neq 0$ for all $s \in \mathbb{C}$) and show that this assumption leads to a contradiction, thereby establishing that $P_m(s)$ in fact has at least one complex root p_m. To accomplish this, for real r and ϕ, define

$$f_m(r,\phi) = \frac{1}{P_m(s)}\Big|_{s=re^{\mathrm{i}\phi}} = \frac{1}{r^m e^{\mathrm{i}m\phi} + a_{m,m-1}r^{m-1}e^{\mathrm{i}(m-1)\phi} + \ldots + a_{m,1}re^{\mathrm{i}\phi} + a_{m,0}}. \tag{B.3}$$

Note that the denominator is continuously differentiable in both r and ϕ, and by assumption is never zero. Thus, $f_m(r,\phi)$ is also continuously differentiable in r and ϕ. Now define

$$F_m(r) = \int_0^{2\pi} f_m(r,\phi)\, d\phi. \tag{B.4}$$

By **Leibniz's integration rule**[6], $F_m(r)$ is continuously differentiable in r, and

$$\frac{dF_m(r)}{dr} = \int_0^{2\pi} \frac{\partial f_m(r,\phi)}{\partial r}\, d\phi.$$

Noting that

$$\frac{\partial f_m(r,\phi)}{\partial r} = -\frac{mr^{m-1}e^{\mathrm{i}m\phi} + (m-1)a_{m,m-1}r^{m-2}e^{\mathrm{i}(m-1)\phi} + \ldots + a_{m,1}e^{\mathrm{i}\phi}}{(r^m e^{\mathrm{i}m\phi} + a_{m,m-1}r^{m-1}e^{\mathrm{i}(m-1)\phi} + \ldots + a_{m,1}re^{\mathrm{i}\phi} + a_{m,0})^2},$$
$$\frac{\partial f_m(r,\phi)}{\partial \phi} = -\frac{\mathrm{i}mr^m e^{\mathrm{i}m\phi} + \mathrm{i}(m-1)a_{m,m-1}r^{m-1}e^{\mathrm{i}(m-1)\phi} + \ldots + \mathrm{i}a_{m,1}re^{\mathrm{i}\phi}}{(r^m e^{\mathrm{i}m\phi} + a_{m,m-1}r^{m-1}e^{\mathrm{i}(m-1)\phi} + \ldots + a_{m,1}re^{\mathrm{i}\phi} + a_{m,0})^2} = \mathrm{i}r\frac{\partial f_m(r,\phi)}{\partial r},$$

it follows that

$$\frac{dF_m(r)}{dr} = \frac{1}{\mathrm{i}r}\int_0^{2\pi} \frac{\partial f_m(r,\phi)}{\partial \phi}\, d\phi = \frac{1}{\mathrm{i}r} f_m(r,\phi)\Big|_{\phi=0}^{\phi=2\pi} = \frac{1}{\mathrm{i}r}\left[\frac{1}{P_m(re^{\mathrm{i}2\pi})} - \frac{1}{P_m(re^{\mathrm{i}0})}\right] = 0.$$

Thus, $F_m(r)$ is constant, and thus $F_m(r) = F_m(0) = \int_0^{2\pi} f(0,\phi)\, d\phi = 2\pi/P_m(0)$; that is, $F_m(r)$ is a nonzero constant. However, (B.3)-(B.4) imply that $F_m(r) \to 0$ as $r \to \infty$. This is a contradiction, which means that the assumption that $P_m(s)$ does not have a root p_m is false. □

[6] **Leibniz's integration rule** states simply that, if $f(r,s)$ and $\partial f(r,s)/\partial r$ are continuous, then $F(r) = \int_a^b f(r,s)\, ds$ is differentiable with respect to r, with $dF(r)/dr = \int_a^b \frac{\partial f(r,s)}{\partial r}\, ds$; that is, that the derivative with respect to r may be pulled outside the integral.

B.2.5 Continuity of a polynomial's roots as a function of its coefficients

Fact B.2 established that any polynomial can be factored via its roots. We now establish that these roots vary continuously as the coefficients of the polynomial are varied. We begin with a significant preliminary result.

Fact B.3 (Rouchés theorem) *Let $p(s)$ and $d(s)$ be complex polynomial functions[7] on the simply-connected domain Ω in the complex plane s. Let Γ be a* **Jordan curve** *(that is, a closed curve that has only a single multiple point where it closes upon itself) within Ω. Define P as the number of zeros of $p(s)$ on the interior of Γ, and Z as the number of zeros of $z(s) = p(s) + d(s)$ on the interior of Γ. If $|d(s)| < |p(s)|$ for all s on Γ, then $P = Z$.*

Proof: Define $F(s) = z(s)/p(s)$, and note that $|F(s) - 1| = |d(s)/p(s)| < 1$, and thus the real part of $F(s)$ is positive, for all s on Γ. By the same logic as in the proof of Cauchy's argument principle (Fact B.1), $\angle F(s)$ after a circuit over Γ is $(P - Z) \cdot 360°$ larger than $\angle F(s)$ before the circuit. As the real part of $F(s)$ is positive everywhere on Γ, $\angle F(s)$ does not change by more that $180°$ as s traverses Γ; thus, $P - Z = 0$. □

Fact B.4 (Continuity of polynomial roots) *Let $p(s)$ be an n'th-order polynomial which may be factored (Fact B.2) such that*

$$p(s) = s^n + a_{n-1}s^{n-1} + \ldots + a_0 = (s - p_1)^{q_1}(s - p_2)^{q_2} \cdots (s - p_P)^{q_P}. \tag{B.5a}$$

Let $\rho = \min_{i \neq j} |p_i - p_j|$. For any ϵ with $0 < \epsilon < \rho/2$, there exists a $\delta > 0$ such that any n'th-order polynomial

$$z(s) = s^n + b_{n-1}s^{n-1} + \ldots + b_0 = (s - z_1)^{r_1}(s - z_2)^{r_2} \cdots (s - z_Z)^{r_Z} \tag{B.5b}$$

with $|b_j - a_j| < \delta$ for $j = 0, 1, \ldots, n-1$ has exactly q_i zeros in each disk $|s - p_i| < \epsilon$ for $i = 1, 2, \ldots, P$. That is, each individual root moves in the complex plane s an amount less than ϵ, for any sufficiently small ϵ, if the coefficients of the polynomial a_i each changes by an amount less than a correspondingly small δ.

Proof (Henrici 1974): Note first that q_i is the multiplicity of the i'th root, p_i, of the polynomial $p(s)$, that r_i is the multiplicity of the i'th root, z_i, of the polynomial $z(s)$, and that $\sum_{i=1}^{P} q_i = \sum_{i=1}^{Z} r_i = n$. Note also that both $p(s)$ and $z(s)$ are taken above to be monic polynomials without loss of generality. Consider the contour Γ_i given by the circle in the s plane, around the root p_i, satisfying $|s - p_i| = \epsilon$. Defining $d(s) = p(s) - z(s)$ and $c_j = a_j - b_j$, noting the assumption that $|c_j| = |a_j - b_j| < \delta$ for all j, we may write

$$d(s) = c_{n-1}s^{n-1} + c_{n-2}s^{n-2} + \ldots + c_0 \quad \Rightarrow \quad |d(s)| \leq \delta|s|^{n-1} + \delta|s|^{n-2} + \ldots + \delta;$$

it thus follows for all s on Γ_i that

$$|d(s)| \leq \delta\, \mu_i(\epsilon) \quad \text{where} \quad \mu_i(\epsilon) \triangleq \sum_{j=0}^{n-1} (|p_i| + \epsilon)^j.$$

Further, for all s on Γ_i, it follows directly from the factored form of (B.5a) that

$$|p(s)| \geq \epsilon^{q_i}(\rho - \epsilon)^{n-q_i} \quad \text{for } i = 1, \ldots, P.$$

Thus, taking $\delta < \epsilon^{q_i}(\rho - \epsilon)^{n-q_i}/\mu_i(\epsilon)$ for each $i = 1, \ldots, P$, it follows that $|d(s)| < |p(s)|$ for all s on each Γ_i, and thus Fact B.3 applies. That is, both $p(s)$ and $z(s)$ have exactly q_i roots inside the disk centered at p_i of radius ϵ, for each $i = 1, \ldots, P$, for any sufficiently small ϵ so long as δ is also made sufficiently small. □

[7] An **analytic function** is a function that is equal to its own Taylor series in some neighborhood of every point, as discussed further in §B of *NR*. We mention analytic functions here only because Rouchés theorem (Fact B.3) may easily be extended from complex polynomial functions to complex analytic functions, though this extension is beyond the scope of the present discussion.

B.2.6 Location of a polynomial's roots with respect to the imaginary axis

Assume $f(s) = f_n s^n + f_{n-1} s^{n-1} + \ldots + f_1 s + f_0$ is a real polynomial (that is, a polynomial with real coefficients) of degree n (that is, $f_n \neq 0$); $f(s)$ is said to be **singular** if $f_{n-1} = 0$ and **nonsingular** otherwise. We will sometimes write $f(s) = [f(s)]_{even} + [f(s)]_{odd}$ where $[f(s)]_{even}$ and $[f(s)]_{odd}$ contain the terms of $f(s)$ with even and odd powers of s, respectively.

The Routh test, implemented as the routh function in RR_poly, counts how many roots of $f(s)$ are in the LHP, on the imaginary axis, and in the RHP [denoted $\{N_-(f),\, N_0(f),\, N_+(f)\}$, respectively, and often referred to as the **inertia** of $f(s)$], without requiring the computation of the roots themselves, which is computationally expensive for large n. Following Meinsma (1995), we derive this test by first proving three preliminary lemmata.

Fact B.5 *Consider an n'th-degree, nonsingular, real polynomial $f(s) = f_n\, s^n + f_{n-1}\, s^{n-1} + \ldots + f_1\, s + f_0$. Defining $\eta^* = f_n / f_{n-1}$, $f(s)$ has the same imaginary roots as the $(n-1)$'th-degree polynomial*

$$q(s) = f_{n-1}\, s^{n-1} + (f_{n-2} - \eta^* f_{n-3})\, s^{n-2} + f_{n-3}\, s^{n-3} + (f_{n-4} - \eta^* f_{n-5})\, s^{n-4} + f_{n-5}\, s^{n-5} + \ldots \tag{B.6}$$

Further, $f(s)$ has the same inertia as $q(s)$, plus one additional root in the LHP if $f_n / f_{n-1} > 0$, and in the RHP if $f_n / f_{n-1} < 0$.

Proof: Note that $f_{n-1} \neq 0$ because $f(s)$ is nonsingular. Define

$$q_\eta(s) = f(s) - \eta\, s\, (f_{n-1}\, s^{n-1} + f_{n-3}\, s^{n-3} + \ldots) = \begin{cases} ([f(s)]_{even} - \eta\, s\, [f(s)]_{odd}) + [f(s)]_{odd} & \text{even } n, \\ ([f(s)]_{odd} - \eta\, s\, [f(s)]_{even}) + [f(s)]_{even} & \text{odd } n. \end{cases} \tag{B.7}$$

Then $q_\eta(\mathrm{i}\omega) = [q_\eta(\mathrm{i}\omega)]_{even} + [q_\eta(\mathrm{i}\omega)]_{odd} = 0$ iff $[q_\eta(\mathrm{i}\omega)]_{even} = 0$ and $[q_\eta(\mathrm{i}\omega)]_{odd} = 0$, as $[q_\eta(\mathrm{i}\omega)]_{even}$ is real, and $[q_\eta(\mathrm{i}\omega)]_{odd}$ is imaginary. It follows from the expressions for both even and odd n at right in (B.7) that, for all η, $q_\eta(\mathrm{i}\omega) = 0$ iff both $f_{even}(\mathrm{i}\omega) = 0$ and $f_{odd}(\mathrm{i}\omega) = 0$; thus, $q(\mathrm{i}\omega) = q_{\eta^*}(\mathrm{i}\omega) = 0$ iff $f(\mathrm{i}\omega) = 0$.

Thus, $f(s) = q_0(s)$ and $q(s) = q_{\eta^*}(s)$ have the same imaginary roots. Further, by Fact B.4, the roots of $q_\eta(s)$ vary smoothly as η is varied from 0 up to (or, down to) η^*; that is, roots do not "jump" between the LHP and the RHP as η is varied over this range. As $\eta \to \eta^*$, $q_\eta(s)$ in (B.7) approaches the $(n-1)$'th-degree polynomial $q(s)$ in (B.6); to see what happens in this limit, we may write

$$\begin{aligned} q_\eta(s) &= (f_n - \eta\, f_{n-1})\, s^n + f_{n-1}\, s^{n-1} + (f_{n-2} - \eta\, f_{n-3})\, s^{n-2} + f_{n-3}\, s^{n-3} + (f_{n-4} - \eta\, f_{n-5})\, s^{n-4} + \ldots \\ &= \left(\frac{f_n - \eta\, f_{n-1}}{f_{n-1}}\, s + 1 \right) r_\eta(s) \end{aligned} \tag{B.8}$$

where $r_\eta(s) \to q(s)$ [see (B.6)] as $\eta \to \eta^*$. It is seen in (B.8) that exactly one root of $q_\eta(s)$ approaches infinity as $\eta \to \eta^*$; this root "starts out" [for $\eta = 0$, that is, for $q_0(s) = f(s)$] in the LHP if $f_n / f_{n-1} > 0$, and in the RHP if $f_n / f_{n-1} < 0$. Of the remaining roots of the n'th-degree polynomial $f(s)$, the number in the LHS, on the imaginary axis, and in the RHS are precisely the same as for the $(n-1)$'th-degree polynomial $q(s)$. □

Fact B.6 *Let $f(s)$ be an n'th-degree, singular, real polynomial that is neither odd nor even, and let $a(s)$ be a real even polynomial with $a(i\omega) > 0$. Define $f_a(s)$ and $f_b(s)$ such that*

$$f(s) = f_a(s) + f_b(s) \quad \textit{where} \quad \begin{cases} f_a(s) = [f(s)]_{even}, & f_b(s) = [f(s)]_{odd} \quad \textit{even } n, \\ f_a(s) = [f(s)]_{odd}, & f_b(s) = [f(s)]_{even} \quad \textit{odd } n. \end{cases} \tag{B.9}$$

If $a(s)$ is chosen such that the degree of $\{a(s)\, f_b(s)\}$ is $n-1$, then the n'th-degree polynomials $f(s)$ and $q(s) = f_a(s) + a(s)\, f_b(s)$ have the same inertia, and $q(s)$ is nonsingular (i.e., $q_{n-1} \neq 0$).

Proof: If $f_a(s)$ is even (resp., odd), then $\{(1-\lambda)+\lambda\, a(s)\}\, f_b(s)$ is odd (resp., even). For $0 \leq \lambda \leq 1$, define the n'th-degree polynomial

$$q_\lambda(s) = f_a(s) + \{(1-\lambda)+\lambda\, a(s)\}\, f_b(s).$$

Noting that $a(\mathrm{i}\,\omega) > 0$, it follows as in the proof of Fact B.5 that $q_\lambda(\mathrm{i}\,\omega) = 0$ iff $f_a(\mathrm{i}\,\omega) = 0$ and $f_b(\mathrm{i}\,\omega) = 0$ [that is, iff $f(\mathrm{i}\,\omega) = 0$], independent of λ. Thus, $f(s)$ and $q_\lambda(s)$ have the same imaginary roots, independent of λ. Further, by Fact B.4, the n roots of $q_\lambda(s)$ vary smoothly as λ is varied; i.e., roots do not "jump" between the LHP and the RHP as λ is varied. Thus, $f(s) = q_0(s)$ and $q(s) = q_1(s)$ have the same inertia. □

Fact B.7 *Let $f(s)$ be a real n'th-degree odd or even polynomial. Defining $r(s) = f(s) + f'(s)$, it follows that $N_+(f) = N_-(f) = N_+(r)$, and $r(s)$ is nonsingular (i.e., $r_{n-1} \neq 0$).*

Proof: It follows from the fact that $f(s)$ is either odd or even that $r_{n-1} \neq 0$, and that $f(s)$ has the same number of RHP roots as LHP roots [that is, $N_+(f) = N_-(f)$]. Define $q_\epsilon(s) = f(s) + \epsilon f'(s)$; it follows from Fact B.6 [with $a(s) = \epsilon$] that, for any $\epsilon > 0$, $r(s)$ and $q_\epsilon(s)$ have the same inertia. We thus focus on $q_\epsilon(s)$ in the limit that $\epsilon \to 0$. If $s = \mathrm{i}\,\omega$ is a root of multiplicity k of the polynomial $f(s) = q_0(s)$, then $s = \mathrm{i}\,\omega$ is a root of multiplicity $k-1$ of the polynomial $q_\epsilon(s)$ for $\epsilon > 0$, and exactly one root of $q_\epsilon(s)$ moves, continuously, away from $s = \mathrm{i}\,\omega$ as ϵ is increased from zero. To quantify what direction this root moves as ϵ is increased from zero, denoting $f^{(k)} = d^k f(s)/ds^k$ and $s = \mathrm{i}\,\omega + \delta$ as the modified root, write the Taylor series expansion of $q_\epsilon(s)$ about $s = \mathrm{i}\,\omega$:

$$\begin{aligned} q_\epsilon(\mathrm{i}\,\omega+\delta) &= \frac{\delta^{k-1}}{(k-1)!}\left[q_\epsilon^{(k-1)}(s)\right]_{s=\mathrm{i}\,\omega} + \frac{\delta^k}{k!}\left[q_\epsilon^{(k)}(s)\right]_{s=\mathrm{i}\,\omega} + \ldots \\ &= \frac{\delta^{k-1}}{(k-1)!}\left[\cancelto{0}{f^{(k-1)}(\mathrm{i}\,\omega)} + \epsilon f^{(k)}(\mathrm{i}\,\omega)\right] + \frac{\delta^k}{k!}\left[f^{(k)}(\mathrm{i}\,\omega) + \epsilon f^{(k+1)}(\mathrm{i}\,\omega)\right] + \ldots \\ &= \frac{\delta^{k-1}}{k!}\, f^{(k)}(\mathrm{i}\,\omega)\left[\epsilon\, k + \delta + \epsilon\,\delta\,\frac{f^{(k+1)}(\mathrm{i}\,\omega)}{f^{(k)}(\mathrm{i}\,\omega)} + \ldots\right] \end{aligned} \tag{B.10}$$

where the "…" terms are higher order in ϵ, and thus negligible for small ϵ. By Fact B.4, δ is proportional to ϵ for small ϵ; the third term in brackets in (B.10) is thus also negligible compared to the first two terms. Solving for $q_\epsilon(\mathrm{i}\,\omega+\delta) = 0$ thus shows that, in addition to the $k-1$ roots fixed (with $\delta = 0$), the remaining root is given by $s = \mathrm{i}\,\omega + \delta$ with $\delta \approx -k\,\epsilon < 0$ for sufficiently small $\epsilon > 0$ (that is, the remaining root moves into the LHP as ϵ is increased from zero). Since $f(s) = q_0(s)$ and $r(s) = q_1(s)$, it follows that $N_+(f) = N_+(r)$. □

Fact B.8 (The Routh Test) *The inertia $\{N_-(f),\, N_0(f),\, N_+(f)\}$ of an n'th-degree polynomial $f(s)$ may be found via application of the following three cases to polynomials successively smaller and smaller degree:*

Case 1: If $f(s)$ is nonsingular (that is, if $f_{n-1} \neq 0$), then define $q(s)$ as in (B.6). It follows that

$$\{N_-(f),\, N_0(f),\, N_+(f)\} = \begin{cases} \{N_-(q)+1,\, N_0(q),\, N_+(q)\} & \text{if } f_n/f_{n-1} > 0, \\ \{N_-(q),\, N_0(q),\, N_+(q)+1\} & \text{if } f_n/f_{n-1} < 0; \end{cases}$$

$\{N_-(q),\, N_0(q),\, N_+(q)\}$ may then be found by applying Fact B.8 to the $(n-1)$'th-degree polynomial $q(s)$.

Case 2: If $f(s)$ is singular (that is, if $f_{n-1} = 0$) but neither even nor odd, then define $f_a(s)$ and $f_b(s)$ as in (B.9). Since $f_{n-1} = 0$, $f_b(s)$ has degree $n-1-2k$ for some $k > 0$; define $a(s) = 1+(-s^2)^k$. Defining $q(s) = f_a(s)+a(s)\, f_b(s)$, it follows that $\{N_-(f),\, N_0(f),\, N_+(f)\} = \{N_-(q),\, N_0(q),\, N_+(q)\}$, which may be found by applying Case 1 of Fact B.8 to the nonsingular n'th-degree polynomial $q(s)$.

Case 3: If $f(s)$ is either even or odd, then define $r(s) = f(s) + f'(s)$. It follows that $N_+(f) = N_-(f) = N_+(r)$, where $N_+(r)$ may be found by applying Case 1 of Fact B.8 to the nonsingular n'th-degree polynomial $r(s)$.

Proof: Case 1 follows immediately from Fact B.5. Case 2 follows from Fact B.6, noting that $a(\mathrm{i}\,\omega) > 0$ for real ω, and that the degree of $\{a(s)\, f_b(s)\}$ is $n-1$. Case 3 follows immediately from Fact B.7. □

B.2.7 Location of a polynomial's roots with respect to the unit circle

Assume $f(z) = f_n z^n + f_{n-1} z^{n-1} + \ldots + f_1 z + f_0$ is a real polynomial of degree n; $f(z)$ is said to be **regular** if $f_0 \neq 0$ (and, thus, $f(z)$ does not have any roots at the origin) and **irregular** otherwise. A polynomial $f(z)$ is said to be **symmetric** if $f_{n-j} = f_j$ for $j = 0, \ldots, n$, and **antisymmetric** if $f_{n-j} = -f_j$ for $j = 0, \ldots, n$.

The Bistritz test, implemented as the `bistritz` function in `RR_poly`, counts how many roots of $f(z)$ are inside the unit circle, on the unit circle, and outside the unit circle [denoted $\{N_i(f),\, N_u(f),\, N_o(f)\}$, respectively, and which we will refer to as the **stationarity** of $f(z)$], without requiring the computation of the roots themselves. The Bistritz test is thus useful for determining the stability of discrete-time systems, much as the Routh test (§B.2.6) is useful for determining the stability of continuous-time systems. Following loosely the related derivation for the Routh test, we derive this test by first proving four preliminary lemmata.

Fact B.9 *Consider an n'th-degree real polynomial $f(z) = f_n\, z^n + f_{n-1}\, z^{n-1} + \ldots + f_1\, z + f_0$ with $f_n(1) \neq 0$. Defining $f^r(z) \triangleq z^n\, f_n(1/z) = f_0\, z^n + f_1\, z^{n-1} + \ldots + f_{n-1}\, z + f_n$, $f_n(z)$ and $f_{n-1}(z)$ have the same roots on the unit circle as the symmetric n'th-degree and $(n-1)$'th-degree polynomials*

$$T_n(z) = f_n(z) + f_n^r(z) \quad \textit{and} \quad T_{n-1}(z) = \frac{f_n(z) - f_n^r(z)}{z-1}.$$

Proof: It follows immediately that T_n is symmetric; note also that $f_n \neq 0$ because $f_n(z)$ is of degree n, and that $f_n(z)$ has no roots at $z = 1$, because $f_n(1) \neq 0$. Writing

$$\begin{aligned} f_n(z) - f_n^r(z) &= (f_n - f_0)z^n + (f_{n-1} - f_1)z^{n-1} + \ldots + (f_1 - f_{n-1})z + (f_0 - f_n) \\ &= (z-1)\big[q_{n-1}z^{n-1} + \ldots + q_{n-2}z^{n-2} + \ldots + q_1 z + q_0\big], \end{aligned}$$

it follows that the q_i may be determined by solving

$$\begin{pmatrix} 1 & & & & \\ -1 & 1 & & & \\ & \ddots & \ddots & & \\ & & -1 & 1 & \\ & & & -1 & \end{pmatrix} \begin{pmatrix} q_{n-1} \\ q_{n-2} \\ \vdots \\ q_1 \\ q_0 \end{pmatrix} = \begin{pmatrix} f_n - f_0 \\ f_{n-1} - f_1 \\ \vdots \\ f_1 - f_{n-1} \\ f_0 - f_n \end{pmatrix}$$

By the first and last rows of this matrix equation, it follows that $q_{n-1} = q_0 = f_n - f_0$; by the second and next-to-last rows, it follows that $q_{n-2} = q_1$, etc.; thus, T_{n-1} is also symmetric.

Since $f(z)$ is a real polynomial, $z_+ = e^{\mathrm{i}\,\phi}$ is a root [on the unit circle] of $f_n(z)$ iff $e^{-\mathrm{i}\,\phi} = 1/z_+$ is also root of $f_n(z)$, that is, iff $z_+ = e^{\mathrm{i}\,\phi}$ is a root of $f_n^r(z) \triangleq z^n\, f_n(1/z) = f_0\, z^n + f_1\, z^{n-1} + \ldots + f_{n-1}\, z + f_n$. Thus, $f_n(z)$ and $f_n^r(z)$, and thus also $T_n(z)$ and $T_{n-1}(z)$, have the same roots on the unit circle. □

Fact B.10 *Consider an n'th-degree real polynomial $f(z) = \{(z-1)T_{n-1}(z) + T_n(z)\}/2$, with $f_n(1) \neq 0$ and either $T_{n-1}(z) \neq 0$ or $T_n(0) = 0$ (or both), such that $T_n(z)$ and $T_{n-1}(z)$ are symmetric, and $f_n(z)$, $T_n(z)$, and $T_{n-1}(z)$ have the same roots on the unit circle. Denote $T_n(z) = t_{n,n}z^n + \ldots + t_{n,0}$ and $T_{n-1}(z) = t_{n-1,n-1}z^n + \ldots + t_{n-1,0}$. Define λ as the smallest non-negative integer such that $t_{n-1,\lambda} \neq 0$. Consider the $(n-2)$'th-degree polynomial $T_{n-2}(z)$ defined such that*

$$zT_{n-2}(z) = \delta(z^{\lambda+1} + z^{-\lambda})T_{n-1}(z) - T_n(z) \qquad \textit{where} \qquad \delta = \begin{cases} t_{n,0}/t_{n-1,\lambda} & \textit{if } T_{n-1}(z) \neq 0, \\ 0 & \textit{if } t_{n,0} = 0. \end{cases}$$

It follows that $f(z)$ has the same stationarity as $f_{n-1}(z) \triangleq \{(z-1)T_{n-2} + T_{n-1}(z)\}/2$, plus one additional root inside the unit circle if $T_n(1)/T_{n-1}(1) > 0$, and outside the unit circle if $T_n(1)/T_{n-1}(1) < 0$.

Proof:

$$f_n(z) = \{-z\,T_{n-2}(z) + [z - 1 + \delta(z^{\lambda+1} + z^{-\lambda})]T_{n-1}(z)\}/2.$$

Now define

$$q_\eta(z) = f(z) - \eta\, f^r(z) = (f_n - \eta\, f_0)\, z^n + (f_{n-1} - \eta\, f_1)\, z^{n-1} + \ldots + (f_1 - \eta\, f_{n-1})\, z + (f_0 - \eta\, f_n), \quad \text{(B.11)}$$

$$\hat{q}_\lambda(z) = f^r(z) - \lambda\, f(z) = (f_0 - \lambda\, f_n)\, z^n + (f_1 - \lambda\, f_{n-1})\, z^{n-1} + \ldots + (f_{n-1} - \lambda\, f_1)\, z + (f_n - \lambda\, f_0). \quad \text{(B.12)}$$

If $|f_0/f_n| < 1$ [and thus the leading coefficient of $q_\eta(z)$ is nonzero for $\eta = 0$ through $\eta = \eta^* = f_0/f_n$], then, by Fact B.4, the n roots of $q_\eta(z)$ vary smoothly as η is varied from 0 up to (or down to) η^*; that is, roots do not "jump" between inside the unit circle and outside the unit circle as η is varied over this range. Since $q_{\eta^*}(z) = z\,q(z)$, it is seen that exactly one root of $q_\eta(z)$ goes to zero as $\eta \to \eta^*$. Of the remaining roots of the n'th-degree polynomial $f(z)$, the number inside the unit circle, on the unit circle, and outside the unit circle are precisely the same as for the $(n-1)$'th-degree polynomial $q(z)$.

If $|f_0/f_n| > 1$ [and thus the leading coefficient of $\hat{q}_\lambda(z)$ is nonzero for $\lambda = 0$ through $\lambda = \lambda^* = f_n/f_0$], then, by Fact B.4, the n roots of $\hat{q}_\lambda(z)$ vary smoothly as λ is varied from 0 up to (or down to) λ^*; that is, roots do not "jump" between inside the unit circle and outside the unit circle as λ is varied over this range. Since $\hat{q}_{\lambda^*}(z)/\lambda^* = q_{\eta^*}(z) = z\,q(z)$, it is seen that exactly one root of $\hat{q}_\lambda(z)$ goes to zero as $\lambda \to \lambda^*$. Of the remaining roots of the n'th-degree polynomial $f(z)$, the number inside the unit circle, on the unit circle, and outside the unit circle are precisely the same as for the $(n-1)$'th-degree polynomial $q(z)$. □

Fact B.11 *Let $f(z)$ be an n'th-degree, irregular, real polynomial that is neither odd nor even, and let $a(z)$ be a real even polynomial with $a(\mathrm{i}\omega) > 0$. Define $f_a(z)$ and $f_b(z)$ such that*

$$f(z) = f_a(z) + f_b(z) \quad \textit{where} \quad \begin{cases} f_a(z) = [f(z)]_{even}, \quad f_b(z) = [f(z)]_{odd} & \textit{even } n, \\ f_a(z) = [f(z)]_{odd}, \quad f_b(z) = [f(z)]_{even} & \textit{odd } n. \end{cases} \quad \text{(B.13)}$$

If $a(z)$ is chosen such that the degree of $\{a(z)\, f_b(z)\}$ is $n-1$, then the n'th-degree polynomials $f(z)$ and $q(z) = f_a(z) + a(z)\, f_b(z)$ have the same stationarity, and $q(z)$ is regular (i.e., $q_{n-1} \neq 0$).

Proof: If $f_a(z)$ is even (resp., odd), $\{(1-\lambda) + \lambda\, a(z)\}\, f_b(z)$ is odd (resp., even). For $0 \leq \lambda \leq 1$, define the n'th-degree polynomial

$$q_\lambda(z) = f_a(z) + \{(1-\lambda) + \lambda\, a(z)\}\, f_b(z).$$

Noting that $a(\mathrm{i}\omega) > 0$, it follows as in the proof of Fact B.9 that $q_\lambda(\mathrm{i}\omega) = 0$ iff $f_a(\mathrm{i}\omega) = 0$ and $f_b(\mathrm{i}\omega) = 0$ [that is, iff $f(\mathrm{i}\omega) = 0$], independent of λ. Thus, $f(z)$ and $q_\lambda(z)$ have the same imaginary roots, independent of λ. Further, by Fact B.4, the n roots of $q_\lambda(z)$ vary smoothly as λ is varied; i.e., roots do not "jump" between the LHP and the RHP as λ is varied. Thus, $f(z) = q_0(z)$ and $q(z) = q_1(z)$ have the same stationarity. □

Fact B.12 *Let $f(z)$ be a real n'th-degree odd or even polynomial. Defining $r(z) = f(z) + f'(z)$, it follows that $N_o(f) = N_i(f) = N_o(r)$, and $r(z)$ is regular (i.e., $r_{n-1} \neq 0$).*

Proof: It follows from the fact that $f(z)$ is either odd or even that $r_{n-1} \neq 0$, and that $f(z)$ has the same number of RHP roots as LHP roots [that is, $N_+(f) = N_-(f)$]. Define $q_\epsilon(z) = f(z) + \epsilon f'(z)$; it follows from Fact B.10 [with $a(z) = \epsilon$] that, for any $\epsilon > 0$, $r(z)$ and $q_\epsilon(z)$ have the same stationarity. We thus focus on $q_\epsilon(z)$ in the limit that $\epsilon \to 0$. If $z = \mathrm{i}\omega$ is a root of multiplicity k of the polynomial $f(z) = q_0(z)$, then $z = \mathrm{i}\omega$ is a root of multiplicity $k-1$ of the polynomial $q_\epsilon(z)$ for $\epsilon > 0$, and exactly one root of $q_\epsilon(z)$ moves (by Fact B.4,

continuously) away from $z = \mathrm{i}\,\omega$ as ϵ is increased from zero. To quantify what direction this root moves as ϵ is increased from zero, denoting $f^{(k)} = d^k f(z)/dz^k$, we write the Taylor series expansion of $q_\epsilon(z)$ around $z = \mathrm{i}\,\omega$:

$$\begin{aligned} q_\epsilon(\mathrm{i}\,\omega + \delta) &= \frac{\delta^{k-1}}{(k-1)!}\left[q_\epsilon^{(k-1)}(z)\right]_{z=\mathrm{i}\,\omega} + \frac{\delta^k}{k!}\left[q_\epsilon^{(k)}(z)\right]_{z=\mathrm{i}\,\omega} + \ldots \\ &= \frac{\delta^{k-1}}{(k-1)!}\left[\cancelto{0}{f^{(k-1)}(\mathrm{i}\,\omega)} + \epsilon f^{(k)}(\mathrm{i}\,\omega)\right] + \frac{\delta^k}{k!}\left[f^{(k)}(\mathrm{i}\,\omega) + \epsilon f^{(k+1)}(\mathrm{i}\,\omega)\right] + \ldots \\ &= \frac{\delta^{k-1}}{k!}\, f^{(k)}(\mathrm{i}\,\omega)\left[\epsilon\, k + \delta + \epsilon\,\delta\frac{f^{(k+1)}(\mathrm{i}\,\omega)}{f^{(k)}(\mathrm{i}\,\omega)} + \ldots\right] \end{aligned} \tag{B.14}$$

where the "..." terms are higher order in ϵ, and thus negligible for small ϵ. By Fact B.4, δ is nearly proportional to ϵ for sufficiently small ϵ; the third term in brackets in (B.14) is thus also negligible compared to the first two terms. Solving for $q_\epsilon(\mathrm{i}\,\omega + \delta) = 0$ thus shows that, in addition to the $k-1$ roots fixed at $\delta = 0$, the remaining root is given by $\delta \approx -k\,\epsilon < 0$ for sufficiently small $\epsilon > 0$ (that is, the remaining root moves into the LHP as ϵ is increased from zero). Since $f(z) = q_0(z)$ and $r(z) = q_1(z)$, it follows that $N_+(f) = N_+(r)$. □

Fact B.13 (The Bistritz Test) *The stationarity $\{N_i(f),\, N_u(f),\, N_o(f)\}$ of an n'th-degree polynomial $f(z)$ may be found via application of the following three cases to polynomials successively smaller and smaller degree:*

Case 1: If $f(z)$ is regular (that is, if $f_0 \neq 0$), then define $q(z)$ as in (B.11). It follows that

$$\{N_i(f),\, N_u(f),\, N_o(f)\} = \begin{cases} \{N_i(q)+1,\, N_u(q),\, N_o(q)\} & \text{if } |f_0/f_n| > 1, \\ \{N_i(q),\, N_u(q),\, N_o(q)+1\} & \text{if } |f_0/f_n| < 1; \end{cases}$$

$\{N_i(q),\, N_u(q),\, N_o(q)\}$ may then be found by applying Fact B.13 to the $(n-1)$'th-degree polynomial $q(z)$.

Case 2: If $f(z)$ is irregular (that is, if $f_{n-1} = 0$) but neither even nor odd, then define $f_a(z)$ and $f_b(z)$ as in (B.13). Since $f_{n-1} = 0$, $f_b(z)$ has degree $n-1-2k$ for some $k > 0$; define $a(z) = 1 + (-z^2)^k$. Defining $q(z) = f_a(z) + a(z)\, f_b(z)$, it follows that $\{N_i(f),\, N_u(f),\, N_o(f)\} = \{N_i(q),\, N_u(q),\, N_o(q)\}$, which may be found by applying Fact B.13 to the regular n'th-degree polynomial $q(z)$.

Case 3: If $f(z)$ is either even or odd, then define $r(z) = f(z) + f'(z)$. It follows that $N_i(f) = N_o(f) = N_o(r)$, where $N_o(r)$ may be found by applying Fact B.13 to the regular n'th-degree polynomial $r(z)$.

Proof: Case 1 follows immediately from Fact B.9. Case 2 follows from Fact B.10, noting that $a(\mathrm{i}\,\omega) > 0$ for real ω, and that the degree of $\{a(z)\, f_b(z)\}$ is $n-1$. Case 3 follows immediately from Fact B.11. □

B.2.8 The simplified Routh test

The **Routh test** (§B.2.6) counts how many roots of a polynomial $f(s)$ are in the open LHP, on the imaginary axis, and in the open RHP [referred to as the **inertia** of $f(s)$], without requiring the computation of the roots themselves.

When analyzing a polynomial $f(s)$ in the denominator of a CT transfer function $T(s) = g(s)/f(s)$ simply to determine whether or not all of the roots of $f(s)$ are in the open LHP [and thus $f(s)$ is a **Hurwitz stable polynomial**, and $T(s)$ is stable], the **simplified Routh test**, implemented as the `routh_simplified` function in `RR_poly`, may be used instead, as defined by the following three-term recurrence:

$$u_n(s) = u_{n,n}\, s^n + u_{n,n-2}\, s^{n-2} + \ldots \qquad = f_n\, s^n + f_{n-2}\, s^{n-2} + \ldots, \tag{B.15a}$$

$$u_{n-1}(s) = u_{n-1,n-1}\, s^{n-1} + u_{n-1,n-3}\, s^{n-3} + \ldots = f_{n-1}\, s^{n-1} + f_{n-3}\, s^{n-3} + \ldots, \tag{B.15b}$$

$$u_i(s) = u_{i+2}(s) - \alpha_i\, s\, u_{i+1}(s) \;\text{ where }\; \alpha_i = u_{i+2,i+2}/u_{i+1,i+1} \;\text{ for }\; i = n-2, n-3, \ldots, 0; \tag{B.15c}$$

all of the roots of $f(s)$ are in the open LHP iff $\{u_{n,n}, u_{n-1,n-1}, \ldots, u_{1,1}, u_{0,0}\}$ are all nonzero and the same sign.

A useful feature of the simplified Routh test is that one can carry one or more *variables* in a control design formulation, such as a controller gain K, all the way through the test, thereby determining necessary and sufficient conditions on these variables for stability.

The simplified Routh test described above determines whether or not all of the poles of a transfer function $T(s)$ are in the open LHP, a condition sometimes referred to as **absolute stability**. A stricter condition that is sometimes useful to determine is whether or not all of the poles of a transfer function $T(s)$ have a real part to the left of $s = -\sigma$ for some $\sigma > 0$, a condition referred to as **relative stability**. This is easy to determine by applying the simplified Routh test discussed above to the modified transfer function $T(s + \sigma)$.

B.2.9 The simplified Bistritz test

The **Bistritz test**[8] (§B.2.7) counts how many roots of a polynomial $f(z)$ are inside the unit circle, on the unit circle, and outside the unit circle [referred to as the **stationarity** of $f(z)$], without requiring the computation of the roots themselves.

When analyzing a polynomial $f(z)$ in the denominator of a DT transfer function $T(z) = g(z)/f(z)$ simply to determine whether or not all of the roots of $f(z)$ are inside the unit circle [and thus $f(z)$ is a **Schur stable polynomial**, and $T(z)$ is stable], the **simplified Bistritz test**, implemented as the bistritz_simplified function in RR_poly, may be used instead, as defined by the following three-term recurrence [cf. (B.15)]:

$$u_n(z) = f(z) + f^r(z) \quad \text{where} \quad f^r(z) = z^n\, f(1/z) = f_0\, z^n + f_1\, z^{n-1} + \ldots + f_n, \tag{B.16a}$$

$$u_{n-1}(z) = [f(z) - f^r(z)]/(z-1) \quad \text{polynomial division (note: can easily do by hand),} \tag{B.16b}$$

$$u_i(z) = [\alpha_i\,(z+1)\,u_{i+1}(z) - u_{i+2}(z)]/z \quad \text{where} \quad \alpha_i = u_{i+2}(0)/u_{i+1}(0) \text{ for } i = n-2, n-3, \ldots, 0; \tag{B.16c}$$

all of the roots of $f(z)$ are inside the unit circle iff $\{u_n(0), u_{n-1}(0), \ldots, u_1(0), u_0(0)\}$ are all nonzero and the same sign. Note in this test that each of the $u_i(z)$ are **symmetric** (that is, $u_{i,i-j} = u_{i,j}$ for $i = 0, \ldots, n$ and $j = 1, \ldots, i$), so the work associated with this test, if programmed efficiently, is similar to that associated with the simplified Routh test described previously, in which each of the $u_i(s)$ considered is either **even** or **odd**.

As with the simplified Routh test, a useful feature of the simplified Bistritz test, is that one can carry one or more variables in the control design formulation, such as a controller gain K, all the way through the test, thereby determining necessary and sufficient conditions on such variables for stability.

The simplified Bistritz test described above determines whether or not all of the poles of a transfer function $T(z)$ are inside the unit circle, a condition sometimes referred to as **absolute stability**. A stricter condition that is sometimes useful to determine is whether or not all of the poles of a transfer function $T(z)$ are inside a circle of some radius $r < 1$, a condition referred to as **relative stability**. This is easy to determine by applying the simplified Bistritz test discussed above to the modified transfer function $T(z/r)$.

Example B.2 Testing the (full and simplified) Routh and Bistritz codes. Embedded in comments in the header of the RR_poly class definition (easily read by typing **help** RR_poly at the Matlab prompt) are several short, self-explanatory examples that illustrate the (full and simplified) Routh and Bistritz tests working on various polynomials. Note that the full Routh and Bistritz tests must be applied to polynomials defined by numerical values in their coefficients, and reveal the entire inertia and stationarity, respectively, of the associated polynomial $f(s)$ or $f(z)$ [that is, how many stable, marginally stable, and unstable poles that the transfer function $T(s) = g(s)/f(s)$ or $T(z) = g(z)/f(z)$ has]. As mentioned previously, the simplified Routh and Bistritz tests, which only indicate the stability of the associated transfer functions, may be applied to polynomials with symbolic variables, like K, included in their coefficients. Used in this fashion, necessary and sufficient constraints on these coefficients for stability of $T(s)$ or $T(z)$ may easily be determined. △

[8] The **Schur-Cohn test** and **Jury test** are alternatives to the Bistritz test for determining whether or not $f(z)$ is Schur. We focus on the Bistritz test in this discussion, due to its strong similarity to the Routh test.

B.3 Vector calculus: div, grad, curl, and Gauss, Stokes, Helmholtz

Vector calculus is the study of scalar and vector fields that vary in space. We focus on three dimensional (3D) problems. Define the **dot product** $\vec{a} \cdot \vec{b} = a_1 b_1 + a_2 b_2 + a_3 b_3 = \|\vec{a}\| \, \|\vec{b}\| \cos(\theta)$ and the **cross product** $\vec{a} \times \vec{b} = \|\vec{a}\| \, \|\vec{b}\| \sin(\theta) \, \vec{n}$, where $\theta = \measuredangle(\vec{a}, \vec{b})$ is the angle between $\vec{a}$ and $\vec{b}$ [see §1.3.1 in *NR*], and $\vec{n}$ is a unit vector perpendicular to the plane containing $\vec{a}$ and $\vec{b}$, oriented via the right-hand rule. It follows that

$$\vec{a} \times \vec{b} = -\vec{b} \times \vec{a} = \begin{vmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} = (a_2 b_3 - a_3 b_2)\,\mathbf{e}_1 + (a_3 b_1 - a_1 b_3)\,\mathbf{e}_2 + (a_1 b_2 - a_2 b_1)\,\mathbf{e}_3 \tag{B.17}$$

$$\vec{a} \times \vec{b} = \big[\vec{a}\big]_\times \vec{b}, \quad \big[\vec{a}\big]_\times \triangleq \begin{pmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{pmatrix} \tag{B.18}$$

$$(\vec{a} \times \vec{b}) \cdot (\vec{c} \times \vec{d}) = (\vec{a} \cdot \vec{c})(\vec{b} \cdot \vec{d}) - (\vec{a} \cdot \vec{d})(\vec{b} \cdot \vec{c}) \tag{B.19}$$

$$\vec{a} \cdot (\vec{b} \times \vec{c}) = \vec{b} \cdot (\vec{c} \times \vec{a}) = \vec{c} \cdot (\vec{a} \times \vec{b}) \tag{B.20}$$

$$\vec{a} \times (\vec{b} \times \vec{c}) = \vec{b}(\vec{a} \cdot \vec{c}) - \vec{c}(\vec{a} \cdot \vec{b}) \tag{B.21}$$

$$\|\vec{a} \times \vec{b}\|^2 = \|\vec{a}\|^2 \, \|\vec{b}\|^2 - (\vec{a} \cdot \vec{b})^2 \tag{B.22}$$

Note that $\vec{a} \times \vec{a} = \big[\vec{a}\big]_\times \vec{a} = 0$, and that $\big[\vec{a}\big]_\times = -\big[\vec{a}\big]_\times^T$. The study of vector calculus is facilitated by the **del** operator (denoted by the **nabla** symbol ∇), which is defined using summation notation such that

$$\nabla = \mathbf{e}_j \frac{\partial}{\partial x_j}; \qquad \text{thus, in 3D,} \quad \nabla = \mathbf{e}_1 \frac{\partial}{\partial x_1} + \mathbf{e}_2 \frac{\partial}{\partial x_2} + \mathbf{e}_3 \frac{\partial}{\partial x_3}. \tag{B.23}$$

With this operator, the **divergence** $\nabla \cdot \vec{v}$, the **gradient** $\nabla \phi$, and the **curl** $\nabla \times \vec{v}$, are respectively

$$\nabla \cdot \vec{v} = \frac{\partial v_1}{\partial x_1} + \frac{\partial v_2}{\partial x_2} + \frac{\partial v_3}{\partial x_3}, \tag{B.24a}$$

$$\nabla \phi = \mathbf{e}_1 \frac{\partial \phi}{\partial x_1} + \mathbf{e}_2 \frac{\partial \phi}{\partial x_2} + \mathbf{e}_3 \frac{\partial \phi}{\partial x_3}, \tag{B.24b}$$

$$\nabla \times \vec{v} = \begin{vmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \\ \frac{\partial}{\partial x_1} & \frac{\partial}{\partial x_2} & \frac{\partial}{\partial x_3} \\ v_1 & v_2 & v_3 \end{vmatrix} = \Big(\frac{\partial v_3}{\partial x_2} - \frac{\partial v_2}{\partial x_3}\Big)\mathbf{e}_1 + \Big(\frac{\partial v_1}{\partial x_3} - \frac{\partial v_3}{\partial x_1}\Big)\mathbf{e}_2 + \Big(\frac{\partial v_2}{\partial x_1} - \frac{\partial v_1}{\partial x_2}\Big)\mathbf{e}_3. \tag{B.24c}$$

The **Laplacian**, $\Delta\phi$, is defined using summation notation such that

$$\Delta \phi = \nabla \cdot \nabla \phi = \frac{\partial^2 \phi}{\partial x_j^2}; \qquad \text{thus, in 3D,} \quad \Delta \phi = \Big(\frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \frac{\partial^2}{\partial x_3^2}\Big)\phi. \tag{B.25}$$

The **bilaplacian** (a.k.a. **biharmonic**) is defined as the Laplacian of the Laplacian, $\Delta\Delta\phi$; thus, in 3D,

$$\begin{aligned} \Delta\Delta\phi &= \Big(\frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \frac{\partial^2}{\partial x_3^2}\Big)\Big(\frac{\partial^2}{\partial x_1^2} + \frac{\partial^2}{\partial x_2^2} + \frac{\partial^2}{\partial x_3^2}\Big)\phi \\ &= \Big(\frac{\partial^4}{\partial x_1^4} + \frac{\partial^4}{\partial x_2^4} + \frac{\partial^4}{\partial x_3^4} + 2\frac{\partial^4}{\partial x_1^2 \, \partial x_2^2} + 2\frac{\partial^4}{\partial x_1^2 \, \partial x_3^2} + 2\frac{\partial^4}{\partial x_2^2 \, \partial x_3^2}\Big)\phi. \end{aligned} \tag{B.26}$$

Identities related to the gradient, divergence, curl, and Laplacian [all easily verified via substitution] include:

$$\nabla(\phi\,\psi) = \phi\nabla\psi + \psi\nabla\phi \tag{B.27}$$

$$\nabla \cdot (\phi\nabla\psi) = \phi\Delta\psi + \nabla\phi \cdot \nabla\psi \tag{B.28}$$

$$\nabla \cdot (\phi\,\vec{v}) = \phi\nabla \cdot \vec{v} + \vec{v} \cdot \nabla\phi \tag{B.29}$$

$$\nabla \times (\nabla\phi) = 0 \tag{B.30}$$

$$\nabla \cdot (\nabla \times \vec{v}) = 0 \tag{B.31}$$

$$\nabla(\vec{u} \cdot \vec{v}) = (\vec{u} \cdot \nabla)\vec{v} + (\vec{v} \cdot \nabla)\vec{u} + \vec{u} \times (\nabla \times \vec{v}) + \vec{v} \times (\nabla \times \vec{u}) \tag{B.32}$$

$$\nabla \cdot (\vec{u} \times \vec{v}) = \vec{v} \cdot (\nabla \times \vec{u}) - \vec{u} \cdot (\nabla \times \vec{v}) \tag{B.33}$$

$$\nabla \times (\phi\vec{v}) = (\nabla\phi) \times \vec{v} + \phi\nabla \times \vec{v} \tag{B.34}$$

$$\nabla \times (\vec{u} \times \vec{v}) = \vec{u}(\nabla \cdot \vec{v}) - \vec{v}(\nabla \cdot \vec{u}) + (\vec{v} \cdot \nabla)\vec{u} - (\vec{u} \cdot \nabla)\vec{v} \tag{B.35}$$

$$\nabla \times (\nabla \times \vec{v}) = \nabla(\nabla \cdot \vec{v}) - \Delta\vec{v} \tag{B.36}$$

Gauss's theorem (a.k.a. the **divergence theorem**) relates the integral over a (3D or 2D) volume V of the *divergence* of a vector field $\vec{v}$ to the integral over the (2D or 1D) surface of the volume, ∂V, of the *normal component* of the vector field (note that $d\vec{A}$ is oriented as an outward-facing normal vector):

$$\int_V (\nabla \cdot \vec{v})\, dV = \int_{\partial V} \vec{v} \cdot d\vec{A}. \tag{B.37}$$

Stokes theorem relates the integral over a (2D) area A [which itself may be defined in $\mathbb{R}^3$] of the *curl* of a vector field $\vec{v}$ to the integral over the boundary of the area, ∂A, of the *tangential component* of the vector field (following the **right-hand rule**, $d\vec{s}$ is a counterclockwise-facing tangential vector when $d\vec{A}$ faces the viewer):

$$\int_A (\nabla \times \vec{v}) \cdot d\vec{A} = \oint_{\partial A} \vec{v} \cdot d\vec{s}. \tag{B.38}$$

An important special case of Stoke's theorem, known as **Green's theorem**, is developed by taking $\vec{v} = \{\psi, \phi, 0\}$ and the (2D) area A as lying in the $x - y$ plane, in which case (B.38) reduces immediately to

$$\int_A \Big(\frac{\partial \phi}{\partial x} - \frac{\partial \psi}{\partial y}\Big) dx\, dy = \oint_{\partial A} (\psi\, dx + \phi\, dy). \tag{B.39}$$

The **Helmholtz decomposition** (a.k.a. the **fundamental theorem of vector calculus**), states that any vector field $\vec{v}$ whose curl and divergence vanish at infinity may be decomposed into an **irrotational** part, $-\nabla\phi$, and a **solenoidal** part, $\nabla \times \vec{\psi}$:

$$\vec{v} = -\nabla\phi + \nabla \times \vec{\psi}, \tag{B.40}$$

where ϕ is called a **scalar potential** and $\vec{\psi}$ is called a **vector potential**; by the identities summarized above, the irrotational part is curl free and the solenoidal part is divergence free.

In (3D) **cylindrical coordinates**, the gradient, divergence, curl, and Laplacian may be written

$$\nabla f = \mathbf{e}_r \frac{\partial f}{\partial r} + \mathbf{e}_\phi \frac{1}{r}\frac{\partial f}{\partial \phi} + \mathbf{e}_z \frac{\partial f}{\partial z}, \tag{B.41a}$$

$$\nabla \cdot \vec{v} = \frac{1}{r}\frac{\partial (r\, v_r)}{\partial r} + \frac{1}{r}\frac{\partial v_\phi}{\partial \phi} + \frac{\partial v_z}{\partial z}, \tag{B.41b}$$

$$\nabla \times \vec{v} = \mathbf{e}_r \Big(\frac{1}{r}\frac{\partial v_z}{\partial \phi} - \frac{\partial v_\phi}{\partial z}\Big) + \mathbf{e}_\phi \Big(\frac{\partial v_r}{\partial z} - \frac{\partial v_z}{\partial r}\Big) + \mathbf{e}_z \Big(\frac{1}{r}\frac{\partial (r\, v_\phi)}{\partial r} - \frac{1}{r}\frac{\partial v_r}{\partial \phi}\Big), \tag{B.41c}$$

$$\Delta f = \frac{1}{r}\frac{\partial}{\partial r}\Big(r\,\frac{\partial f}{\partial r}\Big) + \frac{1}{r^2}\frac{\partial^2 f}{\partial \phi^2} + \frac{\partial^2 f}{\partial z^2}. \tag{B.41d}$$

In (3D) **spherical coordinates**, the gradient, divergence, curl, and Laplacian may be written

$$\nabla f = \mathbf{e}_r \frac{\partial f}{\partial r} + \mathbf{e}_\theta \frac{1}{r}\frac{\partial f}{\partial \theta} + \mathbf{e}_\phi \frac{1}{r\,\sin\theta}\frac{\partial f}{\partial \phi}, \tag{B.42a}$$

$$\nabla \cdot \vec{v} = \frac{1}{r^2}\frac{\partial (r^2\, v_r)}{\partial r} + \frac{1}{r\,\sin\theta}\frac{\partial (v_\theta\, \sin\theta)}{\partial \theta} + \frac{1}{r\,\sin\theta}\frac{\partial v_\phi}{\partial \phi}, \tag{B.42b}$$

$$\nabla \times \vec{v} = \mathbf{e}_r \frac{1}{r\,\sin\theta}\Big(\frac{\partial (v_\theta\, \sin\theta)}{\partial \theta} - \frac{\partial v_\theta}{\partial \phi}\Big) + \mathbf{e}_\theta \frac{1}{r}\Big(\frac{1}{\sin\theta}\frac{\partial v_r}{\partial \phi} - \frac{\partial (r v_\phi)}{\partial r}\Big) + \mathbf{e}_\phi \frac{1}{r}\Big(\frac{\partial (r\, v_\theta)}{\partial r} - \frac{\partial v_r}{\partial \theta}\Big), \tag{B.42c}$$

$$\Delta f = \frac{1}{r^2}\frac{\partial}{\partial r}\Big(r^2\,\frac{\partial f}{\partial r}\Big) + \frac{1}{r^2\,\sin\theta}\frac{\partial}{\partial \theta}\Big(\sin\theta\,\frac{\partial f}{\partial \theta}\Big) + \frac{1}{r^2\,\sin^2\theta}\frac{\partial^2 f}{\partial \phi^2}. \tag{B.42d}$$

In (2D) **polar coordinates**, the bilaplacian may be written

$$\begin{aligned}\Delta\Delta f &= \left[\frac{1}{r}\frac{\partial}{\partial r}\left(r\,\frac{\partial}{\partial r}\right)+\frac{1}{r^2}\frac{\partial^2}{\partial\phi^2}\right]\left[\frac{1}{r}\frac{\partial}{\partial r}\left(r\,\frac{\partial}{\partial r}\right)+\frac{1}{r^2}\frac{\partial^2}{\partial\phi^2}\right]f \\ &= \frac{\partial^4 f}{\partial r^4}+\frac{2}{r^2}\frac{\partial^4 f}{\partial r^2\,\partial\phi^2}+\frac{1}{r^4}\frac{\partial^4 f}{\partial\phi^4}+\frac{2}{r}\frac{\partial^3 f}{\partial r^3}-\frac{2}{r^3}\frac{\partial^3 f}{\partial r\,\partial\phi^2}-\frac{1}{r^2}\frac{\partial^2 f}{\partial r^2}+\frac{4}{r^4}\frac{\partial^2 f}{\partial\phi^2}+\frac{1}{r^3}\frac{\partial f}{\partial r}.\end{aligned} \tag{B.43}$$

B.4 Some useful expansions, sums, identities, and definitions

The following identities, derivatives, and sums are often useful[9]:

$$e^{\mathrm{i}\,x} = \cos x + \mathrm{i}\sin x \;\;\Rightarrow\;\; e^{\mathrm{i}\,\pi}+1=0 \tag{B.44}$$

$$e^{\mathrm{i}\,x}e^{-\mathrm{i}\,x} = 1 \;\;\Rightarrow\;\; \cos^2 x+\sin^2 x = 1 \tag{B.45}$$

$$\sin x = (e^{\mathrm{i}x}-e^{-\mathrm{i}x})/(2\mathrm{i}) \tag{B.46}$$

$$\cos x = (e^{\mathrm{i}x}+e^{-\mathrm{i}x})/2 \tag{B.47}$$

$$\sinh x = (e^{x}-e^{-x})/2 \tag{B.48}$$

$$\cosh x = (e^{x}+e^{-x})/2 \tag{B.49}$$

$$\tan x = \sin(x)/\cos(x) \tag{B.50}$$

$$\tanh x = \sinh(x)/\cosh(x) \tag{B.51}$$

$$\sin(x+y) = \sin x\,\cos y+\cos x\,\sin y \tag{B.52}$$

$$\cos(x+y) = \cos x\,\cos y-\sin x\,\sin y \tag{B.53}$$

$$\begin{aligned}\cos x &= 2\cos^2(x/2)-1 = 1-2\sin^2(x/2)\\ &= \cos^2(x/2)-\sin^2(x/2)\end{aligned} \tag{B.54}$$

$$\tan(x+y) = \frac{\tan x+\tan y}{1-\tan x\,\tan y} \tag{B.55}$$

$$2\,\sin(x)\,\cos(y) = \sin(x+y)+\sin(x-y) \tag{B.56}$$

$$2\,\cos(x)\,\cos(y) = \cos(x+y)+\cos(x-y) \tag{B.57}$$

$$2\,\sin(x)\,\sin(y) = \cos(x-y)-\cos(x+y) \tag{B.58}$$

$$\sum_{m=1}^{M}\cos\,m\,x = \frac{\cos[(M+1)x/2]\sin(Mx/2)}{\sin(x/2)} \tag{B.59}$$

$$a\,\sin x+b\,\cos x = r\,\sin(x+\alpha) \tag{B.60}$$

$$a\,\cos x+b\,\sin x = r\,\cos(x-\alpha) \tag{B.61}$$

$$\operatorname{acos} x = \frac{\pi}{2}-\operatorname{asin} x \tag{B.62}$$

$$d(\sin x)/dx = \cos x;\;\; d(\cos x)/dx = -\sin x \tag{B.63}$$

$$d(\tan x)/dx = 1/\cos^2 x \tag{B.64}$$

$$d(\sinh x)/dx = \cosh x;\;\; d(\cosh x)/dx = \sinh x \tag{B.65}$$

$$d(\tanh x)/dx = 1-\tanh^2 x = 1/\cosh^2(x) = \operatorname{sech}^2(x) \tag{B.66}$$

$$d(\operatorname{asin} x)/dx = 1/\sqrt{1-x^2} \tag{B.67}$$

$$d(\operatorname{acos} x)/dx = -1/\sqrt{1-x^2} \tag{B.68}$$

$$d(\operatorname{atan} x)/dx = 1/(1+x^2) \tag{B.69}$$

$$d(\ln x)/dx = 1/x \tag{B.70}$$

$$\sum_{k=1}^{n}k = \frac{n(n+1)}{2} \triangleq T_n \qquad \text{Triangular numbers} \tag{B.71}$$

$$\sum_{k=1}^{n}k^2 = \frac{n(n+1)(2n+1)}{6} \tag{B.72}$$

$$\sum_{k=1}^{n}k^3 = \frac{n^2(n+1)^2}{4} \tag{B.73}$$

$$\sum_{k=1}^{n}k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} \tag{B.74}$$

$$\sum_{k=1}^{n}T_k = \frac{n(n+1)(n+2)}{6} \triangleq Te_n \qquad \text{Tetrahedral numbers} \tag{B.75}$$

$$\sum_{k=1}^{n}Te_k = \frac{n(n+1)(n+2)(n+3)}{24} \triangleq P_n \qquad \text{Pentatope numbers} \tag{B.76}$$

where, in (B.60) and (B.61), $r=\sqrt{a^2+b^2}$ and $\alpha=\operatorname{atan}(b/a)$. The **binomial coefficients** ${}_nC_k$ are the numerical coefficients in both the expansion of an n'th-degree binomial and the expansion of the n'th derivative of the product of two functions (known as the general Leibniz rule),

$$(x+y)^n = \sum_{k=0}^{n}{}_nC_k\,x^{n-k}y^k = {}_nC_0\,x^n+{}_nC_1\,x^{n-1}y^1+\ldots+{}_nC_n\,y^n, \quad (uv)^{(n)} = \sum_{k=0}^{n}{}_nC_k\,u^{(n-k)}v^{(k)}, \tag{B.77a}$$

[9] Pythagoras (Greece, b. 572 BC) is credited with determining (B.71) [known as the *triangular numbers* T_n, as it gives the number of objects (e.g., rocks or coins) in a triangular pack; the Pythagoreans placed a particular mystic significance on the triangular pack with four objects on a side, corresponding to $T_4=10$], Archimedes (Greece, b. 287 BC) with (B.72), Aryabhata (India, b. 476) with (B.73) and (B.75) [which gives the number of spheres (e.g., cannonballs) in a stack of spheres with a trangular base], and Abu Ali al-Hasan ibn al-Hasan ibn al-Haytham (Egypt, b. 965) with (B.74).

where $u^{(n)}$ denotes the n'th derivative of u with respect to its argument. ${}_nC_k$ may be calculated as follows:

$$ {}_nC_k \triangleq \binom{n}{k} \triangleq \frac{n!}{k!\,(n-k)!} \quad \text{for } n \geq k \geq 0; \tag{B.77b} $$

${}_nC_k$ is pronounced n **choose** k (implemented as **nchoosek** in Matlab), as it represents the number of unordered collections of k objects that can be chosen from a set of n distinct objects. The binomial coefficients ${}_nC_k$ for $k = 0$ through $k = n$ appear on the n'th row (counting from zero) of what is referred to as **Pascal's triangle**[10]:

$$
\begin{array}{ccccccccccccccccccccc}
 & & & & & & & & & 1 & & & & & & & & & & \\
 & & & & & & & & 1 & & 1 & & & & & & & & & \\
 & & & & & & & 1 & & 2 & & 1 & & & & & & & & \\
 & & & & & & 1 & & 3 & & 3 & & 1 & & & & & & & \\
 & & & & & 1 & & 4 & & 6 & & 4 & & 1 & & & & & & \\
 & & & & 1 & & 5 & & 10 & & 10 & & 5 & & 1 & & & & & \\
 & & & 1 & & 6 & & 15 & & 20 & & 15 & & 6 & & 1 & & & & \\
 & & 1 & & 7 & & 21 & & 35 & & 35 & & 21 & & 7 & & 1 & & & \\
 & 1 & & 8 & & 28 & & 56 & & 70 & & 56 & & 28 & & 8 & & 1 & &
\end{array}
$$

Note that the triangular numbers B.71, tetrahedral numbers B.75, and pentatope numbers B.76 are enumerated on, respectively, the third, fourth, and fifth diagonals of Pascal's triangle. Implemented as **pascal** in Matlab.

The **polylogarithm** is defined for $|z| < 1$ and any complex s by an infinite sum, and for integer nonpositive $s = -n$ (with $n \geq 0$) by a finite sum, as follows:

$$ Li_s(z) = \sum_{k=1}^{\infty} z^k / k^s, \tag{B.78a} $$

$$ Li_{-n}(z) = \left(z\,\frac{d}{dz} \right)^n \frac{z}{1-z} = \sum_{k=0}^{n} k!\; S(n+1, k+1) \left(\frac{z}{1-z} \right)^{k+1} \tag{B.78b} $$

where the $S(n,k)$, known as the **Stirling numbers of the second kind**, are defined such that

$$ S(n,k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j}\, j\; {}_kC_j\; j^n. \tag{B.79} $$

Define the **Kronecker delta**, δ_{di}, and two finite approximations of **Dirac delta**, $\delta^{\sigma}(t)$ [see Figure 8.1a, noting its $\sigma \to 0$ limit] and $\delta^{\lambda,m}(t)$ for $m \geq 2$ [see Figures 8.1b and 8.1c, noting their $\lambda \to \infty$ limit], such that

$$ \delta_{di} = \begin{cases} 1 & d = i, \\ o & \text{otherwise}, \end{cases} \qquad \delta^{\sigma}(t) = e^{-x^2/(2\sigma^2)} / (\sigma\sqrt{2\pi}), \qquad \delta^{\lambda,m}(t) = \lambda^m\, x^{m-1}\, e^{-\lambda\, x/(m-1)!}. \tag{B.80} $$

The **continuous-time Heaviside step function**, $h_a(t)$ with $a = 0, 1/2$, or 1, and the **discrete-time Heaviside step function**, h_{dk}, are then defined as

$$ h_a(t-d) = \begin{cases} 0 & t < d, \\ a & t = d, \\ 1 & t > d, \end{cases} \quad \Rightarrow \quad \begin{cases} h_{1/2}(t-d) = \lim\limits_{\sigma \to 0} \displaystyle\int_{-\infty}^{t} \delta^{\sigma}(t'-d)\, dt' \\ h_0(t-d) = \lim\limits_{\lambda \to \infty} \displaystyle\int_{-\infty}^{t} \delta^{\lambda,m}(t'-d)\, dt' \end{cases} \tag{B.81a} $$

$$ h_{dk} = \begin{cases} 0 & k < d, \\ 1 & k \geq d. \end{cases} \quad \Rightarrow \quad h_{dk} = \sum_{i=-\infty}^{k} \delta_{di}; \tag{B.81b} $$

[10] The triangular table of binomial coefficients is often incorrectly attributed, via this name, to Blaise Pascal (b. 1623), though it dates back to several earlier sources, the earliest being Pingala's Sanskrit work *Chandas Shastra*, written in the fifth century BC.

$h_0(t)$ is **left-continuous** at $t = 0$, as $h_0(-\epsilon) \to h_0(0)$ as $\epsilon \to 0$ with $\epsilon > 0$, whereas $h_1(t)$ is **right-continuous** at $t = 0$, as $h_1(\epsilon) \to h_1(0)$ as $\epsilon \to 0$ with $\epsilon > 0$. Note also that h_{0k} is often denoted as simply h_k.

Finally, the following Taylor-series expansions, each valid for sufficiently small ϵ, are often useful:

$$e^{\epsilon} = 1 + \epsilon + \frac{\epsilon^2}{2!} + \frac{\epsilon^3}{3!} + \frac{\epsilon^4}{4!} + \ldots \quad \text{(B.82)}$$

$$\ln(1+\epsilon) = \epsilon - \frac{\epsilon^2}{2} + \frac{\epsilon^3}{3} - \frac{\epsilon^4}{4} + \ldots \quad \text{(B.83)}$$

$$\ln\left(\frac{1+\epsilon}{1-\epsilon}\right) = 2\Big(\epsilon + \epsilon^3/3 + \epsilon^5/5 + \ldots\Big) \quad \text{(B.84)}$$

$$\sin(\epsilon) = \epsilon - \frac{\epsilon^3}{3!} + \frac{\epsilon^5}{5!} - \frac{\epsilon^7}{7!} + \ldots \quad \text{(B.85)}$$

$$\cos(\epsilon) = 1 - \frac{\epsilon^2}{2!} + \frac{\epsilon^4}{4!} - \frac{\epsilon^6}{6!} + \ldots \quad \text{(B.86)}$$

$$\tan(\epsilon) = \epsilon + \frac{\epsilon^3}{3} + \frac{2\epsilon^5}{15} + \frac{17\epsilon^7}{315} + \ldots \quad \text{(B.87)}$$

$$\tanh(\epsilon) = \epsilon - \frac{\epsilon^3}{3} + \frac{2\epsilon^5}{15} - \frac{17\epsilon^7}{315} + \ldots \quad \text{(B.88)}$$

$$\operatorname{atan}(\epsilon) = \epsilon - \frac{\epsilon^3}{3} + \frac{\epsilon^5}{5} - \frac{\epsilon^7}{7} + \ldots \quad \text{(B.89)}$$

$$\frac{1}{1-\epsilon} = 1 + \epsilon + \epsilon^2 + \epsilon^3 + \epsilon^4 \ldots \quad \text{(B.90)}$$

$$\frac{1}{(1-\epsilon)^2} = 1 + 2\epsilon + 3\epsilon^2 + 4\epsilon^3 + 5\epsilon^4 + \ldots \quad \text{(B.91)}$$

$$\frac{1}{(1-\epsilon)^3} = 1 + 3\epsilon + 6\epsilon^2 + 10\epsilon^3 + 15\epsilon^4 + \ldots \quad \text{(B.92)}$$

$$\frac{1}{(1-\epsilon)^n} = {}_nC_n + {}_{n+1}C_n\,\epsilon + {}_{n+2}C_n\,\epsilon^2 + \ldots \quad \text{(B.93)}$$

$$\sqrt{1+\epsilon} = 1 + \frac{\epsilon}{2} - \frac{\epsilon^2}{8} + \frac{\epsilon^3}{16} - \frac{5\epsilon^4}{128} + \ldots \quad \text{(B.94)}$$

$$\frac{1}{\sqrt{1-\epsilon}} = 1 + \frac{1}{2}\cdot\epsilon + \frac{1\cdot 3}{2\cdot 4}\cdot\epsilon^2 + \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\cdot\epsilon^3 + \ldots \quad \text{(B.95)}$$

$$\operatorname{asin}(\epsilon) = \epsilon + \frac{1}{2}\cdot\frac{\epsilon^3}{3} + \frac{1\cdot 3}{2\cdot 4}\cdot\frac{\epsilon^5}{5} + \frac{1\cdot 3\cdot 5}{2\cdot 4\cdot 6}\cdot\frac{\epsilon^7}{7} + \ldots \quad \text{(B.96)}$$

Epilog

References

Chapter 1: Cybernetics and **Chapter 2: Embedded Programming**

[1] Knuth, D (1968-2022) *The Art of Computer Programming, vol. 1-4B*. Addison-Wesley. 2-14

[2] Dowd, K, & Severance, C (1998) *High Performance Computing*. O'Reilly.

[3] Hyde, R (2020a) *Write Great Code, vol. 1: Understanding the Machine*, 2nd ed. No starch press.

[4] Hyde, R (2020b) *Write Great Code, vol. 2: Thinking Low-Level, Writing High-Level*, 2nd ed. No starch press.

[5] Hyde, R (2020c) *Write Great Code, vol. 3: Engineering Software*, 2nd ed. No starch press.

[6] L'Ecuyer, P (1999) Tables of linear congruential generators of different sizes and good lattice structure. *Mathematics of Computation* **68** (225), 249-260. 2-15, 2-16

[7] L'Ecuyer, Blouin and Couture (1993) A search for good multiple recursive random number generators *ACM Transactions on Modeling and Computer Simulation* **3** (2), 87-98.

[8] O'Neill, ME (2014) PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation. Harvey Mudd College technical report HMC-CS-2014-0905. 2-16, 2-18

[9] Press, WH, Teukolsky, SA, Vetterling, WT, and Flannery, BP (2007) *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge University Press.

Chapter 3: Sensors, actuators, and interfaces

?

Chapter 4: Communication

?

Chapter 5: Representative integration: Berets

?

Chapter 6: Kinematics & Dynamics

[10] Wie, B (1998) *Space Vehicle Dynamics and Control*. AIAA.

[11] Tewari, A (2007) *Atmospheric and Space Flight Dynamics*. Birkhäuser.

[12] Landau, L, & Lifshitz, E (1976) *Mechanics*. Butterworth/Heinemann.

[13] Ginsberg, J (1995) *Advanced Engineering Dynamics*. Cambridge.

[14] Greenwood, D (1988) *Principles of Dynamics*. Prentice Hall.

[15] Wells, D (1967) *Shaum's outline of theory and problems of Lagrangian dynamics*. McGraw-Hill.

Chapter 7: Numerical Simulation Methods

Chapter 8: Signals & Systems

Chapter 9: Circuits

[16] Carter, B, & Mancini, R (2009) *Op Amps For Everyone*. Texas Instruments.

[17] Jakas, M, & Llopis, F (2007) LC Sine-Wave Oscillators Using General-Purpose Voltage Operational-Amplifiers, The International Journal of Electrical Engineering & Education **44** (3), 244-248.

Chapter 10: Classical Control

[18] Evans, WR (1950) Control System Synthesis by Root Locus Method *AIEE Trans.* **69** part 2, 66-69.

[19] Gelb, A, & Vander Velde, WE (1968). *Multiple-Input Describing Functions and Nonlinear System Design*. McGraw Hill.

[20] Lutovac, MD, Tosic, DV, & Evans, BL (2001) *Filter Design for Signal Processing Using MATLAB and Mathematica*. Prentice-Hall.

[21] Nelson, R (1997) *Flight Stability and Automatic Control*. McGraw-Hill.

[22] McCormick, BW (1995) *Aerodynamics, Aeronautics, and Flight Mechanics*. Wiley.

Chapter 11: Motion Planning

Chapter 12: Error-Correcting Codes

[23] Lin, S, Costello, D.J. (1983) *Error Control Coding: Fundamentals and Applications*. Prentice-Hall.

Chapter 13: Open vs Proprietary Development Models

Chapter 14: Crowd Funding vs Venture Capital

Chapter 15: Computer Aided Design & Manufacturing (CAD/CAM)

Chapter 16: Design Paradigms

Chapter 17: Case study: myMiP

Appendix A: Matlab programming

Appendix B: Assorted mathematical foundations

[24] Adams, D (1980) *The Restaurant at the End of the Universe*, Pan Books.[11]

[25] Davis, RT (1979) Numerical Methods for Coordinate Generation Based on Schwarz-Christoffel Transformations, *AIAA Paper 79-1463*.

[26] Fine, B, & Rosenberger, G (1997) *The Fundamental Theorem of Algebra*, Springer-Verlag.

[27] Henrici, P (1974) *Applied and computational complex analysis, Volume 1*, Wiley.

[28] Meinsma, G (1995) Elementary proof of the Routh-Hurwitz test. *Systems & Control Letters*, **25**, 237-242.

[29] Santos, JC (2007) The fundamental theorem of algebra deduced from elementary calculus. *The Mathematical Gazette*, **91**, 302-303.

[30] Schey, HM (2005) *Div, grad, curl, and all that: an informal text on vector calculus*. WW Norton & Co.

[11] **Circular Reasoning**: see explanation of **Reasoning, Circular** in footnote on Page x.