Mechatronics 97 (2024) 103094

Contents lists available at ScienceDirect

Mechatronics

journal homepage: www.elsevier.com/locate/mechatronics

A fast planning approach for 3D short trajectory with a parallel framework☆,☆☆

Han Chen [a], Shengyang Chen [b], Chih-Yung Wen [a], Peng Lu [c,*]

[a] Department of Aeronautical and Aviation Engineering, Hong Kong Polytechnic University, Hong Kong, China
[b] Department of Mechanical Engineering, The Hong Kong Polytechnic University, Hong Kong, China
[c] Department of Mechanical Engineering, The University of Hong Kong, Hong Kong, China

ARTICLE INFO

Keywords:
Path planning
UAVs
Autonomous navigation
Unknown environment

ABSTRACT

For real applications of unmanned aerial vehicles, the capability of navigating with full autonomy in unknown environments is a crucial requirement. However, planning a shorter path with less computing time is contradictory. To address this problem, we present a framework with the map planner and point cloud planner running in parallel in this paper. The map planner determines the initial path using the improved jump point search method on the 2D map, and then it tries to optimize the path by considering a possible shorter 3D path. The point cloud planner is executed at a high frequency to generate the motion primitives. It makes the drone follow the solved path and avoid the suddenly appearing obstacles nearby. Thus, vehicles can achieve a short trajectory while reacting quickly to the intruding obstacles.

We demonstrate fully autonomous quadrotor flight tests in unknown and complex environments with static and dynamic obstacles to validate the proposed method. In simulation and hardware experiments, the proposed framework shows satisfactorily comprehensive performance.

1. Introduction

In application scenarios, such as searches and expeditions, small drones are usually used to explore unknown environments. To achieve autonomous unmanned aerial vehicle (UAV) navigation in unknown environments, achieving onboard simultaneous localization and mapping (SLAM) and path planning is required [1,2]. In the research field of path planning for UAVs, the three most essential indicators are usually safety, trajectory length, and calculation time for replanning the trajectory. In general, all methods are designed to ensure that the flight is safe, that is, collision-free. However, regarding path length and calculation time, most researchers have only focused on one of these factors because of the potential conflicts between them. In other words, calculating a shorter path is very likely more time-consuming. On the other hand, minimizing the total calculation (reaction) time often requires direct planning based on the raw sensing data instead of planning on the periodically updated map, and therefore, it is difficult to handle complex environments [3,4]. The UAV is more likely to detour, resulting in an inefficient flight trajectory.

In addition, calculating the shortest path in the global 3D map consumes too much time and is not applicable to real-time planning. While flying in an unknown environment, the environmental information sensed by the drone is continuously used to update the map [5]. After the map is updated, if the planned path cannot be replanned in time, the flight of the drone will be greatly imperiled. Therefore, for real-time calculations during drone flight, using a local map (the part of the global map around the location of the drone) is a common and effective method. Moreover, drones in unknown environments usually do not have a complete map [6], which means that the globally shortest path is difficult to plan.

Admittedly, planning the globally shortest path with only a local map is impossible, shortening the path in the local map will also contribute to shortening the final flight path length. Nevertheless, to respond to emergencies, planning on the map may not be sufficiently fast. Mapping and planning on the map cost too much time to avoid the intruding dynamic obstacles [1]. The drone must be able to avoid sudden obstacles in the unknown environment before the map is updated. Therefore, we propose a framework in which a low-frequency

☆ This paper was recommended for publication by Associate Editor YangQuan Chen.

☆☆ This work was supported in part by General Research Fund under Grant 17204222, and in part by the Seed Funding for Strategic Interdisciplinary Research Scheme, Platform Technology Fund and the project of "Resilient Urban PNT Infrastructure to Support Safety of UAV Remote Sensing in Urban Regions".

* Corresponding author.
E-mail addresses: stark.chen@connect.polyu.hk (H. Chen), shengyang.chen@connect.polyu.hk (S. Chen), chihyung.wen@polyu.edu.hk (C.-Y. Wen), lupeng@hku.hk (P. Lu).

https://doi.org/10.1016/j.mechatronics.2023.103094
Received 11 March 2022; Received in revised form 19 September 2023; Accepted 30 October 2023
Available online 18 November 2023

path planner and a high-frequency trajectory planner work in parallel. The designated goal is cast to the local map as the local goal. The map planner (MP) is first used to determine the 2D path to the local goal with the improved jump point search (JPS) method on the projection map. Then, a discrete angular graph search (DAGS) is used to find a 3D path that is obviously shorter than the 2D one. If the shorter 3D path is found, it is adopted. Otherwise, the 2D path is output. The point cloud planner (PCP) for trajectory planning is based on the design in our previous work [3]. With a given goal, the PCP generates collision-free motion primitives continuously in a computationally efficient way to navigate the drone. In this parallel framework, it calculates the goal from the path output by the MP. In addition, we introduce the calculation formula for obtaining the goal for the PCP from the waypoints in the path. One benefit of the local map is that the computational time for the path planning on the map will not increase with the global map size. The path output frequency and the computational resource usage are guaranteed in a specific range to ensure that the loop frequency of the PCP is unaffected, and the MP can respond to the map change in time. In this framework, all the submodules are designed to minimize the time cost. For UAVs' real-time planning, it is safer when the planning outer loop frequency is higher.

The main contributions of this work are as follows:

- A parallel architecture with the MP and PCP is proposed, considering the planning success rate, path length, and fast response. The framework has been tested to achieve satisfactorily synthesized performance in extensive environments.
- A sliding local map with two resolutions is introduced to increase the planning speed while maintaining a fine-grained path around the drone.
- We introduce the DAGS based on the angular cost and try to find a 3D path shorter than the improved 2D path.
- Based on our former work [3], the PCP is further optimized in time cost, motion planning success rate, and safety. To connect two planners in one framework, we build an optimization problem to calculate the local goal from the path output by the MP. The analytical solution of the optimization problem is found from a geometric view.

Our proposed framework's performance is tested and verified in several simulation and hardware tests. The flight trajectory length and the detailed algorithm execution time are compared with the shortest global path length and those of the state-of-the-art (SOTA) algorithms, respectively, demonstrating the superiority of our method with the best all-around performance.

2. Related work

2.1. Environmental information retrieving method

In the related works, two main categories for environmental information retrieving methods can be summarized: memory-less and fusion-based [7]. The first category only uses the latest sensor measurement data or weights the most recent data [8,9]. In other words, these methods will not record the passed by obstacles [10,11]. An example of this kind of method is to generate motion primitives randomly and check collision with the transformed point cloud and the trajectories [4]. However, path planning directly on the latest point cloud requires the information of high quality and in full view. For UAVs with limited sensors, e.g., only one single depth camera with a narrow field of view, such methods are not applicable.

The second type is based on data fusion. Sensor data will be continuously fused into a map, usually in the form of occupying grid or distance field [12]. Considering the map is important to plan a safe and short path in complex scenarios, many navigation frameworks are applied on a global map or local map. For building a map with the sensed environmental information, representative methods include voxel grids [13], Octomap [14], and elevation maps [15]. Octomap is memory-efficient for presenting a large-scale environment and maintaining the map automatically. In [16], with the point cloud raw data input, Octomap is utilized to provide the map for their proposed planning algorithms, and the experimental results are satisfactory.

2.2. Path planning

For path searching of UAVs, the algorithms commonly used can be classified into two categories: searching-based or sampling-based methods. Searching-based methods discretize the whole space into a grid map and solve path planning by graph searching. The graph can be defined in a 2D, 3D, or higher-order state space. Typical methods include Dijkstra [17], A* [18], anytime repairing A* [19], JPS [20], and hybrid A* [21]. Dijkstra's algorithm is the root of the above methods, which searches path by utilizing an exhaustive method on all the given grids. A* improves the efficiency by setting a cost function to cut off the search away from the goal. As an improved version of the traditional A*, JPS greatly reduces its time cost without sacrificing the optimality in all the cases. However, as the path direction is constrained, the path is not the true shortest in the unconstrained 2D map.

Sampling-based methods usually do not need to discretize the space first. In the representative sampling-based approach such as rapidly exploring random tree (RRT) [22], random and uniform sampling is performed from the space near the starting point, and the root node and child nodes are continuously connected to form a tree that grows toward the target. Sampling-based methods with asymptotic optimality include probabilistic road maps (PRM*) [23], rapid exploration of random graphs (RRG) [24] and RRT* [24], where RRT* can make the solution converge to the global best point with the increase of samples. Based on RRT, the method in [25] ensures the asymptotic optimality of the path and kinematics feasibility.

2.3. Trajectory and motion planning

The trajectory planner utilizes the local obstacle information and the target point's position to plan an optimal path and a corresponding set of motion primitives within a particular time. Path planning on a local map can be conducted before the motion planning to improve the trajectory length. For example, Chen et al. [26] and Liu et al. [27] adopt the minimum-jerk method to generate trajectory through the waypoints from an A* search based on the local occupancy grid map. Also, the similar approach is adopted in [28]. Besides, some works studied how to allocate the flight time for a drone to fly through the waypoints [29,30]. The receding horizon control policy is introduced to plan in a limited time range [31], and a bi-level optimization method is also effective [2,32]. In our previous work [3], we propose an efficient method to continuously generate collision-free motion primitives for a narrow time window by non-linear optimization. The local waypoint constrains the optimization in a collision-free direction, however, the optimization success rate and computation cost are unsatisfactory. In this work, we improve the optimization formula to reduce the problem's complexity. Thus, better performance is achieved. For tasks that no obstacle appears and the waypoints are pre-assigned, the trajectory can be directly planned considering the minimum energies or lowest difficulties in reorienting the vehicle [33].

In the last few year, several research works discussed how to combine the optimal global planning algorithm for static maps with the algorithm applied to real-time online replanning. For the existence of the unknown space in the environment, several methods can be adopted: the unknown space is regarded to be freely passable in [34,35], and the path is continuously adjusted as the obstacle information is updated. We call it the optimistic planner. In [2,36], the optimistic global planner and conservative local planner are combined to ensure the safety of the aircraft. To reduce the heavy computational burden of the global planning, a combined 2-D and higher dimensional state space can effectively finish the global and local planning in a single search scheme [37].

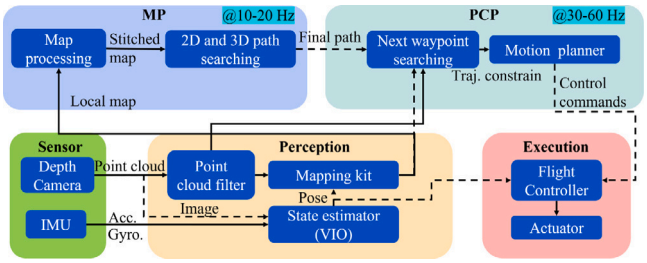


Fig. 1. Architecture of our autonomous navigation system for UAVs. A dashed arrow connects two modules executed at different frequencies, and a solid arrow for the same frequency.

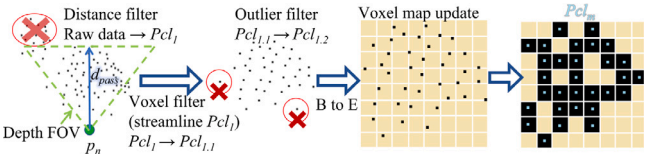


Fig. 2. The visualization of the point cloud filters' function.

3. Mapping and the map planner

The flight system architecture is shown in Fig. 1. From the onboard sensors, such as a depth camera and IMU, visual SLAM algorithms can produce the pose estimation of the drone. The mapping kit can build a 3D voxel map of the environment to store obstacles with the pose estimation and point cloud. The MP obtains the map from the mapping kit and plans the final path as the reference, and the PCP searches the next waypoint based on the final path and optimizes the motion primitives to make the drone fly through the next waypoint. Please note that MP and PCP are still parallel since they are executed at different frequencies. PCP does not have to wait for MP's result, it just takes the latest path regardless of its update. Finally, the motion commands are sent to the underlying controller, to translate to the inputs for the motors on the drone.

This section primarily introduces the construction of a stitched map with two resolutions and the algorithms used for path planning in the local map (the MP). The PCP will be introduced in Section 4. Moreover, the point cloud filter for the raw sensor data preprocessing is introduced at the beginning.

3.1. Point cloud filter

The dense raw point cloud from a real depth camera may overburden computational procedures, and the contained heavy noise will mislead the mapper to mark many nonexistent obstacles on the map. Before building a map, we filter out the noise in the point cloud and keep the actual obstacle points. We use Pcl to denote a point cloud, which is a group of 3D points. First, we filter the original point cloud data through the distance filter to obtain Pcl_1. It removes the points farther than d_{pass} from the camera, which may contain too much noise. Next, a voxel filter is used to downsample Pcl_1 to $Pcl_{1.1}$. Furthermore, the outlier filter removes the outliers to obtain $Pcl_{1.2}$: The local point density distinguishes the outliers because the point cloud density of the noise is usually smaller. Then, we convert $Pcl_{1.2}$ from body frame $B-xyz$ into Pcl_2 in the earth coordinate system $E-XYZ$, and use Pcl_2 in the mapping kit to build and maintain a global map. Finally, the center points of occupied voxels are used as the 3D map for the collision check, referred to as Pcl_m. The complete procession of the point cloud filtering is illustrated in Fig. 2. It is apparent that Pcl_m well retains the basic shape of the obstacle in a more concise and tidy form.

At last, Pcl_2 is used for the map building and collision check in the PCP, and Pcl_m is used for the 3D path collision check in the MP.

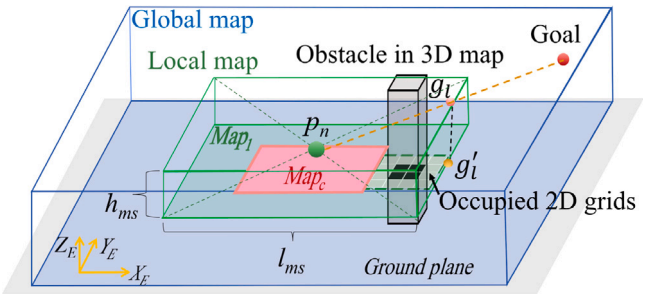


Fig. 3. Local and global maps. A rectangular obstacle stands inside the range of the local map, and we show the occupied grids and inflation in a 2D map (see Fig. 4).

3.2. Mapping and 2D path planning

The mapping kit MLmapping[1] assembled in this project is self-developed. It provides Pcl_m and the projected 2D grid map for the path planning in this paper. Here, we first introduce the basic concepts of the local map. A local map is a subset of the global map and is also presented by voxels' center points. The space covered by the local map is a cuboid with a square bottom surface, and it has no relative rotation to the global map. As shown in Fig. 3, l_{ms} is the square side length, and h_{ms} is the local map height, which is much smaller than l_{ms}. The center of the local map follows the drone's current position $p_n \in \mathbb{R}^3$. The local goal $g_l \in \mathbb{R}^3$ is assigned in a receding horizon way, while g_l' is the goal in 2D map Map_1. We use Pcl_{lm} to represent the subset of Pcl_m corresponding to the local map in the text below. Only the points at the similar height (height difference $< r_{safe}$) with the drone is projected to obtain the 2D map Map_1 for planning the 2D path. r_{safe} is the pre-assigned safety radius, and will be further introduced later.

To plan an optimal path on Map_1, JPS is one of the best choices, because it is fast and can replan the path in real-time. JPS outputs the optimal path by searching a set of jump points where the path changes its direction. However, two problems arise if the path planning is performed directly on Map_1. First, to find a short and safe path, the local map scale should not be small. Otherwise, the optimal path on a tiny local map is more likely to end at a blind alley or differs substantially from the globally optimal path. However, a large local map is computationally expensive, and it is important to leave as much CPU resource as possible to the high-frequency PCP for safety. Second, the path planned directly on Map_1 is adjacent to the obstacle projection. In our framework, considering the drone frame size and flight control inaccuracies, the drone must remain at a safe distance from obstacles. Thus, the path should remain a certain distance from obstacles. The PCP will make the drone closely follow the path obtained by the MP. When the path is found to be occupied, the PCP starts to take effect. As a result, the PCP in this framework can run faster compared to that of [3], because the initial search direction is more likely to be collision-free.

In our framework, we take two measurements to address these two problems. For the first problem, we plan the path on the downsampled local map and the cast local map, respectively, and fuse the paths as shown in Fig. 4. The dark gray grids indicate the obstacle, and the light gray grids are the obstacle's inflation after the convolution. The path planning start point is the center of Map_1 and Map_c. We first conduct the convolution with Map_1 to reduce the map size and obtain a low-resolution version Map_{1b} from Map_1 (Fig. 4, top right). Map_c is segmented from Map_1 afterward as the original resolution map around the drone (Fig. 4, bottom left). Then, we plan $Path_b$ on Map_{1b} and find the intersection point g_{ist} of $Path_b$ and the Map_c boundary (Fig. 4, top

[1] https://github.com/HKPolyU-UAV/MLMapping.

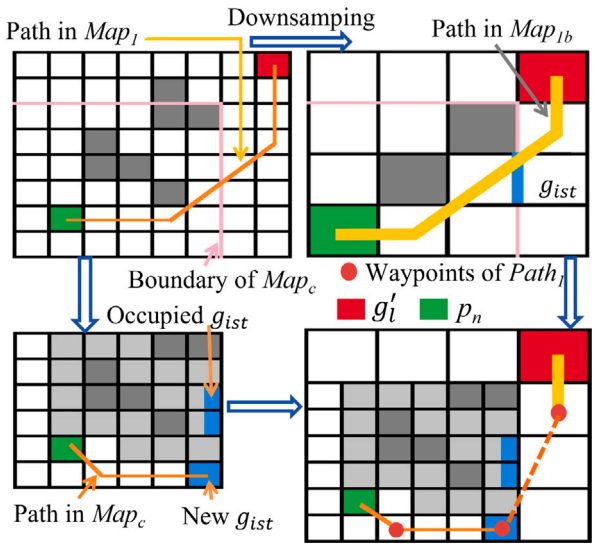


Fig. 4. The map downsampling and obstacle inflating (k = 3, h = 2), and the path planning in the stitched map. In the bottom right figure, $Path_a$ is shown by the orange thin solid line, and $Path_b$ is the yellow thick solid line.

right). The part of $Path_b$ that lies in Map_c is removed. Finally, we use g_{ist} as the goal point to find the path $Path_a$ in Map_c, and splice $Path_a$ and $Path_b$ to form a complete path $Path_1 = \{jp_1, jp_2, \dots\}$ (Fig. 4, bottom right). The grid size of Map_{1b} positively correlates with the map size, so the time cost of 2D path planning can be controlled.

For the second problem, after we have obtained Map_c, we first perform an expansion operation on the obstacles in Map_c. Using a convolution kernel to convolve the binary matrix corresponding to Map_c, the blank area next to the obstacle in the map can be marked as an obstacle so that each point on $Path_1$ maintains a certain distance from true obstacles.

$$Map_1' = \begin{bmatrix} [Map_1]_{i\times j} & [\mathbf{0}]_{(i+s)\times s} \\ [\mathbf{0}]_{s\times j} & \end{bmatrix} \tag{1}$$

$$Map_c' = \begin{bmatrix} & [\mathbf{0}]_{k\times(n+2k)} & \\ [\mathbf{0}]_{(m+2k)\times k} & [Map_c]_{m\times n} & [\mathbf{0}]_{(m+2k)\times k} \\ & [\mathbf{0}]_{k\times(n+2k)} & \end{bmatrix} \tag{2}$$

$$Map_c = Sgn(Conv_1([Map_c']_{(m+2k)\times(n+2k)}, I_{k\times k})) \tag{3}$$

$$Map_{1b} = \langle Conv_h([Map_1']_{(i+s)\times(j+s)}, \frac{I_{h\times h}}{h^2})\rangle \tag{4}$$

$$h = \langle \frac{ij}{2mn} \rangle \ (ij > 3mn \ \&\ (i+s) \text{ is divisible by } h) \tag{5}$$

Eqs. (1)–(4) show the calculation of the downsampling and inflation. i and j denote the size of the original Map_1 ($i = j$), and m and n denote the size of the cut map Map_c ($m = n$). We use [ ] to present a matrix, and the subscript of the matrix denotes its size. $[\mathbf{0}]$ indicates the zero matrix. s is the line and column number for zero padding for Map_1. h and k are the convolution kernels' size for map downsampling and obstacle inflation, respectively. $Sgn()$ is a function that returns the sign matrix corresponding to each element in the input matrix. The sign matrix is used as the binary map with two types of elements: 0 and 1. $Conv()$ indicates the convolution, and it inflates the occupancy grids on the map or downsamples Map_1. Its subscription indicates the step size for the kernel sliding, and the second element is the convolution kernel. $\langle\rangle$ is for rounding the number to the nearest integer. If a matrix is in $\langle\rangle$, it rounds each element in the matrix. (3) represents the obstacle inflation process, and (4) is for the map downsampling. Fig. 4 illustrates the map processing and path planning intuitively. The deep gray grids represent the obstacles, and the light gray grids are the inflation of obstacles after the convolution. g_{ist} is represented in blue on the maps. When g_{ist} is occupied after the inflation, we find the nearest free grid on the map edge as the new g_{ist}. The calculation of h is introduced in (5), and i, j, m, n, s should meet the conditions in the bracket.

3.3. Improved 2D path

In the last subsection, a path $Path_1(jp_1, jp_2, \dots)$ on a plane parallel to the ground plane XY is found using the JPS method in the hybrid map of Map_{1b} and Map_c. However, in some cases, it is not the shortest path in the plane, as search directions of waypoints can only be a multiple of 45°. We can further optimize the original path by deleting the redundant waypoints. For example, in Fig. 5, the red path is the original path, the green path is the improved path, and jp_2 and jp_4 are deleted. The deleting process can be written in Algorithm 1. ti is the iteration number, jp_{ck} is the ckth point in $Path_1$, and the same for jp_{ti}. We connect the third point in the original JPS path with the first point and check if the line collides with the occupied grid in the map. If it does not collide, the point before the checked point in the waypoint sequence of the original JPS path is deleted. The first and third points can be directly connected as the path. Then, the next point will be checked until all the point pairs (the two points are not adjacent) from $Path_1$ are checked, and all excess waypoints in $Path_1$ are removed. The simplified $Path_1$ is composed of $\{jp_1', \dots, jp_N'\}$.

Algorithm 1 Optimize the original JPS path

```
for jp_ck in Path_1 (ck is the iteration number): do
    ti = ck + 2
    while ti < len(Path_1) and len(Path_1) > 2 do
        if the line jp_ck jp_ti does not collide with the occupied grids in the 2D map: then
            ti = ti − 1, delete jp_ti from Path_1
        end if
        ti = ti + 1
    end while
end for
```

3.4. Shorter 3D path searching

After an improved 2D path is found, we notice an obviously shorter 3D path in some scenarios. For example, to avoid a wall, which has large width but limited height, flying above the wall is better than flying over a bypass from right or left. To search for a shorter 3D path with light computation, a generalized method DAGS for all environments is described in Algorithm 2, Figs. 5, and 6 in detail. It is composed of N rounds of search (N is the number of points in the simplified $Path_1$), and each round determines one straight line segment to compose the 3D path. At the beginning, Pcl_{lm} is divided into N parts $\{Pcl_{lm}^1, \dots, Pcl_{lm}^N\}$, the perpendicular projection of Pcl_{lm}^1 on $\overline{p_n g_l}$ falls between the projection of p_n and jp_1', and so on. As shown in Fig. 5, the first segment is $\overline{p_n tp_1}$, the second segment is $\overline{tp_1 tp_2}$, and $p_n - tp_1 - tp_2 - g_l$ represents the shorter 3D path. α_{res} is the angular resolution of the discrete angular graph. $A_g(\alpha_{g1}, \alpha_{g2})$ is the angular part of the spherical coordinates of g_l, and the origin of the spherical coordinate system is p_{sr} for each search round. $min()$ is a function that returns the minimal value of an array.

Here, the procedure for the first round of the search is briefly introduced, and the following rounds are basically identical. First, the discrete angular graph is built by Algorithm 2, line 3–8, as shown in Fig. 6(b). $\lfloor\ \rfloor$ returns the integer part of each element of the input. The angular coordinate in the graph is the direction angle difference between the goal g_l and any point in the space. The colored grids represent all the discrete angular coordinates A'_{mid} corresponding to the input point cloud. Then, the relative direction angle A_{eg} for $\overline{p_n tp_1}$

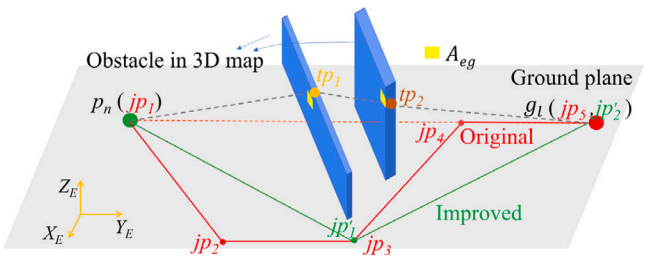


Fig. 5. A scenario where the 3D path is much shorter than the improved 2D path.

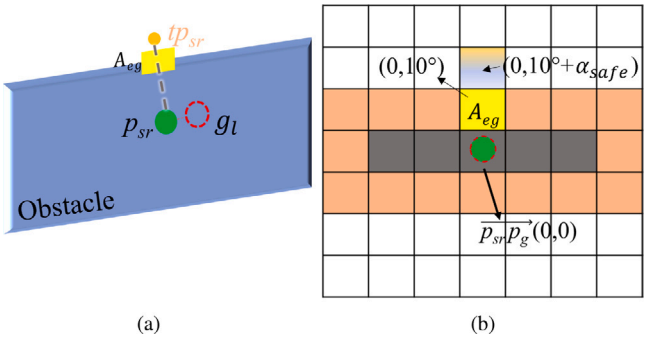


Fig. 6. (a): A wall stands between p_{sr} and g_l, the 3D path segment $\overline{p_{sr}tp_{sr}}$ is found. (b): The discrete angular graph for (a), $\alpha_{res} = 10°$. Please note that our 3D path searching method is universal to obstacle shapes. We draw the obstacle as a low square wall to correspond Fig. 5, which helps to explain our idea clearly.

is found (line 9), which has the minimal angle difference with $\overrightarrow{p_ng_l}$ (the yellow grid in Figs. 5 and 6). Next, the length of path segment $\overline{p_ntp_1}$ is determined in line 10, and the direction angle of this path segment in earth coordinate $E{-}XYZ$ is calculated by line 11. α_{safe} is the angle increment to make the path segment remain a safe distance from obstacles. Finally, the coordinate of the endpoint of this path segment is calculated in line 13.

If the 3D path is found by Algorithm 2, it is compared to the optimized $Path_1$, and the path with the shortest length is denoted as $Path_{fnl}$. Subsequently, the drone follows $Path_{fnl}$, and the MP is suspended until $Path_{fnl}$ collides with the obstacles in the updated map. For any obstacles stand between two waypoints of the 2D path $Path_1$ despite the shape of each, we always find a free grid on the angular graph (6(b)) unless the origin point p_{sr} in the search is tightly surrounded by obstacles in all directions (cannot get out by straight movement). Our proposed DAGS method is only a backup plan in MP, and it is not designed to always result in a 3D path, but it sacrifices completeness to obtain more computation efficiency. We assume the application scenarios of this system are common places for human ground activities, and it is not forced to go through narrow holes or gaps in the ceiling or floor. So 2D paths should exist for most navigation tasks, and MP still works by providing the 2D path planner's result if the 3D one fails. Even if both the 2D and 3D planner fails, PCP can still drive the drone to avoid nearby obstacles while pursuing the sliding local goal along the last final path (safety of the guidance path will not affect PCP's avoidance) until it reaches the final goal.

4. Trajectory planning on the point cloud

This section will introduce how the goal g_n is generated from $Path_{fnl}$ for the PCP's current step n and how the PCP outputs the final motion primitives. First, the discrete angular search (DAS) method specifies the safe waypoint $w_{pn} \in \mathbb{R}^3$ in free space, which the drone should traverse. The motion planner solves the optimization equation to make the drone pass through w_{pn} under the given motion constraints.

Algorithm 2 DAGS method

1: **for** sr in $[1,N]$ ($sr \in \mathbb{N}$ is the searching rounds number) **do**
2: If $sr = 1$, $p_{sr} = p_n$, otherwise $p_{sr} = tp_{sr-1}$
3: The angular coordinate of $\overrightarrow{p_{sr}g_l} \rightarrow A_g(\alpha_{g1}, \alpha_{g2})$
4: **for** each point p_{mi} in Pcl^{sr}_{lm}: **do**
5: The angular coordinate of $p_{mi} \rightarrow A_{mi}$
6: $A_{mi-g} = A_{mi} - A_g$
7: Discretize A_{mi-g}, $A'_{mi-g} = \lfloor A_{mi-g}/\alpha_{res} \rfloor$, build the discrete angular graph (Fig. 6(b)) with A'_{mi-g}
8: **end for**
9: The edge of all A'_{mi-g} in the angular graph $\rightarrow A_{eg-all}$, $A_{eg} \subset A_{eg-all}$ and $\|A_{eg}\|_2 = min(\|A_{eg-all}(1)\|_2, \|A_{eg-all}(2)\|_2, ...)$
10: Points in Pcl^{sr}_{lm} corresponding to $A_{eg} \rightarrow Pcl_{eg}$, $p_{eg} \subset Pcl_{eg}$ and p_{eg} has the maximal distance to $\overline{p_{sr}g_l}$, $l_{tp} = \overline{p_np_{eg}}$
11: Get the direction angle of $\overrightarrow{p_{sr}tp_{sr}}$, $A_{tp} = (\|A_{eg}\|_2 + \alpha_{safe})\frac{A_{eg}}{\|A_{eg}\|_2}$, $\alpha_{safe} = arcsin(r_{safe}/l_{tp})$
12: $tp_{sr} = p_{sr} + l_{tp}(cos(A_{tp}(1)), sin(A_{tp}(1)), sin(A_{tp}(2)))$
13: **end for**

The PCP also includes an additional safety measure to ensure that no collision will occur, which works when no w_{pn} can be found in an emergency.

4.1. Review of DAS

The PCP in this paper is an improved version of our previously proposed trajectory planner based on the DAS method [3]. Thus, briefly introducing it helps to understand the contributions in this section.

In [3], we first search a waypoint w_{pn} near the drone as the constraint for the motion planning, and a quadratic polynomial curve is optimized as the final trajectory in the motion planning. Motion planning is for solving a nonlinear optimization problem. The trajectory traverses w_{pn} within a small, predetermined distance error, and the corresponding motion primitives are within the kinematic constraints. Moreover, the real trajectory between p_n and w_{pn} can be proved collision-free [3]. As shown in Fig. 7, g_n is the goal point for the current step, a cluster of line segments fanned out in the direction of $\overrightarrow{p_ng_n}$, and these line segments have a common starting point p_n and the same length r_{det}. In this paper, g_n is determined by the planned path $Path_{fnl}$ (see Section 3.4). r_{det} is the point cloud distance threshold, and the collision check only considers the points within distance r_{det} from p_n. The two symmetrical lines about $\overrightarrow{p_ng_n}$ on the plane parallel to the ground plane are first checked (the red dashed circle and red arrows in Fig. 7) to determine if they collide with obstacles (minimal distance smaller than r_{safe}). r_{safe} is an important parameter for safety, which is a function of the preassigned maximal speed v_{max} and acceleration a_{max}. Then, the two symmetrical lines about $\overrightarrow{p_ng_n}$ in the vertical plane (green dashed circle and green arrows) are checked. Fig. 7 shows that when the first round of the search fails (marked with a red cross), another round with a greater angle difference is conducted until a collision-free line $\overline{p_np_w}$ is found. w_{pn} is on $\overline{p_np_w}$, and $\overline{p_nw_{pn}}$ should satisfy the safety analysis in [3]).

When the drone encounters a suddenly intruded obstacle in the sensor detection range, the PCP plans a safe trajectory in 0.02 s before the map and $Path_{fnl}$ are updated.

4.2. Connection between the PCP and MP

For the PCP, an updated goal point g_n is always required at every step n. The direction $\overrightarrow{p_ng_n}$ is the initial search direction. If this direction does not collide with any obstacles, the planned trajectory will head to g_n directly.

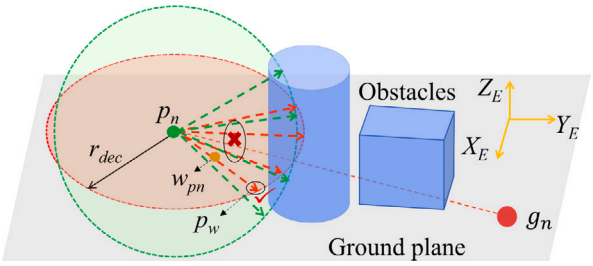


Fig. 7. Illustration of the DAS process. The red line segments with arrowheads are on the horizontal plane used for collision check, and the green lines are in the vertical plane.

The final path $Path_{fnl} = [pt_1, pt_2, \dots, g_l]$ is received from the MP ($Path_{fnl}$ dose not include p_n), and an optimization problem (6) is designed to find the current goal g_n. It is designed to make the final trajectory smooth by sliding g_n continuously. If pt_1 is simply assigned as g_n, g_n will jump to pt_2 as the drone approaching pt_1. This result may cause w_{pn} to also jump with g_n and cannot be reached within the drone's kinematic constraints. The drone should start to turn earlier to avoid a violent maneuver, which leads to a greater control error and undermines safety. The endpoint of the planned trajectory remains near $Path_{fnl}$ under the premise of safety assurance.

$$\begin{aligned} \min_{v_t} \quad & \|v_t \frac{\|v_0\|_2}{\|v_t\|_2} - (p_n + v_0)\|_2 + \|v_t - \kappa_1 a_1\|_2 + \|v_t - \kappa_2 a_2\|_2 \\ \text{s.t.} \quad & a_1 = pt_1 - p_n,\ a_2 = pt_2 - p_n,\ v_t = \overrightarrow{p_n g_n} \end{aligned} \tag{6}$$

In (6), the three components are the acceleration cost, the cost of pt_1, and the cost of pt_2. p_n is the current position of the drone, and v_0 is the current velocity. v_t presents the initial search direction of the local planner. r_{det} is the search range radius for the DAS. κ_1 and κ_2 are the weight factors for adjusting the influence of pt_1 and pt_2 on v_t, and κ_1 is much larger than κ_2. Fig. 8 intuitively demonstrates the initial search direction v_t for the PCP, drone position, and waypoints on $Path_{fnl}$. The green, dashed line displays the rough shape of the trajectory if the PCP does not check for a collision and w_{pn} is always on $\overline{p_n g_n}$. We can see from (6) and Fig. 8 that as the drone approaches the next waypoint pt_1, the influence on g_n from a_2 overwhelms a_1. When the drone is far from pt_1 and pt_2, a_1 is the governing influence factor of v_t. We can also reduce the difference between the trajectory and $Path_{fnl}$ by regulating κ_1 and κ_2. If only one waypoint g_l remains in $Path_{fnl}$, $pt_1 = pt_2 = g_l$.

Solving a nonlinear optimization problem, such as (6), is computationally expensive. From a geometric point of view, the nature of (6) is to find a point (v_t) in the space with the minimal total distance between three fixed points ($\kappa_1 a_1 + p_n$, $\kappa_2 a_2 + p_n$, $v_0 + p_n$). The triangle composed of these three points is called a Fermat triangle. It can be solved by locating the **Fermat point** f_m of the triangle, as shown in Fig. 8, and $g_n = f_m$. The calculation of f_m is illustrated in (7). First, the plane coordinates (x_{f1}, y_{f1}), (x_{f2}, y_{f2}), and (x_{f3}, y_{f3}) in the plane P_{fm} for the three points are determined (P_{fm} is the plane defined by the Fermat triangle). S_{fm} is the area of the triangle. l_1, l_2 and l_3 is the length of the side opposite to the triangle vertex point (x_{f1}, y_{f1}), (x_{f2}, y_{f2}), and (x_{f3}, y_{f3}) respectively. $f'_m(x_{fm}, y_{fm})$ is the coordinate of f_m in plane P_{fm}. Finally, f'_m is converted to $E - XYZ$ to obtain f_m.

$$\begin{aligned} x_{fm} &= (\sum_{i=1}^{3} x_{fi}(4S_{fm} + \sqrt{3}l_i^2) + g(y))/f_{Sl} \\ y_{fm} &= (\sum_{i=1}^{3} y_{fi}(4S_{fm} + \sqrt{3}l_i^2) + g(x))/f_{Sl} \\ g(x) &= [x_{f1}, x_{f2}, x_{f3}][l_3^2 - l_2^2, l_1^2 - l_3^2, l_2^2 - l_1^2]^T \\ g(y) &= [y_{f1}, y_{f2}, y_{f3}][l_3^2 - l_2^2, l_1^2 - l_3^2, l_2^2 - l_1^2]^T \\ f_{Sl} &= 12S_{fm} + \sqrt{3}(l_1^2 + l_2^2 + l_3^2) \end{aligned} \tag{7}$$

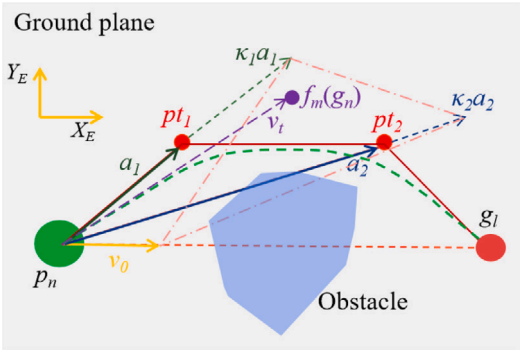


Fig. 8. Geometric illustration of the analytical solution of (6). The pink, dashed line marks the Fermat triangle. The marker (dot and line) and the corresponding symbol are in the same color.

4.3. Improvements on the PCP

4.3.1. Improve the motion primitives generation efficiency

After obtaining the next waypoint w_{pn}, the next step is to calculate the motion primitives and send the command to the flight controller. Sending w_{pn} directly as the control command may cause the flight to be unstable, and the acceleration magnitude may exceed a_{max}. Because w_{pn} may vary significantly between two continuous motion planning steps, the point cloud quality is harmed when the drone acceleration magnitude is too large. In addition, the position commands cannot control the speed. To ensure that the aircraft can fly within its kinematic limits and reach the next waypoint, the motion primitives are generally obtained by solving an optimization problem. Previously [3], we considered the flight time to reach w_{pn} the optimization variable. However, we found the solver may fail in the given number of iteration steps in some cases. The solving success rate with an error tolerance 10^{-3} is approximately 83.7% within 40 steps. When the solver fails in several continuous PCP steps, the planned trajectory deviates considerably from w_{pn}, and the drone is very dangerous. Furthermore, considering the requirement of low time cost for real-time computing, the maximal number of iteration steps should be limited. The time variable increases the problem complexity, and the flight controller does not require it. Therefore, the flight time can be removed from the optimization variables and the optimization strategy (8) is proposed. It is slightly different from that of [3], to improve the success rate and time cost.

$$\begin{aligned} \min_{a_n} \quad & \|a_n\|_2^2 + \eta_1 \|\overrightarrow{w_{pn} p_{n+1}}\|_2 + \eta_2 \frac{\|\overrightarrow{p_n p^*_{n+1}} \times \overrightarrow{p^*_{n+1} w_{pn}}\|_2}{\|\overrightarrow{p_n w_{pn}}\|_2} \\ \text{s.t.} \quad & v_n = \dot{p}_n,\ a_n = \dot{v}_n \\ & \|v_{n+1}\|_2 \le v_{\max},\ \|a_n\|_2 \le a_{\max} \\ & v_{n+1} = v_n + a_n t_{avs} \\ & p_{n+1} = p_n + v_n t_{avs} + \frac{1}{2} a_n t_{avs}^2 \\ & p^*_{n+1} = p_n + 2v_n t_{avs} + 2a_n t_{avs}^2 \end{aligned} \tag{8}$$

In the revised optimization formula, we fix the trajectory predicting time to t_{avs}. t_{avs} is the average time cost of the last 10 executions of the PCP. The endpoint constraint is moved to the objective function. The endpoint of the predicted trajectory need not coincidence with w_{pn}. Because the execution time of the PCP is always much smaller than the planned time to reach w_{pn}, before the drone reaches w_{pn}, a new trajectory is generated, and then the remainder of the formerly predicted trajectory is abandoned. Predicting only the trajectory between the current step to the next step of the PCP is sufficient. Therefore, minimizing the distance between the trajectory endpoint at t_{avs} and w_{pn} is reasonable. Given that the current step run time of the PCP may exceed t_{avs}, the distance from the trajectory endpoint at $2t_{avs}$ to $\overline{p_n w_{pn}}$

should also be optimized. The subscript n presents the current step in a rolling process of the PCP. $v_n \in \mathbb{R}^3$ and $a_n \in \mathbb{R}^3$ are the current velocity and acceleration of the drone. v_{n+1}, p_{n+1}, and p^*_{n+1} are the vehicle's states in the future predicted using the kinematic formula. v_{max} and a_{max} are the kinematic constraints for speed and acceleration, respectively. η_1, η_2 are the weight factors for the trajectory endpoint constraint.

After the modification, the success rate with error tolerance 10^{-3} within 20 steps is increased to 99.8%, and no dangerous trajectory deviation from w_{pn} can be detected. The time cost of the motion planning and safety of the PCP is greatly improved.

4.3.2. Safety backup plan

On some occasions, such as when the obstacles are too dense or an obstacle suddenly appears near the drone (distance is smaller than r_{safe}), DAS may fail to find a feasible direction. To solve this problem, the minimum braking distance $d_{bkd} = \frac{\|v_n\|_2^2}{2a_{max}}$ at current velocity v_n is introduced. It is smaller than r_{safe} by setting the appropriate maximum acceleration constraint a_{max} and velocity constrain v_{max} ($\|v_n\|_2 \le v_{max}$). If the minimum distance from the drone to obstacles is greater than d_{bkd}, the search direction having the maximum distance to the obstacles is chosen (although the distance is smaller than r_{safe}). Otherwise, the drone brakes immediately and flies back to the position at the former PCP step, and the chosen search direction of the former step will not be considered after the drone has flown back in place. In addition, when the motion primitives cannot be solved successfully by Eq. (8), the underlying controller of the drone takes the position command w_{pn} (obtained by DAS, see Section 4.1) only to make the obstacle avoidance and navigation proceed in a down-graded manner. There measures are called the "safety backup plan".

4.4. The whole framework

To summarize, Algorithm 3 shows the overall proposed framework. The two planners (MP and PCP) are designed to run in Robot Operating System (ROS) parallelly and asynchronously[2] because of their large difference in operation time and share all the data involved in the calculation via communications between CPU processes. In addition, the point cloud filter and the mapping kit are run in parallel on different threads.

Algorithm 3 our proposed framework

```
while true: (Thread 1) do
    Filter the raw point cloud data, output Pcl_2
end while
while true: (Thread 2) do
    Build a global 3D map Pcl_m, project the piece around the drone's
    height on the ground to obtain Map_1
end while
while the goal is not reached: (Thread 3) do
    if the shorter 3D path has not been found or it collides with the
    updated Pcl_m: then
        Apply downsampling on Map_1 to get Map_1b, find the 2D path
        Path_1 on the stitched map (Map_1b,Map_c).
        Try to find a shorter 3D path, output Path_fnl
    end if
end while
while the goal is not reached: (Thread 4) do
    Calculate the goal g_n from Path_fnl
    Find the waypoint w_pn by DAS method
    if found a feasible waypoint: then
        Run the motion planner to get motion primitives
    else
        Run the safety backup plan, and go to line 14
    end if
    Send the motion primitives to the flight controller
end while
```

5. Test results

In this section, the static tests and real-time flight tests are introduced to validate the effectiveness of the methods in our proposed framework.

5.1. Algorithm performance static test

To prove our proposed algorithms' effectiveness, we first individually test them offline with static data input. This approach can avoid the influence from the fluctuations in computing performance caused by other simultaneously running algorithms when one algorithm is analyzed. Moreover, the data can be customized, so the tests are more effective and targeted. In this subsection, all the time costs are measured on a personal computer with an Intel Core i7-8565U 1.8–4.6 GHz processor and 8 GB RAM, and Python 2.7 is used as the programming language.

[2] They are two programs running on two different processes of CPU but can communicate at a given frequency. Please see this link for detail.

5.1.1. Path planning on the 2D map

The size of the local map is the main influencing factor of the actual flight trajectory length and computing time of each replanning step. In addition, we apply the JPS algorithm twice on two maps of different sizes and resolutions and splice the two paths into a whole. The Map_c size is also a key to balancing the time cost and the path length. As the effectiveness of our proposed method should be verified and analyzed, two rounds of numerical simulations are designed and conducted.

The first round tests the influence of the local map size, the second-round tests the effect of the Map_c size (see Fig. 4). A large-scale 2D map is used in the numerical tests, as shown in Fig. 9. The map size is 800 m $*$ 800 m, and the local map size is tested with three configurations (unit: m): 75 $*$ 75, 200 $*$ 200, and 400 $*$ 400. The sizes of Map_1 and Map_c in Fig. 9(b) are 200 $*$ 200 and 100 $*$ 100 (unit: m), and the configuration for the obstacle inflation and map downsampling is the same with Fig. 4. The blue line indicates the real trajectory of the drone, the red dash line indicates the global JPS path, and the purple line is the JPS path on the local map. We assume the local map center (robot position) moves along the local map path and can only move one meter (including the diagonal move) in one step. The test is conducted with 10 combinations of the randomly assigned start point and goal point, whose straight-line distance is greater than 500 m. For each local map size, the 10 combinations are identical.

For the first round, the average time cost and real trajectory length are compared with that of the planning on the entire map, as shown in Table 1. Len_1 represents the average trajectory length, while Len_2 denotes the average global JPS path length. T_{c1} is the average total computing time of each replanning step with the local map (including the map downsampling), and T_{c2} is that of the global planning. Table 1 shows that Len_1 does not increase substantially compared to Len_2, while the time cost is saved considerably compared to the global planning time. We use green color to highlight the data corresponding to our proposed method, and use **bold characters** to mark the best performance. Len_1 increases by only 2.2% and T_{c1} decreases by 96.6% compared to Len_2 and T_{c2}, respectively, when the Map_1 size is 200 m $*$ 200 m. It is the most time-efficient map size among the three tested sizes, and the effectiveness of planning on the local 2D map is verified.

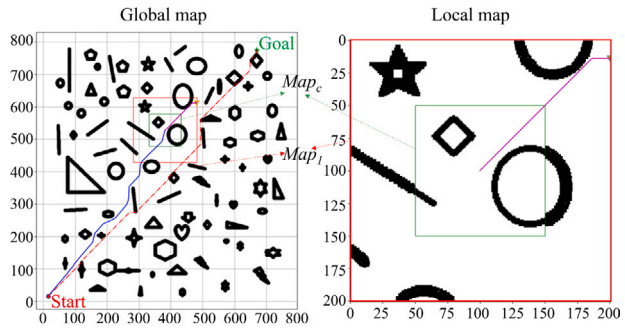


Fig. 9. Visualized result during the numerical simulation when the double layer map is used.

Table 1
Test results of different Map_1 sizes.

Map_1 size (m)	Len_1 (m)	T_{c1} (s)	Len_2 (m)	T_{c2} (s)
75 ∗ 75	1021.962	0.036	980.439	3.454
200 ∗ 200	**1001.783**	**0.118**	**980.439**	**3.454**
400 ∗ 400	997.486	0.284	980.439	3.454

Table 2
Test results of different Map_c sizes.

Map_c size (m)	Len_3 (m)	T_{c3} (s)	Len_1 (m)	T_{c1} (s)
70 ∗ 70	1110.374	0.040	1001.783	0.118
100 ∗ 100	**1004.848**	**0.055**	**1001.783**	**0.118**
120 ∗ 120	1002.917	0.082	1001.783	0.118

For the second round, the size of Map_1 is fixed at 200 m ∗ 200 m and the size of Map_c has three alternatives (unit: m): 70 ∗ 70, 100 ∗ 100, and 120 ∗ 120. In Table 2, the average total computing time T_{c3} and trajectory length Len_3 are compared between the tests that do and do not use the stitched map. When the size of Map_c is 100 m ∗ 100 m, the real trajectory length Len_3 increases by only 0.31%, while the time cost T_{c3} is reduced by 53.4% compared to Len_1 and T_{c1}, respectively. Thus, the effectiveness of planning on the multi-resolution hybrid map is well validated.

5.1.2. The improvements in optimization formulation

After several hardware flight tests with our proposed framework, we record all the required data for solving the optimization problem, including p_n, v_n, and w_{pn}, at each step (over 5.3×10^4 steps in total). To validate the improvements of the optimization formula, the collected data is input to the original optimization formula and the improved one in this paper for comparison. In addition, the optimization solving performance under different maximum iterating numbers is studied. The average time cost and overall success rate are counted for quantitative comparison. In Table 3, R_{og} and T_{og} are the solution success rate and the average time cost of the original optimization formula, respectively, and R_{im} and T_{im} are for the improved version. We can see that the success rate and time cost are greatly improved. When the maximum step is more than 20, the success rate improvement is minor. Because 99.83% is a satisfactory success rate, we set the maximum step number as 20 to reduce the time cost. The motion optimization problem-solving time decreases by 39.33% compared to that of the original formula.

5.2. Simulated flight tests with real-time planning

Compared to the counterpart that follows the original JPS path directly on the 2D map, our proposed framework is proved to shorten the actual trajectory substantially with limited additional time cost. Another test is required to compare the final trajectory length and time cost with the 3D global optimal path searching methods to further validate the framework's performance. However, the globally optimal path can only be obtained after the map is entirely constructed, so we cannot obtain it after exploring the environment. We adopt the asymptotically optimal method RRT* to generate the globally optimal path using a large amount of offline iteration computing. Also, we use JPS3D algorithm for global path planning to enrich the comparison, and it can ensure RRT* has converged to the global optimum. For comparison, the same simulation configuration with the former flight tests is used in this test. We first fly the drone manually with the mapping kit to build up the complete 3D map for the simulation world (Fig. 10(b)). Then, we apply the JPS method by converting the point cloud map into the voxel map to obtain the globally shortest path. Next, the RRT* method is applied, the iteration breaks until the resulted path length is very close to the JPS path (<1% for relative difference). Table 4 describes the parameter settings of the framework in the simulation test. "pcl" is short for the point cloud. The simulation flight tests are conducted 10 times with different randomly assigned initial and goal positions, with the straight distance from 30 m to 40 m.

Table 3
Test results for the improvements of optimization formula.

Max steps	R_{og} (%)	R_{im} (%)	T_{og} (ms)	T_{im} (ms)
5	41.51	89.12	3.24	2.12
10	61.14	95.49	4.87	2.94
20	**76.87**	**99.83**	6.56	**3.98**
40	83.68	99.96	9.45	5.23
80	92.15	100.00	14.78	5.61

Table 4
Parameters for the framework.

Parameter	Value	Parameter	Value
l_{ms}	20 m	h_{ms}	6 m
α_{res}	10°	i, j	100
m, n	50	k	3
h	2	r_{safe}	0.5 m
Voxel size	0.2 m	pcl frequency	30 Hz
Depth resolution	640 ∗ 360	κ_1, κ_2	4.2, 1.5
η_1, η_2	40, 10	$\|p_n - w_{pn}\|_2$	0.3 m

The mean of the 10 flight tests' results are obtained for comparison. The mean of length of the actual trajectory, RRT* path, and JPS path is 37.024 m, 32.985 m, and 32.822 m, respectively. We use the average JPS path length as the globally optimal path length. The actual path length increases by approximately 12.8% compared to the global optimum. t_{mp} is the average step time cost of MP in flight. For time cost, our proposed method takes 0.043 s for each MP re-planning step, while RRT* costs 5.243 s and JPS3D costs 2.151 s. Thus, our proposed method results in the comparable path length while takes much fewer computing time.

Fig. 10 illustrates the detailed visualized data of the flight test corresponding to one of the 10 flights. The initial point $(11.4, 14.4, 0.8)$ and goal point $(-12.0, -10.0, 1.2)$ of the resulted flight trajectory is shown in Fig. 10(a). The global optimal path length for this flight is 33.252 m, and the actual flight trajectory length is 36.057 m. In Fig. 10(b) we show the comparison of the globally optimal paths (found by RRT* and JPS3D) and the actual trajectory in (a), the 3D map is shown in the form of a point cloud. We see that the real trajectory has obvious differences with the other optimal paths. Nevertheless, the path length gap is not large, which is similar to the numerical test results on the 2D map.

The trajectory length comparison between our proposed framework and the SOTA algorithms is shown in Table 5. [3] is our former work proposes a local motion planner. η_{3D} indicates the relative length increment comparing the resulted path with the optimal 3D path. The results are evaluated from the flight test data in the same simulation world, with the open-sourced code [38] and the default parameters. We conduct 20 flights for each work and the initial and goal position

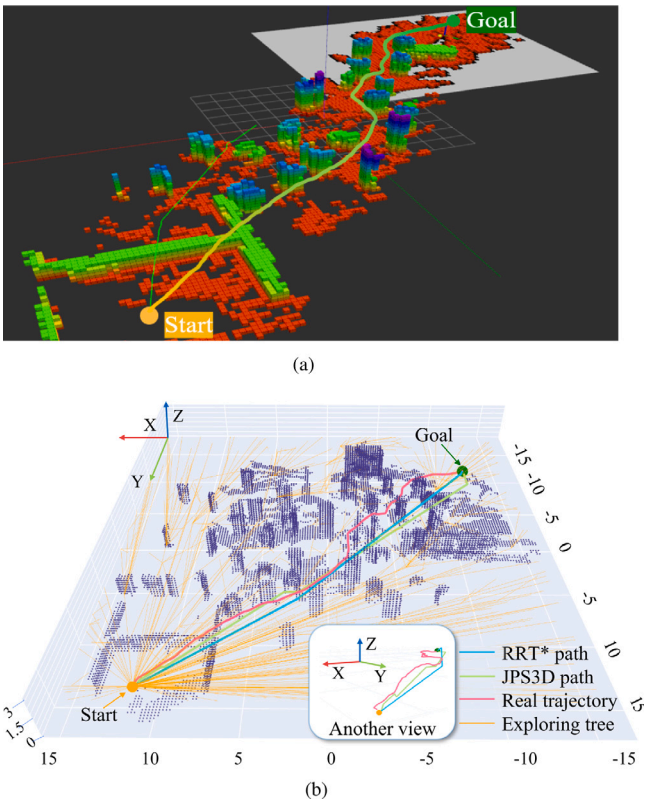


Fig. 10. The visualized results for the flight tests in Gazebo.

Table 5
3D path length comparison with the SOTA algorithms.

Work	η_{3D}	Success rate	Failure position
Ours	**12.8%**	**100%**	N/A
[3]	29.5%	85%	High wall
[38]	20.2%	**100%**	N/A

is randomly assigned as the above tests, the maximum velocity and acceleration are 1 m/s and 5 m/s^2, respectively.

Our proposed framework can fly the shortest trajectory compared to the works listed in Table 5, also the success rate is the highest. [3] suffers from the narrow range of the collision check (perception range), and the range cannot be extended due to the nature of the HAS method, so the resulted trajectory is longer and it is easy to be trapped in complex scenarios. Our improved PCP can brake in time and chose another direction thanks to the effective safety backup plan, thus resulted in better safety. Also, with the help of MP, the proposed framework can avoid the obstacles earlier following a short path. Although the planning range of [38] is farther, the front-end path has no optimality guarantee even in a local map, thus the resulted trajectory is longer.

5.3. Hardware flight tests

The video for the hardware test has been uploaded online.[3] In the test environment, static and dynamic obstacles are present to validate the fast reaction of the PCP.

5.3.1. Introduction of hardware platform

We conduct the hardware tests on a self-assembled quadrotor, as shown in Fig. 11. The Intel RealSense depth camera D435i is installed

Fig. 11. Hardware configuration of our drone. The onboard computer is mounted beneath the quadrotor frame.

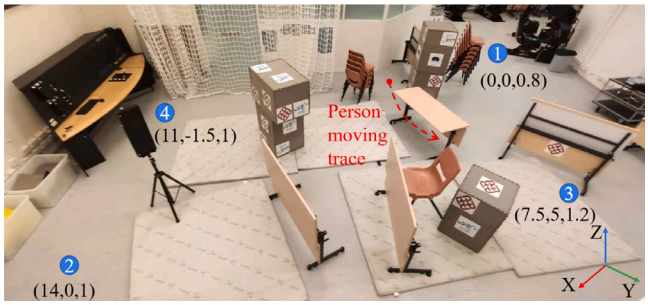


Fig. 12. Indoor environment for the hardware flight tests. Please refer to the associated video for more details about the experiment.

under the frame as the only perception sensor. The drone frame is QAV250, with the diagonal length 25 cm. The Pixracer autopilot with a V1.10.1 firmware version is adopted as the underlying flight controller. A LattePanda Alpha 800S with an Intel Core M3-8100Y, dual-core, 1.1–3.4 GHz processor and 8 GB RAM is installed as the onboard computer, where all the following timing breakdowns are measured. For the hardware test, we fly the drone in multiple scenarios with our self-developed visual-inertial odometry (VIO) kit[4] to demonstrate the practicality of our proposed framework. The point cloud filter, the VIO kit, and the mapping kit are executed by C++ code, while the other modules are run by the Python scripts. All the parameters used in the hardware test are identical to the simulation flight tests, as shown in Table 4. The yaw angle of the drone is controlled to keep the camera always heading toward the local goal g_l'.

5.3.2. Indoor tests

Fig. 12 is our indoor flight test environment. First, the drone takes off from point 1 and flies through the cluttered static obstacles (shown in the picture), following the sequence 2-3-4. The environment is unknown to the drone before it takes off. We can see from the video that the drone can avoid obstacles agilely. Meanwhile, the drone posture is stable and the final flight trajectory is smooth. After the drone reaches point 3 and starts flying toward point 4, a person who hides behind the boxes suddenly appears closely in front of the drone. In the video, the drone quickly maneuvers after the person appears, and the avoidance is successful. The map is updated afterward, and the drone continues to follow the path from the MP.

The constructed voxel map and flight trajectory after this flight are shown in Fig. 13. The position, attitude, and velocity curves of the drone can be found in Fig. 14. The framework begins to work at time 0 s, and the data shown in the figures start at 70.38 s and end at 94.31 s, corresponding to the flight from point 3 to point 4. The moving person enters the depth camera's FOV at 76.09 s (marked with

[3] https://youtu.be/AOENvwf8sfM.

[4] https://github.com/HKPolyU-UAV/FLVIS.

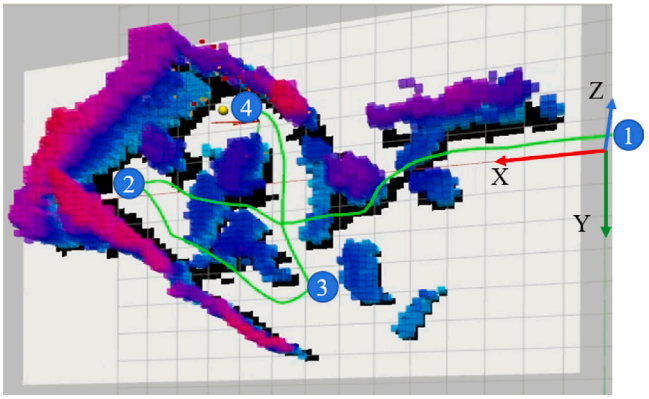


Fig. 13. Explored map and the drone trajectory after the hardware flight test in the static environment.

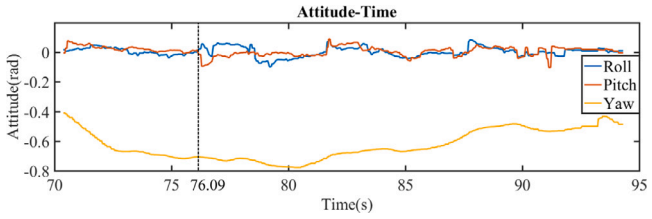


Fig. 14. The UAV attitude angles when dealing with a dynamic obstacle.

the vertical dashed lines). The attitude angles react immediately to the appearance of the person, and then the velocity changes considerably. Sequentially, the drone decelerates and flies to the left side to pass the person. The maximum speed reaches 1.23 m/s at 74.40 s. Because $\|\overrightarrow{p_n w_{pn}}\|_2$ is assigned to be always greater than $\|\overrightarrow{p_n p_{n+1}}\|_2$, in the motion optimization problem (8) minimizing the cost $\eta_1\|\overrightarrow{w_{pn}p_{n+1}}\|_2$ leads to positive acceleration when no obstacle is around. The drone will decelerate when the obstacles are near it, because of the acceleration cost and the endpoint cost in the objective function.

5.3.3. Outdoor tests

In addition, the proposed framework is tested in diverse outdoor scenarios. Fig. 15 shows the environments of two experiments, and the flight tests in other environments are included in the video. Fig. 15(a) is entirely static, with several obstacles standing on a slope and dense bushes and trees. This environment is challenging, requiring the 3D precise trajectory planning and motion control. Fig. 15(b) is a larger environment compared to those of the tests above. In addition to the complex static obstacles, five moving people in the field continuously interrupt the drone from the original planned path and validate its reaction performance. The video demonstrates that the drone performs agile and safe flights in various test environments, so the practicability and flight efficiency of our proposed framework can be proven.

Finally, the average time cost of each part of the MP and PCP for the hardware tests is counted and analyzed in Fig. 16 to show the computational efficiency. In Fig. 16, the average time cost and the percentage are provided on a pie chart. For the MP, most of the time cost (63%) is used for path planning and optimization on the 2D map. The path search with the 3D point cloud is computationally inexpensive. The average total time cost of each MP step is 0.078 s (Python implementation), and the loop frequency is approximately 12 Hz. These results are slower than the offline test results because the computing resource is occupied by the other part of the framework (VIO, mapping kit, point cloud filter, and the PCP). For the PCP, the average time cost of the PCP step is 16.2 ms, of which searching for w_{pn} is the most time-consuming part, contributing 65.2% of each step

Fig. 15. Two of the outdoor flight test environments. (a) locates in a campus and (b) is at the sports corner in a park. Please refer to the associated video for more details about the experiment.

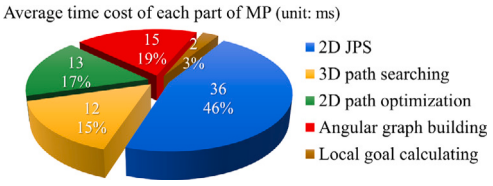


Fig. 16. Average time cost and the proportion of each submodule of MP.

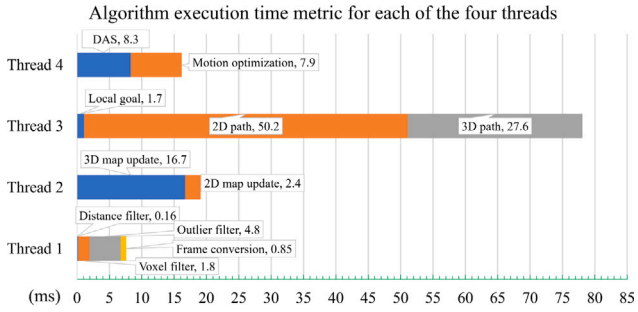


Fig. 17. Average execution time cost for algorithms on each thread.

for average. We also calculate the average execution time cost for the algorithms on each thread, and the results are detailed in Fig. 17.

Moreover, the time cost is compared with those of the SOTA algorithms in Table 6. Because our proposed method is composed of two planners running asynchronously, no single value represents the framework execution time. Thus, the average step time costs of the MP and PCP are listed for the comparison. Notably, the time costs of the related works are measured on different hardware platforms with different program code types, and we obtain the data directly from the publications. "MSCF" is the abbreviation for the maximum single-core frequency of the hardware platform processor. Although Table 6 cannot be used for the absolute performance comparison, the trajectory replanning time cost of our proposed method (PCP) is believed promisingly the best level among the SOTA algorithms.

At last, we evaluate the real-time performance of MP (only 2D path planning and DAGS) on our tiny onboard computer and comparing with JPS3D, both the algorithms are implemented with C++ in this test for the best efficiency. The results can be found in Table 7, where condition "vacant" means that only the tested algorithm running on the computer, and "full-load" means that the d435i API, VIO, and mapping kit are all running to simulate the computing pressure of the autonomous flight field tests, together with the tested algorithm. We first recorded all the required data from the onboard camera and IMU into a ROS bag. After that, we replay the bag and measure the time costs under the two conditions. We can conclude that our method always cost less time than JPS3D and can run in real-time even in full-load

Table 6
Comparison with state-of-the-art algorithms.

Works	Time cost (ms)	MSCF (GHz)
MP	78	3.4
PCP	**16.2**	**3.4**
[30]	>100	3.0
[27]	>160	3.4
[39]	>40	N/A
[3]	19	4.6
[40]	199	3.1
[29]	106	3.0

Table 7
Real-time computing performance comparison.

Algorithm	Condition	Time cost (ms)
MP	Vacant	**18.6**
MP	Full-load	**42.1**
JPS3D	Vacant	64.8
JPS3D	Full-load	142.4

condition. Although JPS3D can run in real-time with C++ implementation when the computing resource is sufficient, it is no longer real-time to meet the map updating rate (15 Hz) when the onboard computer is much occupied. Thus, our approach is more practical in real-world applications with limited computing resource.

6. Conclusion and future work

In this paper, a framework of trajectory planning for UAVs with two parallel planners is introduced. The map planner tries to find the shortest possible path in limited computational time. The point cloud planner takes effect when the point cloud near the drone differs from the 3D map to ensure safety. It reacts much faster than the path planning on the map. The test results verify that the techniques proposed in this paper can reduce the computing time cost, with the performance basically unchanged or even improved compared to that of the original method [3]. The real-time flight trajectory length outperforms those of the SOTA algorithms, and the reacting time of the PCP is also superlative. The entire framework is tested extensively in simulation and hardware experiments, demonstrating excellent rapid response capabilities and flight safety.

However, the test environments do cover all the UAV application scenarios. Moreover, the UAV flight speed in our tests is not sufficiently high. In the future, the framework will be tested in more challenging environments with a higher vehicle speed than the current study.

CRediT authorship contribution statement

Han Chen: Conceptualization, Methodology, Software, Simulation, Field tests, Writing - original draft. **Shengyang Chen:** Software, Field tests. **Chih-Yung Wen:** Supervision, Resources, Manuscript reviewing. **Peng Lu:** Conceptualization, Discussion, Funding acquisition, Supervision, Manuscript reviewing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgment

The authors would like to thank Mr. Ching-wei Chang for his kind help in the hardware equipment debugging and Miss Yuyang Hu for her assistance in the hardware tests.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.mechatronics.2023.103094.

References

[1] Liu S, Watterson M, Mohta K, Sun K, Bhattacharya S, Taylor CJ, Kumar V. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments. IEEE Robot Autom Lett 2017;2(3):1688–95. http://dx.doi.org/10.1109/LRA.2017.2663526.

[2] Oleynikova H, Taylor Z, Siegwart R, Nieto J. Safe local exploration for replanning in cluttered unknown environments for microaerial vehicles. IEEE Robot Autom Lett 2018;3(3):1474–81. http://dx.doi.org/10.1109/LRA.2018.2800109.

[3] Chen H, Lu P. Computationally efficient obstacle avoidance trajectory planner for UAVs based on heuristic angular search method. In: 2020 IEEE/RSJ international conference on intelligent robots and systems (IROS). 2020, p. 5693–9. http://dx.doi.org/10.1109/IROS45743.2020.9340778.

[4] Lopez BT, How JP. Aggressive collision avoidance with limited field-of-view sensing. In: 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS). 2017, p. 1358–65. http://dx.doi.org/10.1109/IROS.2017.8202314.

[5] Mohta K, Watterson M, Mulgaonkar Y, Liu S, Qu C, Makineni A, Saulnier K, Sun K, Zhu A, Delmerico J, et al. Fast, autonomous flight in GPS-denied and cluttered environments. J Field Robotics 2018;35(1):101–20.

[6] Zhou B, Zhang Y, Chen X, Shen S. FUEL: Fast UAV exploration using incremental frontier structure and hierarchical planning. IEEE Robot Autom Lett 2021;6(2):779–86.

[7] Duchoň F, Babinec A, Kajan M, Beňo P, Florek M, Fico T, Jurišica L. Path planning with modified a star algorithm for a mobile robot. Procedia Eng 2014;96:59–69. http://dx.doi.org/10.1016/j.proeng.2014.12.098.

[8] Dey D, Shankar KS, Zeng S, Mehta R, Agcayazi MT, Eriksen C, Daftry S, Hebert M, Bagnell JA. Vision and learning for deliberative monocular cluttered flight. In: Field and service robotics. Springer; 2016, p. 391–409. http://dx.doi.org/10.1007/978-3-319-27702-8-26.

[9] Florence PR, Carter J, Ware J, Tedrake R. NanoMap: Fast, uncertainty-aware proximity queries with lazy search over local 3D data. In: 2018 IEEE international conference on robotics and automation (ICRA). 2018, p. 7631–8. http://dx.doi.org/10.1109/ICRA.2018.8463195.

[10] Schulman J, Duan Y, Ho J, Lee A, Awwal I, Bradlow H, Pan J, Patil S, Goldberg K, Abbeel P. Motion planning with sequential convex optimization and convex collision checking. Int J Robot Res 2014;33(9):1251–70. http://dx.doi.org/10.1177/0278364914528132.

[11] Augugliaro F, Schoellig AP, D'Andrea R. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In: 2012 IEEE/RSJ international conference on intelligent robots and systems. 2012, p. 1917–22. http://dx.doi.org/10.1109/IROS.2012.6385823.

[12] Lau B, Sprunk C, Burgard W. Improved updating of euclidean distance maps and voronoi diagrams. In: 2010 IEEE/RSJ international conference on intelligent robots and systems. 2010, p. 281–6. http://dx.doi.org/10.1109/IROS.2010.5650794.

[13] Elfes A. Using occupancy grids for mobile robot perception and navigation. Computer 1989;22(6):46–57. http://dx.doi.org/10.1109/2.30720.

[14] Hornung A, Wurm KM, Bennewitz M, Stachniss C, Burgard W. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. Auton Robots 2013;34(3):189–206. http://dx.doi.org/10.1007/s10514-012-9321-0.

[15] Choi S, Park J, Lim E, Yu W. Global path planning on uneven elevation maps. In: 2012 9th international conference on ubiquitous robots and ambient intelligence (URAI). 2012, p. 49–54. http://dx.doi.org/10.1109/URAI.2012.6462928.

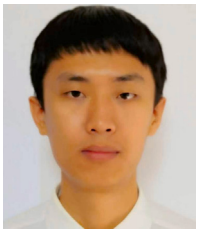
[16] Coenen S, Lunenburg J, van de Molengraft M, Steinbuch M. A representation method based on the probability of collision for safe robot navigation in domestic environments. In: 2014 IEEE/RSJ international conference on intelligent robots and systems. 2014, p. 4177–83. http://dx.doi.org/10.1109/IROS.2014.6943151.

[17] Dijkstra E. A note on two problems in connexion with graphs. Numer Math 1959;1:269–71. http://dx.doi.org/10.1007/BF01386390.

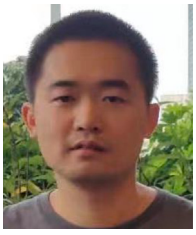
[18] Hart PE, Nilsson NJ, Raphael B. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans Syst Sci Cybern 1968;4(2):100–7. http://dx.doi.org/10.1109/TSSC.1968.300136.

[19] Likhachev M, Gordon GJ, Thrun S. ARA*: Anytime A* with provable bounds on sub-optimality. Adv Neural Inf Process Syst 2003;16. http://dx.doi.org/10.5555/2981345.2981441.

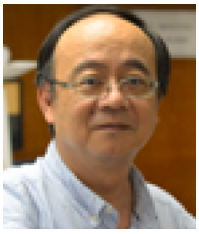
[20] Harabor D, Grastien A. Online graph pruning for pathfinding on grid maps. Proc AAAI Conf Artif Intell 2011;25(1):1114–9. http://dx.doi.org/10.5555/2900423.2900600.
[21] Dolgov D, Thrun S, Montemerlo M, Diebel J. Path planning for autonomous vehicles in unknown semi-structured environments. Int J Robot Res 2010;29(5):485–501. http://dx.doi.org/10.1177/0278364909359210.
[22] LaValle SM, Kuffner JJ. Randomized kinodynamic planning. Int J Robot Res 2001;20(5):378–400. http://dx.doi.org/10.1177/02783640122067453.
[23] Kavraki L, Svestka P, Latombe J-C, Overmars M. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans Robot Autom 1996;12(4):566–80. http://dx.doi.org/10.1109/70.508439.
[24] Karaman S, Frazzoli E. Sampling-based algorithms for optimal motion planning. Int J Robot Res 2011;30(7):846–94. http://dx.doi.org/10.1177/0278364911406761.
[25] Webb DJ, van den Berg J. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In: 2013 IEEE international conference on robotics and automation. 2013, p. 5054–61. http://dx.doi.org/10.1109/ICRA.2013.6631299.
[26] Chen H, Lu P, Xiao C. Dynamic obstacle avoidance for UAVs using a fast trajectory planning approach. In: 2019 IEEE international conference on robotics and biomimetics (ROBIO). 2019, p. 1459–64. http://dx.doi.org/10.1109/ROBIO49542.2019.8961790.
[27] Liu S, Watterson M, Tang S, Kumar V. High speed navigation for quadrotors with limited onboard sensing. In: 2016 IEEE international conference on robotics and automation (ICRA). 2016, p. 1484–91. http://dx.doi.org/10.1109/ICRA.2016.7487284.
[28] Phang SK, Lai S, Wang F, Lan M, Chen BM. Systems design and implementation with jerk-optimized trajectory generation for UAV calligraphy. Mechatronics 2015;30:65–75. http://dx.doi.org/10.1016/j.mechatronics.2015.06.006.
[29] Gao F, Wu W, Lin Y, Shen S. Online safe trajectory generation for quadrotors using fast marching method and Bernstein basis polynomial. In: 2018 IEEE international conference on robotics and automation (ICRA). 2018, p. 344–51. http://dx.doi.org/10.1109/ICRA.2018.8462878.
[30] Zhou B, Gao F, Wang L, Liu C, Shen S. Robust and efficient quadrotor trajectory generation for fast autonomous flight. IEEE Robot Autom Lett 2019;4(4):3529–36. http://dx.doi.org/10.1109/LRA.2019.2927938.
[31] Watterson M, Kumar V. Safe receding horizon control for aggressive MAV flight with limited range sensing. In: 2015 IEEE/RSJ international conference on intelligent robots and systems (IROS). 2015, p. 3235–40. http://dx.doi.org/10.1109/IROS.2015.7353826.
[32] van den Berg J, Wilkie D, Guy SJ, Niethammer M, Manocha D. LQG-obstacles: Feedback control with collision avoidance for mobile robots with motion and sensing uncertainty. In: 2012 IEEE international conference on robotics and automation. 2012, p. 346–53. http://dx.doi.org/10.1109/ICRA.2012.6224648.
[33] Zhu R, Sun D, Zhou Z. Integrated design of trajectory planning and control for micro air vehicles. Mechatronics 2007;17(4):245–53. http://dx.doi.org/10.1016/j.mechatronics.2007.01.002.
[34] Chen J, Liu T, Shen S. Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In: 2016 IEEE international conference on robotics and automation (ICRA). 2016, p. 1476–83. http://dx.doi.org/10.1109/ICRA.2016.7487283.
[35] Pivtoraiko M, Mellinger D, Kumar V. Incremental micro-UAV motion replanning for exploring unknown environments. In: 2013 IEEE international conference on robotics and automation. 2013, p. 2452–8. http://dx.doi.org/10.1109/ICRA.2013.6630910.
[36] Oleynikova H, Burri M, Taylor Z, Nieto J, Siegwart R, Galceran E. Continuous-time trajectory optimization for online UAV replanning. In: 2016 IEEE/RSJ international conference on intelligent robots and systems (IROS). 2016, p. 5332–9. http://dx.doi.org/10.1109/IROS.2016.7759784.
[37] Zhang H, Chen H, Wei L. Differentially constrained path planning with graduated state space. Mechatronics 2016;38:132–42. http://dx.doi.org/10.1016/j.mechatronics.2016.02.006.
[38] Zhou X, Wang Z, Ye H, Xu C, Gao F. EGO-planner: An ESDF-free gradient-based local planner for quadrotors. IEEE Robot Autom Lett 2021;6(2):478–85. http://dx.doi.org/10.1109/LRA.2020.3047728.
[39] Burri M, Oleynikova H, Achtelik MW, Siegwart R. Real-time visual-inertial mapping, re-localization and planning onboard MAVs in unknown environments. In: 2015 IEEE/RSJ international conference on intelligent robots and systems (IROS). 2015, p. 1872–8. http://dx.doi.org/10.1109/IROS.2015.7353622.
[40] Bircher A, Kamel M, Alexis K, Oleynikova H, Siegwart R. Receding horizon "next-best-view" planner for 3D exploration. In: 2016 IEEE international conference on robotics and automation (ICRA). 2016, p. 1462–8. http://dx.doi.org/10.1109/ICRA.2016.7487281.



Han Chen received his Bachelor of Engineering in 2016 from Beijing Institute of Technology, P.R.China and Master of Science from Beijing Institute of Technology in 2019. Currently, he is a Ph.D. candidate in MAV/UAV Lab and ARC Lab, Department of Aeronautical and Aviation Engineering, The Hong Kong Polytechnic University.



Shengyang Chen received his Bachelor of Engineering from Northwestern Polytechnical University, P.R.China and Master of Science from University of Siegen, Germany respectively. Currently, He is a Ph.D. candidate in MAV/UAV Lab, Department of Mechanical Engineering, The Hong Kong Polytechnic University.



Chih-Yung Wen received his Bachelor of Science degree from the Department of Mechanical Engineering at the National Taiwan University in 1986 and Master of Science and Ph.D. from the Department of Aeronautics at the California Institute of Technology (Caltech), U.S.A. in 1989 and 1994 respectively. He joined the Department of Mechanical Engineering, The Hong Kong Polytechnic University in 2012, as a professor. In 2019, he became the interim head of the Department of Aeronautical and Aviation Engineering in The Hong Kong Polytechnic University. His current research interests include modeling and control of tail-sitter UAVs, visual-inertial odometry systems for UAVs and AI object detection by UAVs.

Peng Lu obtained his B.Sc. degree in automatic control and M.Sc. degree in nonlinear flight control both from Northwestern Polytechnical University (NPU). He continued his journey on flight control at Delft University of Technology (TU Delft) where he received his Ph.D. degree in 2016. After that, he shifted a bit from flight control and started to explore control for ground/construction robotics at ETH Zurich (ADRL lab) as a Postdoc researcher in 2016. Since 2020 he works in the University of Hong Kong, Department of Mechanical Engineering, as an assistant professor.