2020 International Conference on Unmanned Aircraft Systems (ICUAS)
Athens, Greece. September 1-4, 2020

Point Cloud-Based Target-Oriented 3D Path Planning for UAVs

Zhaoliang Zheng[1], Thomas R. Bewley[2], and Falko Kuester[3]

Abstract— **This paper explores 3D path planning for unmanned aerial vehicles (UAVs) in 3D point cloud environments. Derivative maps such as dense point clouds, mesh maps, octomaps, etc. are frequently used for path planning purposes. A target-oriented 3D path planning algorithm, directly using point clouds to compute optimized trajectories for a UAV, is presented in this article. This approach searches for obstacle-free, low computational cost, smooth, and dynamically feasible paths by analyzing a point cloud of the target environment, using a modified connect RRT-based path planning algorithm, with a k-d tree-based obstacle avoidance strategy and three-step optimization. This presented approach bypasses the common 3D map discretization, directly leveraging point cloud data. Following trajectory generation, the algorithm creates way-point based, closed-loop quadrotor controls for pitch, roll, and yaw attitude angle as well as dynamics commands for the UAV. Simulations of UAV 3D path planning based on different target points in the point cloud map are presented, showing the effectiveness and feasibility of this approach.**

Index Terms— **Point cloud maps, Target-oriented, RRT, UAVs, 3D path planning, Trajectory optimization**

I. INTRODUCTION

In the last decades, Unmanned aerial vehicles (UAVs) have been greatly developed [1]. Nowadays, UAVs, as a relatively simple but powerful, consumer-leveled and easy-to-manipulate aerial robotic system, play a more and more important role in many areas. UAVs are widely used in inspection, midair collision surveillance [2], unknown environment exploration [3], mapping and 3D modeling [4], etc. To have a better and smarter strategy of performing these tasks, 3D path planning is a necessary and key technique in all these. 3D path planning for UAVs can be defined as the process of finding an optimal and collision-free path from an initial point to the target point in the 3D (three-dimensional) workspace while avoiding all the static obstacles or other mobile agents as well as taking into account kinematic constraints [5]. It has gained popularity among researchers around the world due to the development of affordable equipment like GPS, lightweight laser and LIDAR (Light Detection And Ranging) scanners, UAVs as well as the development of new algorithms like simultaneous localization and mapping (SLAM) [6] and next-best-view [7].

A huge number of articles have proposed methods and algorithms to deal with 2D path planning. While in recent years, more and more articles extended their path planning algorithms into 3D environments for UAVs such as [8][9][10]. Most of the work in this area implement their path planning algorithms in the lab environment or 3D occupancy grid maps. However, in the outdoor real world, derivative maps such as dense point clouds, mesh maps, octomaps, etc. are frequently used for UAVs path planning purposes. Some researches have done some correlative work in this area such as [11][12][13], while others have started to implement derivative maps in SLAM [14]. As for point clouds, most of the work uses extremely dense point cloud or transform point cloud into other forms of maps after post-processing. The main shortcomings of exiting methods are: first, the computational complexity will greatly increase as the density of point clouds grow, situations will get worse in extremely dense point cloud; second, some detailed information (e.g. color and position) will be lost during the transformation from point cloud to other forms of maps.

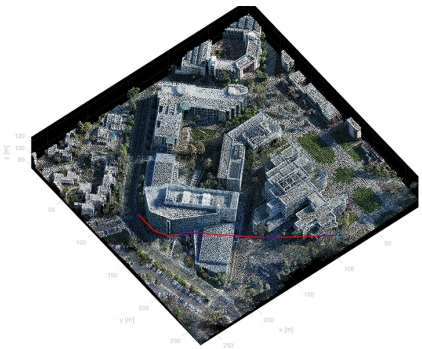

Fig. 1: The blue trajectory is generated by our path planning algorithm and the red trajectory is the real UAVs trajectory after implementing the way-point based close-loop quadrotor control.

In this paper, we propose a target-oriented 3D path planning algorithm, directly using sparse point clouds as input to compute the optimal trajectories for UAVs in outdoor environment(see fig. 1). Since the algorithm deals with sample data point cloud directly, sample-based path planning algorithm like RRT would be a good choice and more details are introduced in section III. Our algorithm analyzes a point cloud of the target environment, using a modified connect RRT-based path planning algorithm. Our approach utilize a k-d tree based obstacle avoidance strategy and three-step optimization to searching for collision-free, low computational cost, smooth, and dynamically feasible paths. Following trajectory generation, a series of way points are created along the trajectory by algorithm. By using way-

[1]Zhaoliang Zheng and [2]Thomas R. Bewley are with the Department of Mechanical and Aerospace Engineering, University of California San Diego, La Jolla, CA, 92093 USA zhz503@eng.ucsd.edu, bewley@eng.ucsd.edu

[3]Falko Kuester is with the Department of Structural Engineering, University of California San Diego, La Jolla, CA, 92093 USA fkuester@ucsd.edu

point planning, the closed-loop quadrotor control algorithm will give out desired velocity command $\dot{x}(t)$, $\dot{y}(t)$, $\dot{z}(t)$, rotational rate about the vehicle body ω_x, ω_y, ω_z and desired acceleration for UAVs dynamics. The video of fig. 1 is available at: https://youtu.be/TwDDlZ-Sr4M

The remainder of this paper is arranged as follows: in section II, mainly introduce point cloud application and generation; in section III, mainly build up the point cloud based target-oriented algorithm and technical details; in section IV, mainly derive the UAV dynamics and way-point based control method; in section V, simulations of UAV 3D path planning based on different target points in the point cloud maps are presented to show effectiveness and feasibility of our approach.

II. Point Cloud Map Representation

Maps (or environment model) are resources that enable robots to better perform their tasks [15]. Most robot maps are very important and used mainly for robots to proceed localization and navigation [16]. Many robot maps representation were proposed in the past decades, for examples: gird maps, polygonal maps, occupancy maps, counter maps, 3D mesh maps, octomap, point cloud maps. In [16], a map data representation of environment of a mobile robot performing a navigation task is specified. Nowadays, as the development of AI, computer vision, robotics and more powerful embedded systems, the functionality of the robot is not limited to positioning and navigation. The robot is also a mobile platform for real-time localization and mapping, target tracking, pattern recognition and the digital reconstruction of cultural heritage. Thus, a more informative and interactive 3D map representation is needed, which is, point clouds.

Fig. 2: Point clouds representing real-world maps (courtesy of Vid and Eric at Drone Lab@UCSD)

Point clouds is a set of data points in space which contains color and position information of the real world. It is relative simple but very powerful and vivid way to represent the real world as shown in Fig. 2. In Fig. 2, the top left is the scenery around Jacumba hot springs, CA; the top right is the Jacob school of engineering, UCSD; the bottom left is the Yosemite national park; the bottom right is one hospital somewhere in Columbia. In the following sections, point cloud generation and applications will be introduced.

A. Point Cloud Generation

Point cloud generally produced by scanner systems, which measures a large number of data points on the external surfaces of objects around them. Many researches have proposed different methods and algorithms to create point cloud models, for example: stereo cameras [17], LIDAR systems [18], monocular cameras [19], etc.

Nowadays, As development of UAV and high-resolution bui-in cameras, high-resolution point large scale cloud map become possible. More and more researches leverage UAVs and the build-in cameras as a platform to create large scale point cloud maps such as in [20][21]. The point clouds in Fig. 2 are also created by using the aerial images from UAVs.

B. Point Cloud Applications

Recent years, the point clouds are widely used in many aspects. In geographic information systems, point clouds are one of the sources used to make digital elevation models of the terrain [22]. It can also be used in feature extraction [17], contextual classifications [23], autonomous navigation [24], etc. In the Drone LAB@UCSD, we used point clouds as a visualizing way to 3D re-construct cultural heritages [25]. Because of the characteristics and practical significance of point cloud data, it is meaningful to use point cloud maps for path planning purposes.

III. Target-Oriented 3D RRT Algorithm

RRT algorithm was first proposed in [26] as a sampling way of solving the high-dimensional path planning problems. It has been proven to be probabilistically completed and computationally efficient [27]. First, RRT algorithm will randomly sample in obstacle-free space and look for a nearest point to the sample point from the random tree. Second, Extend function extends from nearest point an incremental distance in the direction of the random point to get a new vertex. Third, if collision detection happens between new vertex and random point, tree growth is abandoned, otherwise, the new vertex is added to the tree. Repeating the above process until the distance between the nearest point and goal point is less than a threshold, RRT algorithm will return a feasible path. The advantage of RRT algorithm is that it does not need to model the system or geometrically divide the search space. It has a high coverage in the search space and a wide search scope, so it can explore the unknown space as much as possible. However, the defects of RRT algorithm are also obvious. For example, the algorithm is not deterministic, a narrow passage is also difficult to pass and the found path is sharp-edged, which can't be driven easily.

To improve RRT algorithm, many of its variants are proposed in the past 20 years, for example: RRT-connect [28], RRT* [29], Bidirectional RRT* [30], RRT*-Smart [31], SRRT* [32]. These variants improve the efficiency and rate of convergence, and solve the problem of asymptotic

optimization. However, for UAVs autonomous control, these pure algorithms are not enough since the dynamics constrains of vehicles are not considered. In [33] a closed-loop RRT (CL-RRT) is proposed to involve non-holonomic constraints of the four-wheel vehicle dynamics. In [34] the CL-RRT idea was improved and implemented on quadrotor UAVs. Although the UAV non-holonomic constraints and RRT-based controller design method is provided, the Simulation environment is too ideal and final path is not smooth enough and not cost optimal.

This paper proposes a point cloud-based target-oriented 3D RRT (PT-RRT) algorithm that obtains obstacle-free, smooth, dynamically-feasible trajectories in real world and in real time. In the following sections, subsection A introduces the core algorithm description and overall workflow; subsection B introduces the point cloud map analysis to determine step size before algorithm running; subsection C introduces the obstacle avoidance strategy in point cloud maps; subsection D introduces three-steps optimization to get a smooth trajectory.

A. Algorithm Design and Workflow

PT-RRT algorithm we develop is a RRT-connect based path planning algorithm. Some modifications is made in RRT-connect. In order to be target-oriented and to reduce computational cost, we modify the extension condition in "EXTEND tree function" as shown in Algorithm 1.

Algorithm 1 EXTEND(T,x)

1: FLAG $\leftarrow 0$; $i \leftarrow 0$;
2: **if** COLLISIONDETECT = NOCOLLISION **then**
3: $x_{rand} \leftarrow x$
4: **else**
5: $x_{rand} \leftarrow$ Sample(i)
6: **end if**
7: $x_{nearest} \leftarrow$ NEARESTNODE(T,x)
8: $x_{new} \leftarrow$ STEER $(x_{nearest},x)$
9: **if** COLLISIONDETECT = NOCOLLISION **then**
10: $V \leftarrow \{x_{new}\}$; $E \leftarrow \{x_{new},x_{near}\}$
11: T $\leftarrow$\{V, E\}
12: **if** $|x_{new}-x| < StepSize$ **then**
13: **return** FLAG = 1
14: **end if**
15: **end if**
16: **return** T, FLAG

Fig. 3 shows the overall workflow of our algorithm. Dotted boxes specify the differences between our algorithm and RRT-connect algorithm. Point cloud maps analysis and three-step optimization are added to our algorithm.

B. Point Cloud Map Analysis

According to the analysis in previous part, it is difficult to do path planning in a very sparse point cloud map. In the PT-RRT algorithm, given a point cloud map, analysis is necessary before our algorithm runs.

Fig. 3: PT-RRT algorithm workflow

Point cloud density is one of the most important properties. In the map analysis, point cloud density has to be maintain at an acceptable level for the application of UAV path planning. According to [35] and [36], point cloud density δ is classified as in TABLE I. In the outdoor space, UAV 3D path planning can successfully run in sparse point cloud maps where its $0.5 < \delta < 1$, and lower than is this level, it may be problematic. Jacob school of engineering of UCSD will be used as the point cloud environment in the simulation. The δ of this model is 0.597 ± 0.02 pts/m^2.

TABLE I: Point Cloud Density Classification

Density level	δ (pts/m^2)	Applications
Sparse point clouds	$[0.5,1)$	Normally collected for large scale digital height models.
Low density point clouds	$[1,2)$	For flood modelling applications.
Medium density point clouds	$[2,5)$	Suitable for satisfying most usages.
High density point clouds	$[5,10)$	For capturing the details of buildings.
Extremely dense point clouds	$[10,\infty)$	For capturing more and all the details.

The point cloud analysis algorithm to calculate δ is shown in Algorithm 2. To reduce computational complexity, algorithm 3 randomly selects 1% sample of the whole point cloud data as the sub data to analyze maps, which is in Line 1. In Line 4, "Find Nearest Neighbor" (FINDNN) function

returns back the Euclidean distance between the K nearest neighbor points to the i^{th} subdata. In Line 6, "Find Nearest Neighbor in Radius" (FINDNR) function returns back the index of the nearest neighbor to the i^{th} subdata in radius. The principle of FINDNN and FINDNR will be discussed in detail in next section C. In Line 9, "Surface Density" (SURFDENSITY) function returns the point cloud density based on average number of index $index_{number}$ and radius. In Line 13, "Statistical Analysis" (SA) function returns back the step size for target-oriented RRT algorithm. And to ensure safety, 120% offset is added to StepSize. In Line 14, "Configuration Space Analysis" (CSA) function returns back the configuration space CSpace for UAV, which includes the x,y,z limitation of the UAV flight space.

Algorithm 2 POINTCLOUDANALYSIS($ptClMap$)

1: $subdata \leftarrow$ 1%RANDOMSAMPLE($ptClMap$);
2: $K \leftarrow \{NeighborNumber\}$; $r \leftarrow \{radius\}$;
3: **for** i =1...$subdata.Size()$ **do**
4: $dists \leftarrow$ FINDNN($subdata(i)$,$ptClMap$,K)
5: $dists_{min}(i) \leftarrow$ MIN($dists$)
6: $index \leftarrow$ FINDNR($subdata(i)$,$ptClMap$,r)
7: $index_{number}(i) \leftarrow index.Size()$
8: **end for**
9: $ptClDensity \leftarrow$SURFDENSITY($index_{number}, r$)
10: **if** $ptClDensity \leq LowerBound$ **then**
11: **return** FAIL
12: **else**
13: $StepSize \leftarrow$ SA($dists_{min}$)
14: $CSpace \leftarrow$ CSA($ptClMap$)
15: **return** $StepSize$,$CSpace$
16: **end if**

C. Obstacle Avoidance Strategy

In path planning, designing a feasible path is the first priority task. The maps environment that most path planning algorithms are dealing with are all full information geometrical maps like grids maps, occupancy maps, 3D mesh maps and so on. Under the condition that assuming all the geometry information is known, it is easier to apply obstacle avoidance strategy.

In a set of sampling data points like point cloud, things will become more complicated. The challenges of path planning in point cloud maps are obvious as discussed in previous point cloud chapter 2. Researcher normally will not deal with point cloud environment directly. A tensor voting framework is proposed in [11] to transform point cloud into a promising outcome for application of 3D path planning. Ortherwise, a direct way to do global path planning on point cloud maps is using an extremely dense point clouds as in [13]. According to the test dataset parameters table in [13], its point cloud density is $\geq$ 970 pts/m^2. Such point cloud is not just computationally expensive, but also too difficult for SLAM to realize under the current technology. Thus, a novel collision avoidance strategy is proposed in this paper.

Algorithm 3 COLLISIONDETECT($x_{curr}, x_{next}, ptClMap$)

1: $Status \leftarrow$ NOCOLLISION;
2: **if** OUTOFRANGE($x_{curr}, ptClMap$) = TRUE **then**
3: **return** $Status \leftarrow$ COLLISION
4: **end if**
5: $x_{mid} \leftarrow$ INTERP (x_{curr}, x_{next})
6: **if** CHECKPOINT($x_{mid}, ptClMap$)=HIT **then**
7: **return** $Status \leftarrow$ COLLISION
8: **end if**
9: **return** $Status$
10: **function** CHECKPOINT($x_{mid}, ptClMap$)
11: $SafeDist \leftarrow \{S\}$;
12: **for** i = 1...K **do**
13: **if** FINDNN($x_{mid}, ptClMap, i$) $\leq S$ **then**
14: **return** HIT
15: **end if**
16: **end for**
17: **end function**

The collision detection algorithm is shown in Algorithm 3. In Line 2, "out of range" (OUTOFRANGE) function checks whether current point x is outside the map. In the CHECKPOINT function, a safety distance S is specified to check if the current point is potentially about to hit the obstacle. In the algorithm, the value of S is based on $StepSize$ we got from Algorithm 2. A reference value is given in equation 1.

$$S = (0.5 \text{ or } 0.8) \times StepSize \tag{1}$$

In the left subfigure of Fig. 4, node 3 actually "sees" the goal point and try to expand a random new node on that direction (gray dashed line). The algorithm will calculate distances between the nearest point cloud neighbors to that node and check if it is in the ball-shape safe region, which is defined by S. Eventually, the node on dashed line is discarded and node 4 is added to the tree. In the right subfigure of Fig. 4, same rules are applied and since CHECKPOINT function returns "no hit" between node 5 and the goal point, node 6 is added to the tree on that direction. In

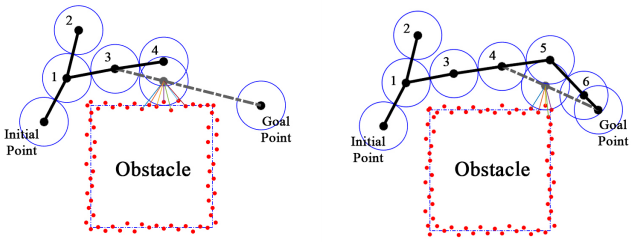

Fig. 4: Obstacle avoidance strategy 2D presentation

the CHECKPOINT function, FINDNN function is the key to determine if current point potentially hits the obstacle. FINDNN function and FINDNR function (in the previous section B) can be regarded as feature point matching query. Normally, there are two types of feature matching operators:

one is the linear scanning method, which compares the distance between the points in the dataset and the query points one by one, that is also known as exhaustive; another is to index the data and then match it quickly. One kind of representative index tree for the second method is kd-Trees and both FINDNN function and FINDNR function are all based on kd-Trees search algorithm.

Kd-Trees is a binary data structure inveted in 1970s by Jon Bentley [37]. Kd-Trees algorithm can separate into two parts[38]: first part is the algorithm of constructing the kd-tree; second part is algorithm of searching for the nearest neighbor. According to the complexity analysis in [38], the time complexity of kd-trees nearest neighbor algorithm is $O(n^{1-1/d}+k)$, where d is the d-dimension and k is the k nearest neighbors need to retrieve. It is far more efficient than exhaustive $O(n^d)$ and more flexible than octrees.

1) down sample optimization: Down sample starts from the initial point, and keeps checking the collision status between next vertex and previous break point along the connected tree. In Fig. 5, the orange trajectory is the outcome after down sample optimization. The down sample algorithm is shown in Algorithm 4.

Algorithm 4 DOWNSAMPLE(*Path*)

$Path_{break} \leftarrow \{Path(1)\}$;
for i =2...*Path.size* **do**
  $dis(i) \leftarrow$ EURDIST($Path_{break}$,$Path(i)$);
  if $dis(i) > dis(i-1)$ OR $dis(i) > 0$ **then**
    if COLLISIONDETECT = COLLISION **then**
      $indx_{new} \leftarrow \{i-1\}$
      $Path_{break} \leftarrow \{Path(i-1)\}$
    end if
  end if
end for
$Path \leftarrow Path.indx_{new}$
return *Path*

2) Up Sample Optimization: After down sample, compared with connect-tree path, the trajectory we get is locally shorter but it is not short enough since the down sample trajectory is fully based on connected tree, which is not close enough to obstacles especially at the corner. Up sample generates more sample points that are closed to the nearest obstacle and its safe boundary compared with down sample trajectory which will further shorten the trajectory. In Fig. 5, the green line is the outcome after up sample optimization. The up sample algorithm is shown in Algorithm 5.

3) B-Spline Curve Optimization: To make sure that the final trajectory generated by PT-RRT algorithm is smoother, this paper uses cubic B-Spline Curves to smooth the final trajectory. In [39] properties of B-spline curves are described to show the advantages of B-spline curves to smooth trajectories in path planning compared with Bezier curve. According to lecture [40], a B-spline curve P(t) is defined in equation 2:

$$P(\mathrm{t}) = \sum_{i=0}^{n} P_i N_{i,k}(t) \tag{2}$$

where,

Algorithm 5 UPSAMPLE(*Path*)

$dist \leftarrow \{0\}$; $iter \leftarrow 1$;
for k = 2...*Path.Size*() **do**
  $dist(k) \leftarrow$ EURDIST($Path_k, Path_{k-1}$)
  $dist(k) \leftarrow \{dist_k + dist_{k-1}\}$
end for
while $iter \leq iter_{max}$ **do**
  $R_1 \leftarrow$ RANDOMSAMPLE($dist_{end}$)
  $R_2 \leftarrow$ RANDOMSAMPLE($dist_{end}$)
  if $S_2 \leq S_1$ **then**
    SWAP(R_1, R_2)
  end if
  for k = 2...*Path.Size*() **do**
    determine index i:=$k-1$ if $dist(k) > R_1$
  end for
  for k = $(i+1)$...*Path.Size*() **do**
    determine index j:=$k-1$ if $dist(k) > R_2$
  end for
  if $j \leq i$ **then** jump to the next iteration
  end if
  $\alpha \leftarrow$ INTEPL($dist_i, dist_{i+1}, Path_i, Path_{i+1}$)
  $\beta \leftarrow$ INTEPL($dist_j, dist_{j+1}, Path_j, Path_{j+1}$)
  Pairwise CollisionDetect in $\{Path_i$,α,β,$Path_{j+1}\}$
  if ANY(Line 22) = COLLISION **then**
    jump to the next iteration
  end if
  $Path \leftarrow \{Path_i, \alpha, \beta, Path_{j+1}\}$
  $iter \leftarrow \{iter+1\}$
end while
return *Path*,ψ

- The control points $P_i = \{1,2,3...n\}$ are given by up sample optimization algorithm.
- k is the order of the polynomial segments of the B-spline curve, and in our case $k = 3$.
- The $N_{i,k}(t)$ are the "normalized B-spline blending functions". They are described by k and a non-descending sequence of breaking points $t_i = \{t_0, ..., t_{n+k}\}$.
  The $N_{i,k}(t)$ are described in equation 3, and starts with equation 4.

$$\mathrm{N}_{i,k}(t) = \frac{t-t_i}{t_{i+k-1}-t_i} N_{i,k}(t) + \frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} N_{i+1,k-1}(t) \tag{3}$$

$$N_{i,k}(t) = \{^{1,if\, t_i \leq t \leq t_{i+1}}_{0,otherwise} \tag{4}$$

Since using third-degree B-spline curves, it should be noted that the generated trajectory is 2^{nd}-order continuous and it is continuous in acceleration. Compared with Bezier curves, B-spline curves are more flexible: by adding controlling points locally to the line segment, we can locally optimize the final trajectory. After finishing fitting, the collision detection is needed for the generated trajectory. When a trajectory segment collides with an obstacle, the nearest control point to the collision point needs to be located and refitted after interpolation. The midpoint between the located control point and the previous control point is selected as

the interpolation point. Collision detection is carried out after each fitting until an obstacle-free safety trajectory is generated. In the Fig. 5, the smooth blue curve line is the final

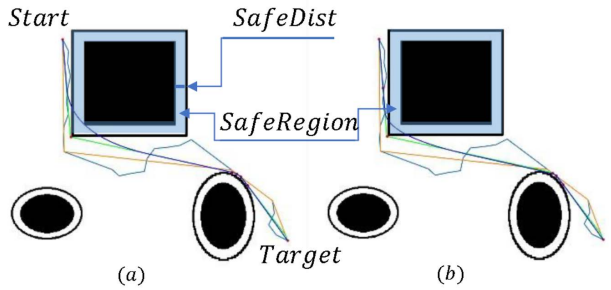

Fig. 5: Trajectory optimization and interpolation process

trajectory after B-spline optimization. Subplot (a) shows that the B-spline optimal trajectory may be able to hit the obstacle and subplot (b) shows that the outcome after interpolation process. After one red control point is interpolated into the trajectory segment, the re-fitted B-spline optimal trajectory could avoid bumping into the obstacle. The safe region is a virtual space around the obstacles where RRT sample points cannot appear in, and it is defined by S.

IV. Way-Point Based Closed-Loop Quadrotor Control

According to the method in [41], a quadrotor system has the differential flatness property, makes it possible to reduce optimal trajectories to a sequence of 3-D position and yaw angle and their derivatives. And also, as presented by Hehn et al. [42], trajectory feasibility constraints includes vehicle dynamics and input constraints, in which control inputs can be calculated from the generated trajectory. In this section, we adapt ideas from these two papers and design the way-point based closed-loop control.

A. Way-Point Based Closed-Loop Control

The way-point based closed-loop control design is shown in Fig. 6. The control input here is the desired trajectory state $Tra_{state} \in \mathbb{R}^{13}$, which is given in equation 5. The output state from quadrotor dynamics is the quadrotor state $x_{state} \in \mathbb{R}^{13}$, given in equation 6. In order to make the dimensions consistent, both Tra_{state} and x_{state} have to transform into a middle quadrotor state $Q_{state} \in \mathbb{R}^{12}$ before feeding in the quadrotor controller, given in equation 7.

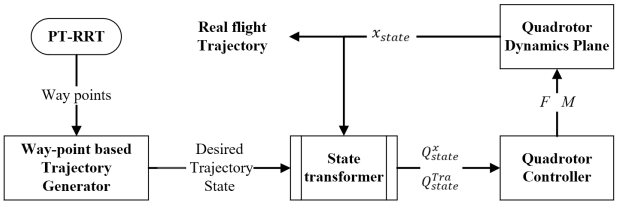

Fig. 6: Way-point based closed-loop control block diagram

$$Tra_{state} = \begin{bmatrix} p_{xyz} & \nu_{xyz} & a_{xyz} & el_{xyz} & \dot{\psi} \end{bmatrix}^T \in \mathbb{R}^{13} \quad (5)$$

$$x_{state} = \begin{bmatrix} p_{xyz} & \nu_{xyz} & q_{wxyz} & \omega_{pqr} \end{bmatrix}^T \in \mathbb{R}^{13} \quad (6)$$

$$Q_{state} = \begin{bmatrix} p_{xyz} & \nu_{xyz} & el_{xyz} & \omega_{pqr} \end{bmatrix}^T \in \mathbb{R}^{12} \quad (7)$$

where,

$p_{xyz} = \begin{bmatrix} x & y & z \end{bmatrix}$, is the position in world coordinates;
$\nu_{xyz} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}$, is the velocity in world coordinates;
$a_{xyz} = \begin{bmatrix} \ddot{x} & \ddot{y} & \ddot{z} \end{bmatrix}$, is the acceleration in world coordinates;
$el_{xyz} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}$, is the Euler roll, pitch and yaw;
$\omega_{xyz} = \begin{bmatrix} p & q & r \end{bmatrix}$, is the angular velocity around body xyz-axis;
$q_{wxyz} = \begin{bmatrix} q_w & q_x & q_y & q_r \end{bmatrix}$, is the quaternion.

The state transformer will transform quaternion q_{wxyz} into Euler angle el_{syz}, and transform Tra_{state} into quadrotor state. The whole control system has to be initialized every time before system runs. The trajectory generator will first give out an initial trajectory state Tra_{state} and quadrotor state x_{state} will be assigned by Tra_{state} in the initialization.

B. Quadrotor Controller for Position Control

The inputs of controller are desired quadrotor middle state Q^{des}_{state} and current quadrotor middle state Q_{state}. The desired middle state is Q^{des}_{state} calculated based on desired trajectory state Tra_{state}. The current quadrotor middle state Q_{state} is calculated from the quadrotor current state x_{state} The controller is given in equation 8, 9, 10, 11.

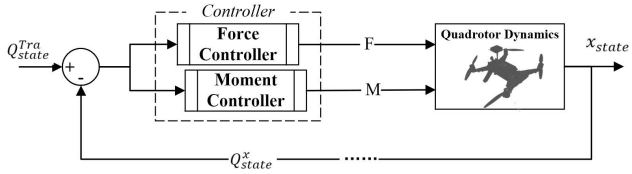

Fig. 7: Controller diagram for position control

$$a_{xyz} = a^{des}_{xyz} + K_\nu(\nu^{des}_{xyz} - \nu_{xyz}) + K_p(p^{des}_{vxyz} - p_{xyz}) \quad (8)$$

$$\begin{cases} \phi &= \frac{1}{g}(a_x sin\psi^{des} - a_y cos\psi^{des}) \\ \theta &= \frac{1}{g}(a_x cos\psi^{des} - a_y sin\psi^{des}) \\ \psi &= \psi^{des} \end{cases} \quad (9)$$

$$F = (mg + ma)R\bar{i}_z \quad (10)$$

$$M = I_{xyz}(K_{vm}(\omega^{des}_{xyz} - \omega_{xyz}) + K_{pm}(el^{des} - el)) \quad (11)$$

The outputs of quadrotor controller are F, M and x_{state}. These outputs will be feeding into the quadrotor dynamics plane to solve for the state space equations.

V. Simulation and Result Discussion

In this paper simulation tests for three different targets are involved. Three simulations are all based on a real world point cloud map, which is the Jacob School of Engineering at UCSD. The parameters of point cloud dataset are given in TABLE II. The dataset is provided by Drone Lab@UCSD.

TABLE II: Test Dataset Parameters

Dataset	**Jacob School of Engineering@UCSD**
3D model .ply format size	10.8mb
Number of points	405934
Point cloud density	$0.597 \pm 0.01 pts/m^2$
Actual size	$(270 \times 270 \times 52)m$

According to point cloud analysis, the histogram of the distance and number of the nearest neighboring points of the point cloud is shown in figure 8. The SA function gives out the step size for PT-RRT algorithm: $StepSize = 5m$, and safety distance: $SafeDist = 3m$.

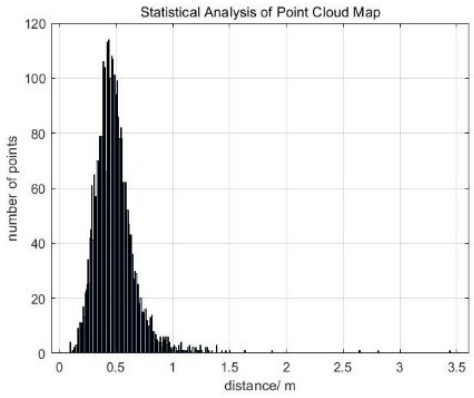

Fig. 8: Statistical analysis of test dataset

The simulation is running in MATLAB software, on a personal laptop with i5-9300H 2.4GHz processor, 8G memory. Simulation parameters settings are shown in TABLE III. We design three simulation tasks based on different initial and target points in order to show target orientation, three-step optimization and interpolation process of our algorithm.

TABLE III: Simulation Setting

No.	**Description**		
	Initial point	**Goal point**	**Verification**
1	(230,120,95)	(150,25,90)	Target orientation
2	(230,120,95)	(90,260,95)	Up-Sample Optimization
3	(158,150,96)	(175,185,100)	B-Spline Optimization

In the Fig. 9, Fig. 10, and Fig. 11, blue and red star dots are the vertexes in tree 1(starts from initial point) and tree 2(starts from target point); the red polyline is connected tree 1 and 2; the yellow line is the down sample trajectory, the green line is the up sample trajectory, and the blue curve is the final trajectory. In Fig. 11, the top right and left pictures show the comparison between no interpolation process and after interpolation process.

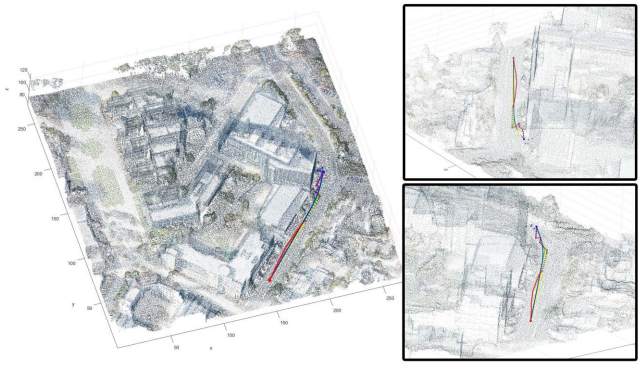

Fig. 9: test1, PT-RRT run time: 10.55s

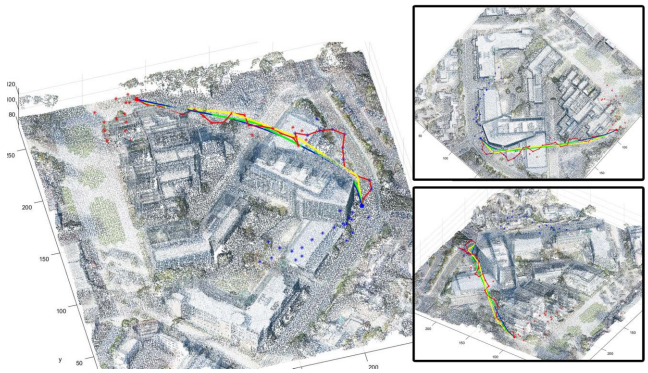

Fig. 10: test2, PT-RRT run time: 14.30s

Fig. 11: test3, PT-RRT run time: 9.88s

Fig. 12, Fig. 13, Fig. 14 show way-point close-loop quadrotor flight control simulation running in test 1, test 2, test 3, the blue line is trajectory generated by PT-RRT algorithm, the red line is the real quadrotor flight trajectory with the respect of quadrotor dynamics constrains. This simulation shows that the effectiveness and feasibility of our algorithm.

However, there indeed in some cases that proposed 3D path planning algorithm may fail. For instance, if the point cloud map we used is incomplete, in another words, if the point cloud map couldn't capture and whole feature surface of the obstacle and there exists cavities on the surface, then the trajectory generated by the 3D path planning algorithm may enter the obstacle. So in the future, combining my algorithm and SLAM will be a promising way of solving this problem.

Fig. 12: UAV flight control simulation for test 1, flight run time t=24.5s

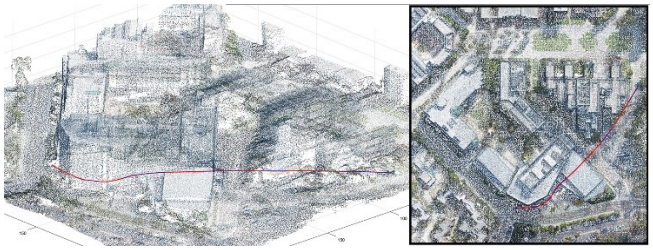

Fig. 13: UAV flight control simulation for test 2, flight run time t=39.6s

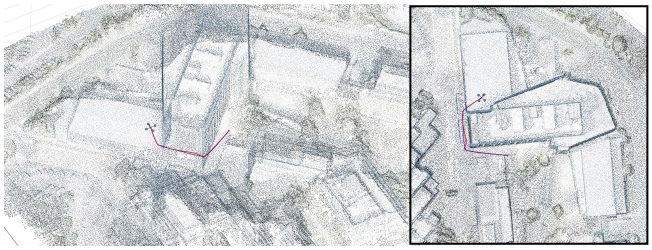

Fig. 14: UAV flight control simulation for test 3, flight run time t=12.4s

VI. Conclusion and Future Work

This paper proposes a RRT based 3D path planning algorithm and a way-point based closed-loop quadrotor control for UAVs in 3D point cloud environment. The trajectory generated by our algorithm is obstacle free, smooth, target-oriented, computationally low-cost, and dynamically feasible for UAVs. The closed-loop quadrotor control treats 3-D position and yaw angle and their derivatives as control inputs, satisfying the dynamics constraints while following the way points. The simulations for different target points show the effectiveness and feasibility of our algorithm and control method.

For future work, the author plans to run experimental testing using real quadrotor with GPS, cameras, IMU and altimeter. In the outdoor space, more noise will be involved. We will check how good is our trajectory planning in real world and how robust is our control system will be.

ACKNOWLEDGMENT

This work is supported by the Dron Lab@UCSD and the Coordinated Robotics Lab@UCSD. Zhaoliang would like to thank his lab mates Vid Petrovic and Eric Lo for their ideas and contributions in the point cloud maps.

References

[1] J. F. Keane and S. S. Carr, "A brief history of early unmanned aircraft," Johns Hopkins APL Tech. Dig. Applied Phys. Lab., vol. 32, no. 3, pp. 558-571, 2013.

[2] C. Fu, A. Carrio, M. Olivares-Mendez, R. Suarez-Fernandez, and P. Cam- poy, "Robust real-time vision-based aircraft tracking from Unmanned Aerial Vehicles," in Robotics and Automation (ICRA), 2014 IEEE International Conference on, 2014, pp. 5441-5446

[3] C. Fu, A. Carrio, and P. Campoy, "Efficient visual odometry and mapping for Unmanned Aerial Vehicle using ARM-based stereo vision pre-processing system," in Unmanned Aircraft Systems (ICUAS), 2015 International Conference on, 2015, pp. 957-962.

[4] F. Remondino, L. Barazzetti, F. Nex, M. Scaioni, and D. Sarazzi, "UAV PHOTOGRAMMETRY FOR MAPPING AND 3D MODELING," vol. XXXVIII, no. September, pp. 14-16, 2011.

[5] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, and Y. Xia, "Survey of Robot 3D Path Planning Algorithms," J. Control Sci. Eng., vol. 2016, 2016.

[6] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," in IEEE Robotics and Automation Magazine, vol. 13, no. 2, pp. 99-110, June 2006.

[7] V. Zielke, "Design and implementation of Next-Best-View algorithms for automatic robotic-based (dis) assembly tasks," Abschlussarbeit (Diplom), 2016.

[8] S. Liu, N. Atanasov, K. Mohta and V. Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control," 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, 2017, pp. 2872-2879.

[9] S. Liu, K. Mohta, N. Atanasov and V. Kumar, "Search-Based Motion Planning for Aggressive Flight in SE(3)," in IEEE Robotics and Automation Letters, vol. 3, no. 3, pp. 2439-2446, July 2018.

[10] H. Dai and Q. Liu, "EB-RRT * based Navigation Algorithm for UAV Dai Hongliang , Liu Qinglin," in 2017 3rd International Forum on Energy, Environment Science and Materials (IFEESM 2017), vol. 120, no. Ifeesm 2017, pp. 1313-1329.

[11] F. Gao and S. Shen, "Online quadrotor trajectory generation and autonomous navigation on point clouds," SSRR 2016 - Int. Symp. Safety, Secur. Rescue Robot., pp. 139-146, 2016.

[12] M. Liu, "Robotic online path planning on point cloud," IEEE Trans. Cybern., vol. 46, no. 5, pp. 1217-1228, 2016.

[13] R. Fedorenko, A. Gabdullin, and A. Fedorenko, "Global UGV Path Planning on Point Cloud Maps Created by UAV - IEEE Conference Publication," 2018 3rd IEEE Int. Conf. Intell. Transp. Eng., pp. 253-258, 2018.

[14] Fioraio, Nicola et al. "Realtime Visual and Point Cloud SLAM Nicola Fioraio Willow Garage." (2011).

[15] R. B. Rusu, N. Blodow, Z. Marton, A. Soos, and M. Beetz, "Towards 3D object maps for autonomous household robots," IEEE Int. Conf. Intell. Robot. Syst., pp. 3191-3198, 2007.

[16] S. Vasudevan, S. Gächter, V. Nguyen, and R. Siegwart, "Cognitive maps for mobile robots-an object based approach," Rob. Auton. Syst., vol. 55, no. 5, pp. 359-371, 2007.

[17] Sukhan Lee, Daesik Jang, Eunyoung Kim, Suyeon Hong and JungHyun Han, "A real-time 3D workspace modeling with stereo camera," 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton, Alta., 2005, pp. 2140-2147.

[18] B. Yunfei et al., "Classification of Lidar Point Cloud and Generation of Dtm," Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci., pp. 313-318, 2002.

[19] R. A. Newcombe and A. J. Davison, "Live dense reconstruction with a single moving camera," 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Francisco, CA, 2010, pp. 1498-1505.

[20] T. Rosnell and E. Honkavaara, "Point cloud generation from aerial image data acquired by a quadrocopter type micro unmanned aerial vehicle and a digital still camera," Sensors, vol. 12, no. 1, pp. 453-480, 2012.

[21] B. O. Abayowa, A. Yilmaz, and R. C. Hardie, "Automatic registration of optical aerial imagery to a LiDAR point cloud for generation of city models," ISPRS J. Photogramm. Remote Sens., vol. 106, pp. 68-81, Aug. 2015.

[22] A. K. Pankaj, A. Lars, and D. Andrew. "From Point Cloud to Grid DEM: A Scalable Approach." 2006.

[23] M. Weinmann, A. Schmidt, C. Mallet, S. Hinz, F. Rottensteiner, and B. Jutzi, "Contextual classification of point cloud data by exploiting individual 3D neigbourhoods," ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci., vol. 2, no. 3W4, pp. 271-278, 2015.

[24] M. Whitty, S. Cossell, and K. Dang, "Autonomous navigation using a real-time 3D point cloud," Australas. Conf. Robot. Autom., pp. 1-10, 2010.

[25] V. Petrovic, D. J. Vanoni, A. M. Richter, T. E. Levy, and F. Kuester, "Visualizing high resolution three-dimensional and two-dimensional data of cultural heritage sites," Mediterr. Archaeol. Archaeom., vol. 14, no. 4, pp. 93-100, 2014.

[26] Steven M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," TR 98-11, Computer Science Dept, Iowa State University, 1998.

[27] Steven M. LaValle and James J. Kuffner, "Rapidly-Exploring Random Trees: Progress and Prospects," Algoritlunic and. Comput. Robot. New Dir., 2000.

[28] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," no. Icra, pp. 995-1001, 2002.

[29] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," Int. J. Rob. Res., vol. 30, no. 7, pp. 846-894, 2011.

[30] M. Jordan and A. Perez, "Optimal Bidirectional Rapidly-Exploring Random Trees," Comput. Sci. Artif. Intell. Lab., 2013.

[31] I. Noreen, A. Khan, and Z. Habib, "A Comparison of RRT, RRT* and RRT*-Smart Path Planning Algorithms," IJCSNS Int. J. Comput. Sci. Netw. Secur., vol. 16, no. 10, pp. 20-27, 2016.

[32] R. Hess, R. Jerg, T. Lindeholz, D. Eck, and K. Schilling, "SRRT* - a probabilistic optimal trajectory planner for problematic area structures," IFAC-PapersOnLine, vol. 49, no. 30, pp. 331-336, 2016.

[33] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," IEEE Transactions on Control Systems Technology, vol. 17, no. 5, pp. 1105-1118, 2009.

[34] Y. Dong, C. Fu, and E. Kayacan, "RRT-based 3D path planning for formation landing of quadrotor UAVs," 2016 14th Int. Conf. Control. Autom. Robot. Vision, ICARCV 2016, vol. 2016, no. November, pp. 13-15, 2016.

[35] USGS, "National Geospatial Program Lidar Base Specification Lidar Base Specification Techniques and Methods 11-B4," no. August 2012, p. 114, 2018.

[36] F. Rohrbach, "Point Density and Point Spacing," Geomatics Technologies Blog, 2015 [Online]. Available: https://felix.rohrba.ch/en/2015/point-density-and-point-spacing/ [Accessed: 14- Oct- 2015]

[37] J. L. Bentley, "Multidimensional binary search trees used for associative searching," Communications of the ACM, vol. 18, no. 9, pp. 509-517, 1975.

[38] H. M. Kakde, "Range Searching using Kd Tree," Report, pp. 1-12, 2005.

[39] A. Babaei and A. Karimi, "Optimal Trajectory-Planning of UAVs via B-Splines and Disjunctive Programming," pp. 1-12, 2018.

[40] K. I. Joy, "DEFINITION OF A B-SPLINE CURVE." University of California, Davis

[41] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," Proc. - IEEE Int. Conf. Robot. Autom., pp. 2520-2525, 2011.

[42] M. Hehn and R. D'Andrea, "Quadrocopter trajectory generation and control," IFAC Proc. Vol., vol. 44, no. 1 PART 1, pp. 1485-1491, 2011.