AUTONOMOUS FLIGHT IN UNKNOWN AND DYNAMIC ENVIRONMENT

HAN CHEN
PhD

The Hong Kong Polytechnic University

2023

The Hong Kong Polytechnic University

Department of Aeronautical and Aviation Engineering

AUTONOMOUS FLIGHT IN UNKNOWN AND DYNAMIC ENVIRONMENT

CHEN HAN

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Aug 2022

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____(Signed)

Han Chen (Name)

To people who inspire me

ABSTRACT

The development of aerial robots in recent years has involved Micro Aerial Vehicle (MAV) more and more in our daily life. MAVs, especially quadrotors, have been widely used in field applications, such as disaster response, field surveillance, and search-and-rescue. For accomplishing such missions in challenging environments which narrow free space, crowded obstacles, and intruding and moving objects, the capability of navigating with full autonomy without collision is the most crucial requirement. This thesis is arising from the real-world applications of MAVs, and we present novel methodologies, complete system designs, and progressive test results in simulation and field robots, with a focus on the adaptability to cheap and affordable hardware platforms. Also, a basic requirement is finishing all the autonomous navigation task with onboard resources only, no external guidance is required. We start by introducing a computing efficient motion planning algorithm which is more concise and outperform existing autonomous navigation system of UAVs in single-step calculation time. Later, based on the above research, we developed a path planner working on the map which is much larger than the camera FOV to guide the local motion planner and avoid potential detours, confronting a more complex and dense environment. To better avoid those continuously moving obstacles, after that we propose a velocity planning algorithm based on the relative velocities toward obstacles in the environment and implicitly include the future obstacle's position. A novel lightweight pointcloud-based obstacle tracking and velocity estimation algorithm is also designed to fulfill a complete system. At last, we improve the robustness of the dynamic object perception part by introducing the image-based object detector and tracker and active yaw control, enhance the flight smoothness and speed by the polynomial trajectory optimization approach, and upgrade flight safety by involving object state's uncertainty. Extensive experimental and contrasting results, and detailed system set-up are presented throughout the thesis. We conclude this thesis by introducing the motivation backside each chapter and summarizing the current limitation of

the aerial autonomy development, and propose future potential research opportunities based on our research experience.

PUBLICATIONS ARISING FROM THE THESIS

Published:

[1] M. Lu*, **H. Chen***, and P. Lu, "Perception and Avoidance of Multiple Fast Moving Small Objects for Quadrotors with Only Low-cost RGBD Camera", in Robotics and Automation Letters (RAL)[J]. Early access in `https://arclab.hku.hk/files/RA-L%202022_Lu.pdf`. *** equally contributed authors.**

[2] **H. Chen** and P. Lu, "Real-time Identification and Avoidance of Simultaneous Static and Dynamic Obstacles on Point Cloud for UAVs Navigation", in Robotics and Autonomous Systems (RAS)[J], doi: 10.1016/j.robot.2022.104124.

[3] **H. Chen** and P. Lu, "Computationally efficient obstacle avoidance trajectory planner for uavs based on heuristic angular search method." 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 5693-5699, 2020.

[4] **H. Chen**, C. Xiao and P. Lu, "Dynamic Obstacle Avoidance for UAVs Using a Fast Trajectory Planning Approach", In 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO), pp. 1459-1464, 2019. *** equally contributed authors.**

[5] S. Chen, **H. Chen**, C.W. Chang, and C.Y. Wen, "Multilayer Mapping Kit for Autonomous UAV Navigation". in IEEE Access, 9, 31493-31503, 2021.

[6] S. Chen, W. Zhou, A. Yang, **H. Chen**, B. Li, Chih-Yung Wen, "End-to-End UAV Simulation Platform for Visual SLAM and Navigation". in Aerospace[J], 9.2 (2022): 48.

Under process:

[7] **H. Chen**, S. Chen, P. Lu and C. Wen, "A fast planning approach for 3D short trajectory with a parallel framework", under review in Mechatronics[J].

[8] **H. Chen**, C. Wen, and P. Lu, "Uncertainty-Aware and Perception-Enhanced trajectory planning for dynamic environments", under review in IEEE T-MECH[J].

Awards:

The third place in IROS 2019 Autonomous Drone Racing.

ACKNOWLEDGMENTS

Many people helped me a lot to finish this thesis, and I would like to thank them for their vital support and encouragement to me. First of all, I would like to thank my chief supervisor Professor WEN Chih-yung for his strong and selfless support of this research topic and inspiring suggestions. I learned from him a serious and rigorous attitude towards academics. Under his encouragement, I began communicating and cooperating with other colleagues, which made me realize the importance of frequent academic communication. I would also like to thank my co-supervisor, Professor LU Peng. We have frequently communicated and discussed my research progress during the three years, and he gave me detailed and patient guidance on many aspects of my research work.

I would like to thank Prof. CHEN Benmei and Prof. WANG Zhengjie for serving on my Ph.D. defense external committee. I also want to thank my colleagues in our research group for the inspiring technical discussion in robotics and the enthusiastic help in my field tests. Especially, I want to thank Dr. CHEN Shengyang for his detailed answers to my questions in the research and engineering and the energetic discussion of the problems. I sincerely thank Prof. GAO Fei from Zhejiang University for the generous sharing of his ideas, thoughts, and experience in the research of UAV motion planning. I gained huge progress in both research and engineering during the journey in FAST lab.

Last but not least, I thank my parents for their forever strong support and selfless love. Also, I hope to say thank you to my girlfriend Yuyang, who went through the toughest days and nights with me during my pursuit of the degree, and brought me the treasured happy and shining days.

TABLE OF CONTENTS

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

ABBREVIATIONS

AABB	Axis-aligned bounding box
AAC	Autonomous aerial cinematography
API	Application programming interface
BB	Bounding box
DAS	Discrete angular search
DAGS	Discrete angular graph search
DBSCAN	Density-based spatial clustering of applications with noise
EKF	Extended Kalman filter
FFT	Fast Fourier transform
FOV	Field of view
fDSST	fast Discriminative Scale Space Tracking
HAS	Heuristic angular search
IMU	Inertial measurement unit
JPS	Jump point search
KCF	Kernelized correlation filters
KF	Kalman filter
MAV	Micro Aerial Vehicle
MP	Map planner
MOTP	Multiple object tracking precision
MOTA	Multiple object tracking accuracy
PCP	Point cloud planner
PIV	Particle image velocimetry
RMSE	Root mean square error
ROS	Robot operating system

ROI	Region of interest
RRT	Rapidly-exploring random tree
SFC	Safe flight corridor
SLAM	Simultaneous localization and mapping
SOTA	State-of-the-art
UAV	Unmanned Aerial Vehicle
UUV	Unmanned underwater Vehicle
VIO	Visual-Inertial odometry

1. INTRODUCTION

With robotics developing from its birth to the present, an intention always stands unchanged: people hope that robots can help humans undertake some heavy and dangerous work, such as exploring dangerous and unknown environments [1]. Unmanned aerial vehicles (UAVs), especially micro UAVs (MAVs), have become one of the best choices for exploring the unknown environment because of their motion flexibility in space and flight stability. MAVs enjoy a great advantage in price, portability, and environmental adaptability compared to large UAVs, which are more welcome in many application scenarios.

For accomplishing such missions in challenging environments that are usually unknown and chaotic, the capability of navigating with full autonomy while avoiding unexpected obstacles is the most crucial requirement for MAVs in real applications. In addition, MAVs always face rapid unexpected changes, while moving obstacles pose a greater threat than static ones. Avoiding dynamic obstacles with limited onboard sensing and computing with an efficient flight strategy is still an untackled challenge preventing MAVs from being applied in complex real-world tasks.

To tackle this challenge, dynamic environment perception and motion planning methods play the most vital role in the autonomous aerial system. The MAV should sense the obstacle's position, size and motion in time and give reliable results. The motion planner for MAVs needs to constantly and quickly generate collision-free and feasible trajectories in different scenarios, and its response time is required to be as short as possible. In addition, the optimality of the motion strategies should also be considered to save the limited energy of MAVs.

In this thesis, the primary consideration is to develop a complete and effective software system for autonomous navigation in dynamic. We review and introduce our past works in the scope of environment perception and quadrotor motion planning, from the perspective of not only the methodology but also the system design and experimental validation. The

software system is specially designed for those cheap MAV platforms, which are usually equipped with only one depth camera and weak computing power, and the limited sensing and computing ability motivate most algorithms and technical approaches in the thesis.

1.1 Research problems and targets

1.1.1 Dynamic environment perception

Before planning, the trajectory planner needs to obtain information about environmental obstacles to avoid them. In most related studies, obstacles are obtained by using two types of sensors: lidar or depth binocular camera. Lidars are generally large in size and weight and consume too much energy. Although lidars have higher detection accuracy and more stable obstacle information, they are unsuitable for small drones. The detection accuracy of the depth camera within a certain distance (0.5-8m) is sufficient for MAV obstacle avoidance in most cases. However, the field of view is narrow and will likely lose obstacles. They are supposed to be memorized for static obstacles to avoid collision when they are lost in FOV. So we need to use the information obtained by the depth camera to build a map. For dynamic obstacles, the motion estimation should have low latency because the depth camera can only detect close obstacles, and the reaction time required for avoiding close moving obstacles is short. Also, low sensing latency is necessary to catch the dynamic objects flashed by in FOV.

Another important defect of the depth camera is its heavy noise. We need filters to suppress the noise and design a dynamic obstacle tracking and motion estimation algorithm which is robust to the depth noise. The movement of the camera, the narrow FOV, the heavy noise, and the limited computing power make the perception of dynamic obstacles very challenging for autonomous MAV with onboard vision.

1.1.2 Efficiently planning efficient trajectory

The planning algorithm for generating safe trajectory or motion primitives must be very effective in computing. Since we focus on the autonomous flight in an unknown environment, which requires frequent replanning regarding the latest environment information, the computing should be finished in a reasonable time before the vehicle reaches the state which leads to an irreparable collision. Although many research works demonstrate successful planning results in numerical simulation, there lacks the evidence to justify the computing time cost, and they are doubtful to be capable for real-time tasks because the environmental information is usually oversimplified in simulation. The real world contains irregularly shaped obstacles, to precisely describe them, we typically use a pointcloud or voxel map. However, the planning algorithms based on both two approaches require the operation with a large amount of data for collision check. The pointcloud of one single frame from a depth camera usually contains 100000-300000 points, and updating the map still requires iterating each point and doing raycasting, though the collision check with the map is more efficient than checking on a sparse pointcloud directly. Thus, the planning algorithm should be efficient considering the environment information procession.

Under the premise of ensuring safety, planning an efficient trajectory is another essential consideration. For completing the same navigation task, people always hope to reduce costs, such as energy consumption or time consumption. From the view of the flight trajectory's shape, an efficient trajectory should be smooth (indicating the energy is not wasted) and short (no detours, save time). However, finding a more efficient trajectory usually requires more effort in computing, which presents a challenge in algorithm design. For our purpose, the autonomous flight in dynamic environment with one depth camera, the influence from motion planning to dynamic obstacle perception should also be considered. The dynamic object's motion can be estimated more accurate if the tracking duration time be longer, and the position estimation of obstacle is more accurate at a closer distance, while the entire body of obstacle is in FOV. The favorite distance from the vehicle to dynamic obstacle and camera heading direction are also worth considering in motion planning.

1.1.3 System integration

Integrating all the algorithms inside the MAV onboard computer and achieving satisfactory real-world autonomous flight is also worthy of attention. The localization is very important for UAV's navigation. Global Navigation Satellite System (GNSS) is only applicable in the open field, its accuracy is much affected in the crowded urban area, and the satellite signal will be completely blocked in close space. The vision-based localization technique, such as visual-inertial odometry (VIO), has been prevalent in recent years, and it has demonstrated robust performance after being applied to many systems. While lidar can provide better localization accuracy and robustness, its high payload and price are beyond the scope of this thesis. The RGBD camera, micro quadrotor platform, obstacle perception, motion planning algorithm, and VIO algorithm make up our complete autonomous aerial system. The most challenging part is running all the algorithms together on a tiny onboard computer. Each algorithm should meet its real-time standard while sharing very limited computing resources. The code of algorithms is developed with Robot Operating System (ROS) to meet the requirement of parallel and asynchronous computing, and different submodules of the system can be executed at their favorite frequency. During the algorithm and code design, we always put emphasis on streamlining the unnecessary computing or data copy and optimizing each submodule in computing efficiency. After the whole system is integrated, the most cost-effective frequency setting of the sensors and submodules is also worth studying.

1.2 Thesis overview

In the thesis, the author proposes a series of algorithms for both estimating the dynamic obstacles' states and planning the trajectory of the UAV to avoid static and dynamic obstacles. In addition, a complete autonomous flight system is designed based on the proposed algorithms and verified in simulation and real-world experiments, achieving safe navigation in complex dynamic environments. The system is conceived to be practical for real-world applications, and applicable for those micro unmanned platforms with very limited com-

puting resources and cheap sensors. In this thesis, flight safety of the autonomous MAV system has an overwhelming importance level compared to other indicators, such as energy or time cost. With regard to each submodule, flight safety can be decomposed into some more specific indicators to be followed. For example, the motion estimation accuracy of dynamic obstacles, the theoretical guarantee of trajectory safety, and acceptable computation cost for an onboard computer are most important and the consideration is fulfilled in the methodology in this thesis. In fact, optimality in energy or time cost is also contemplated, but under the compromise to computation time cost.

To react to the rapidly changing environment, a vital consideration is utilizing a very lightweight algorithm for local motion planning, thus the time cost for generating collision-free motion primitives can be short enough for successful avoidance. We first propose a very computational effective sampling-based motion planning method in Chapter 3. A feasible sample point in 3D space is selected regarding safety and angular deviation from the direction to the navigation goal. The feasible sample point acts as one constraint in the non-linear optimization problem, the desired control input is solved to control the vehicle through the sample point and meet other kinodynamic limits. The total computation time is only about 15 ms, which outperforms the SOTA algorithms, and our method can avoid those intruding obstacles in real-world experiments.

However, to autonomously fly in complex terrain, global path planning on a map is important. Local planners may often fall into "traps" in the environment, such as a corridor with a dead end, because only a narrow local part of environmental information is utilized for planning. Therefore, in Chapter 4, we propose a map planner to find a sub-optimal path in the large 3D map, combining it with the fast reactive local motion planner. The Fermat point of a manually designed triangle is the local goal of the local motion planner introduced above, smoothing the path and connecting the two planners. Since the local planner directly works on point cloud which is of low latency, the system can react to the intruding obstacles agilely, while resulting in a short trajectory in a global view.

Although a fast algorithm can help the drone react to environmental change and intruding obstacles, it is not enough. If an object moves continuously at a comparable speed to the

UAV, the above local planner will fail because obstacles are still considered static in the planner. At least the velocity of dynamic obstacles should be considered while planning vehicle motion, and the collision can be avoided in advance. To safely avoid dynamic obstacles, as well as save energy, in Chapter 5 we propose a velocity planning algorithm based on the relative velocity between the vehicle and obstacles. The velocity is chosen from samples, and it results in the relative velocities that are safe for each obstacle and is of the minimum acceleration (energy) cost. The velocity of a dynamic object is estimated by calculating the object's displacement from two pointcloud frames with a certain time gap. All computation is conducted online on the tiny onboard computer, and the method is tested successfully in field experiments.

At last, in Chapter 6 we upgrade the autonomous fight system from each aspect to further improve the dynamic object perception robustness, trajectory optimality, and flight safety. In the original practice, we first estimate the velocity of all point clusters (objects), but many static objects are often wrongly assigned a non-zero speed by the perception algorithm because of the horrible depth noise from the RGBD camera, which will disturb the planner a lot. So we first utilize an image-based object detector and classifier to find objects that tend to be dynamic, and the point clusters are matched with the ROIs of dynamic objects on the image. The velocity of objects is estimated by an improved and expanded PIV method, which can address the wrong cluster match between two point cloud frames. In addition, the trajectory planning is based on optimizing segmented polynomials and MINCO class, which can result in a spatial-temporal optimal trajectory for avoiding both static and dynamic obstacles. The acceleration uncertainty and object visibility are also considered in trajectory planning.

In each chapter, we demonstrate the flight performance in both Gazebo simulation which is very close to reality, and the real-world tests with our self-assembled MAV platform. At last, the insufficiency of our proposed methods and the possible technical approaches for improvement in this research field is discussed in Chapter 7.

Mathematical symbol announcement

For the reader's convenience, the mathematical symbols in this thesis are defined independently in each chapter when they appear at the first time in this chapter, since we introduce a complete system in each chapter. To fit the algorithms used in each chapter, the symbols may be different in those chapters to represent a very similar concept, and please refer to the definition in its current chapter.

2. LITERATURE REVIEW

2.1 Dynamic and static obstacle perception

2.1.1 Mapping the environment

It is required for hardware experiments to encode and utilize the information of identified static impediments efficiently. In the majority of relevant research, point clouds are the most commonly utilized format for expressing obstacle information. One strategy for employing point clouds is just to utilize the most current sensor measurement data or to weight the most recent data [2, 3]. In other words, these methods will not record obstacles that have moved away from the camera's field of view (FOV) [4, 5]. The alternative, which is also the most typical practice, is to continually fuse a filtered point cloud into a map, generally in the form of an occupied grid or distance field [6], and then do trajectory planning on the basis of the grid map. When considering vehicle state estimation, many methods have been proposed to convert the depth measurements generated by the onboard sensors into a global map. Representative methods include voxel grids [7], Octomap [8], and elevation maps [9]. Each method has advantages and disadvantages in a particular environment. The voxel grid is suitable for fine-grained representation of small volumes, but the storage complexity is poor. Elevations are suitable for representing artificial structures composed of vertical walls but are less efficient in describing natural and unstructured scenes. Octomap is memory-efficient when indicating an environment with a large open space. This storage structure is beneficial for further utilizing maps for trajectory planning and has the function of automatic map maintenance, which is convenient to use and has satisfactory results in both simulation and hardware flight tests.

In the previous work [10], they use Octomap building on point cloud raw data to develop their own method and gained satisfactory experimental results. In another way, in

order to reduce the computational time consumed by this step of building the map, some researchers have directly planned the trajectory on the original point cloud. Lopez et al. utilize the transformed point cloud for the collision check with trajectories corresponding to the randomly generated motion primitives [11]. However, directly planning on the point cloud requires high-quality point cloud information. This method is not suitable for drones carrying a single depth camera if a global map has not been established.

2.1.2 Dynamic obstacle perception

Most researchers employ the camera's raw image and mark the corresponding pixels before measuring the depth to identify and track moving obstacles from the environment. The semantic segmentation network with a moving consistency check method based on images can distinguish the dynamic objects [12]. Also, a block-based motion estimation method to identify the moving obstacle is used in [13], but the result is poor if the background is complex. Some work [14]-[15] segment the depth image and regard the points with similar depth belong to one object. But such methods cannot present the dynamic environment accurately because static and dynamic obstacles are not classified. If only humans were considered as moving obstacles, the human face recognition technology could be applied [16]. However, the abovementioned works do not estimate the obstacle velocity and position. A multi-purpose approach is proposed in [17], which jointly estimates the camera position, stereo depth, and object detections, and tracks the trajectories. Some works adopt feature-based vision systems to detect dynamic objects [18]-[19], which require dense feature points. Also, detector-based or segmentation-network-based methods can work well in predefined classes such as pedestrians or cars [20]. However, they cannot handle generic environments. A similar dynamic obstacle perception approach to ours is proposed in [21], but the tracking by detection method they used is still computationally expensive for micro onboard computers, and the Intersection over Union (IoU) matching metric is not good for dense scenes. Considering the onboard microcomputer's limited resource, the above image-based methods are computation-expensive and thus unable to run in real-time.

Based on the point cloud data, it is also possible to estimate the moving obstacles' position and velocity in the self-driving cars [22]-[23]. However, they all rely on high-quality point clouds from LiDAR sensors and powerful GPUs to detect obstacles from only predefined classes. For enhancing the versatility, tracking point clusters with the global feature has been proved as a practical idea in austere environments [24]. As for the point cloud of a depth camera, the existing works are rare and they all match the obstacle by only the center position of obstacles, depending on the Kalman filter to predict the status of dynamic obstacles from past to present. However, this may fail when the predicted position of one obstacle is close to other obstacles at present. Varying from them, we propose the feature vector for each obstacle to tackle this challenge, and the matching robustness and accuracy are improved a lot. Our method also can be run at a higher frequency with low computational power. Event cameras can distinguish between static and dynamic objects and enable the drone to avoid the dynamic ones in a very short time [25]. However, the event camera is expensive for low-cost UAVs, and the high-quality depth information of the obstacle may also rely on another depth camera because the generated events are sparse [26].

2.2 Fast and feasible motion planning methods

After obtaining the environmental information, we can generate motion primitives and perform the collision check with the obstacle's information. Here we would like to mention several most classical algorithms in motion planning in robotics. The artificial potential field (APF) method [27] conceives that the goal point generates an "attractive force" on the vehicle, and the obstacle exerts a "repulsive force". The movement of the vehicle is controlled by seeking the resultant force. Its expression is concise, but it is easy to fall into the optimal local solution. The vector field histogram (VFH) [28] is a classical algorithm for robot navigation with a lidar, improved from the APF method. VFH will calculate the travel cost in each direction. The more obstacles in this direction, the higher the cost. The dynamic window approach (DWA) is a sampling-based method that samples the motion primitives

within the feasible space and chooses one set by ranking them with a cost function [29]. The concepts in these classical methods are of the excellent reference value and enlightening significance for our algorithm. However, for a UAV with a single depth camera flying in an unknown environment, they are not sufficient.

The related methods that appeared in recent years can be divided into two categories. One is the hierarchical approach, composed of a front-end path planner and a back-end trajectory or motion primitive planner. The path planner usually works in lower-dimensional state space ($SO(3)$ for UAV), and aims to robustly find a viable path in a short time. Then, the back-end plans the vehicle's states in higher order, such as velocity, and acceleration, to generate a smooth trajectory while respecting the vehicle's kinodynamic limits. The path offers geometric constraints for the back-end. Another way is planning in the entire state space and generating motion primitives, and usually, it is based on sampling.

2.2.1 Path planning

Before introducing the hierarchical approach, we should study the front-end path planning algorithms. 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 [30], A* [31], anytime repairing A* [32], JPS [33], and hybrid A* [34]. Dijkstra's algorithm is the root of the above methods, which searches paths 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 optimality in all 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 a random tree (RRT) [35],

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. The RRT algorithm can effectively find a viable path, but it has no asymptotic optimality, and its search will stay at the first feasible solution. Sampling-based methods with asymptotic optimality include probabilistic road maps (PRM*) [36], rapid exploration of random graphs (RRG) [37] and RRT* [37], where RRT* can make the solution converge to the global best point with the increase of samples. RRG is an extension of the RRT algorithm because it connects the new sample with all other nodes within a specific range and searches for the path after constructing the graph. Based on RRT, the method in [38] cancels the optimal control of time to ensure the asymptotic optimality of the path and kinematics feasibility. Also, a belief roadmap can be combined with RRG [39] to solve the problem of trajectory planning under the state uncertainty. A technique called "partial ordering" balances confidence and distance to complete the expansion graph in the confidence space.

2.2.2 Hierarchical approach

The hierarchical approach first converts the obstacle information and the position of the UAV in three-dimensional space into a local map. This map contains only the obstacle information near the UAV, the global goal and the points of obstacles are projected in this local map in some way. A static path planning algorithm is run on the local map, and the motion primitives are obtained by solving the motion planning equations. For instance, [40]-[41] built a local occupancy grid map with the most recent perception data and generated a minimum-jerk trajectory through waypoints from an A* search.

In addition, you can also obtain motion primitives by solving an optimization problem with other kinds of geometrical constraints except for the waypoint. Thanks to the convex property of the Bezier curve, we can ensure that the final trajectory is collision-free by setting constraints to the control points only, which is very effective in computing. The safety corridor (SFC) is one popular technique for giving geometrical constraints in trajectory optimization. It is composed of a set of simply-connected polyhedrons, the polyhedrons

connect end to end and cover the front-end path completely. The most crucial benefit of SFC is the simplification of collision checks. We only need to check a position with several polyhedrons instead of a large number of points in pointcloud. Mellinger et al.[42] first systematically propose using corridor constraints to make the aircraft's trajectory confined to an interval of space, but they did not give a method to construct SFC. Deits et al.[43–45] use a series of connected convex polyhedra to represent feasible spaces, describe the linear inequality constraints restricting polynomial trajectories in polyhedra as a set of polynomial sum-of-squares conditions, and use mixed integer second-order conic programming to solve the distribution of each segment of the trajectory in different polyhedrons. [46] achieved satisfactory results by utilizing SFC, the segmented Bezier splines are guaranteed safe by protecting the control points inside the SFC. A similar approach can be fount in [47]. However, SFC is not necessary, the collision check of control points can also be achieved with a voxel map by querying in a compact data structure [48].

As for time allocation for the segments of curves, an approximate method was used in [47, 49], and a bi-level optimization was used in [50]-[51] to find the optimal time allocation.

2.2.3 Direct approach

The other method is skipping the map's search paths and directly generating motion primitives by sampling. Then, the evaluation function can be designed to select the most suitable group of motion primitives as the output, which is very similar to DWA. Mueller et al. present a representative work, even making the quadrotor catch a falling ball [52]. An effective algorithm for state space sampling utilizing a model-based trajectory generation approach is proposed in [53], demonstrating the potential of the state space sampling-based method to adapt to real-time motion planning in a complex environment. One of the great advantages of the sampling method is that it does not explicitly construct the exact boundary of the feasible state space, but obtains discrete state points by sampling in the continuous state space and connects them through the graph structure or tree structure (the tree structure can also be regarded as a special graph structure) in order to explore

the connectivity of the feasible space efficiently. Therefore, it is mainly used to solve the problem complexity caused by obstacle constraints. The most influential algorithm in the state space sampling method should be kinodynamic-RRT* [54], which generates a complete trajectory in the state space and has the asymptotic global optimality. In recent years, many variants of kinodynamic-RRT* have been proposed to improve its computation efficiency and be capable for real-time planning. Gammell et al.[55] proposed informed RRT*, which proved that when there is an initial solution, the sampling that can improve the quality of the solution is constrained in a high-dimensional ellipsoid, so direct sampling can be carried out. Compared with sample rejection after uniform sampling, it greatly improves the sampling efficiency, especially in high-dimensional space. Later, he proposed bit*[56], ait*[57], Eit*[58], and other algorithms together with Sturb et al., which also use this feature for adaptive sampling.

2.3 Global and local planning

In some environments where existing huge obstacles larger than the vehicle's maximum sensing range or the local planner's horizon, the local environment information is no longer enough for navigation. A planner working on a larger map will be helpful to guide the local planner and avoid falling into local minima and flying blindly. In the last few years, several research works discussed how to combine the optimal global planning algorithm for static maps with the algorithm applied to real-time online replanning. The algorithms can learn from each other's strengths and weaknesses, i.e., working out a short path and responding quickly to map changes for replanning the trajectory. 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 [59, 60], and the path is continuously adjusted as the obstacle information is updated. We call it the optimistic planner. In [51, 61], the optimistic global planner and conservative local planner are combined to ensure the safety of the aircraft. Tordesillas et al. [62] proposed a planning framework with multi-fidelity models to diminish the inconsistency between the global and local planners. They run the JPS

algorithm on the local slide grid map, and the constraints of motion optimization were divided into three parts according to the distance from the drone, where the most strict constraints are for the closest area. With the large map and the navigation task request moving a long distance, the sampling-based path searching methods (such as RRT) usually find a feasible path faster than the graph-searching ones (A* or JPS). However, the resulted path may be much longer than the optimal global solution. RRT* [37] with a maximal time cost bound can be used in real-time planning and improve the path optimality, but there is still much uncertainty in optimality. RRTX[63] should be a very effective solution for global path fast replanning. It can replan the near-optimal path very fast when needed because it refines and repairs the same search graph over the entire duration of navigation, instead of regrowing the entire search tree. In addition, the authors demonstrate the fast replanning in a dynamic environment, which indicate the broad prospects in UAVs' application.

2.4 Planning safely in dynamic scenes

Several published works of autonomous UAV navigation systems have demonstrated agile, reliable flights in a clustered static environment, and the motion primitives can be solved in real-time [48, 64, 65]. In terms of the avoidance of moving obstacles for navigation tasks, the majority of research works are based on the applications of ground vehicles. The avoidance between multiple agents in the robot swarm is a similar problem, and the related research appears earlier than the dynamic obstacle avoidance problem. The relative velocity between agents (or between the local machine and other dynamic obstacles) is an important clue, and Fiorini et al. first realized this and proposed the velocity Obstacle (VO) algorithm [66]. The VO method assumes all other agents will not actively avoid the collision, and their velocities are always constant. Van den burg et al. solve the shaking problem and propose the reciprocal velocity obstacles (RVO) method in multiple agents navigation when two agents actively avoid each other using the same VO method [67]. The forbidden velocity map [68] is designed to solve all the forbidden 2D velocity vectors, and they are represented as two separate areas in the map, thus the optimal velocity with minimum acceleration cost

for avoidance can be solved. Alonso et al. [69] further address the shaking problem when multiple agents actively avoid each other, and the optimal velocity can be solved by linear programming (LP) with a very light computation.

The artificial potential field (APF) method can avoid the moving obstacles by considering their moving directions [70]-[71]. For UAVs, the model predictive control (MPC) method is tested to be one feasible solution, but the time cost is too large for real-time flight [72]. The probabilistic safety barrier certificates (PrSBC) define the space of admissible control actions that are probabilistically safe, which is more compatible for multi-robot systems [73]. Recently, some global planners for UAV navigation in crowded dynamic environment are proposed [74]-[75]. However, the states of all obstacles are known, they are not suitable for a fully autonomous aerial platform. [76] utilizes the kinodynamic A* algorithm to find a feasible initial trajectory first and the parameterized B-spline is used to optimize the trajectory from the gradient. However, it does not optimize the flight time and does not consider the inaccuracy of the constant velocity model while ours does.

The narrow FOV of a single depth camera is the stumbling block for the autonomous vison-based navigation systems to perceive the environment. To save cost, reduce the weight, and lighten the computation burden for the onboard computer, most researchers choose to fix only one camera on the drone frame as the environmental sensor [48, 64]. A reasonable solution to make up for this defect is adjusting the vehicle trajectory while planning the yaw to sense unknown space during the flight [77]. Later in [78], the object visible capability of the planner is enhanced with the star-convex constrained optimization, obtaining a spatial-temporal optimal trajectory. However, such approaches are all for the static environment, we propose a different method to plan the yaw to enable the camera to track dynamic objects more robustly. With the help of a mechanical gimbal, Chen et al. [79] design an active-sensing-based UAV for obstacle avoidance. Instead of fusing multiple sensors to enlarge the narrow FOV of a single depth camera, they mount the camera on the gimbal and plan its heading direction while planning the trajectory of the vehicle to get better sensing results of dynamic obstacles or unexplored areas. However, this increases

the mechanical complexity and brings extra load to the propulsion system, and we seek an intelligent planning algorithm for those simple hardware platforms.

3. EFFICIENT SAMPLING-BASED MOTION PLANNING

3.1 Research Background

For autonomous flight in an unknown or even dynamic environment, we believe two indicators are crucial for the motion planning algorithm. First, the future trajectory of the vehicle should be guaranteed safe (or collision-free) from at least a theoretical view. Second, the computation of the motion primitives should be very light so that even on tiny onboard computers, the motion replanning can be finished in a very short time (a few milliseconds) to react to intruding obstacles or sudden environmental change. Responding quickly enough to the newly detected static obstacles, or even moving obstacles also ensures the vehicle safety from another aspect. Suppose the planning algorithm is too complex and always takes a long time when it is executed on board. In that case, collisions will occur when the UAV is flying in an unknown and clustered environment. The updated environmental information may contain a new obstacle close to the vehicle and occupies the originally planned trajectory, and the complex algorithm may fail to solve the motion plan before the collision happens. In addition, for accomplishing a variety of missions in challenging environments, the capability of navigation to a user-defined goal is required. The goal should be considered in the motion planning, so that the goal could be approached with the planned trajectory.

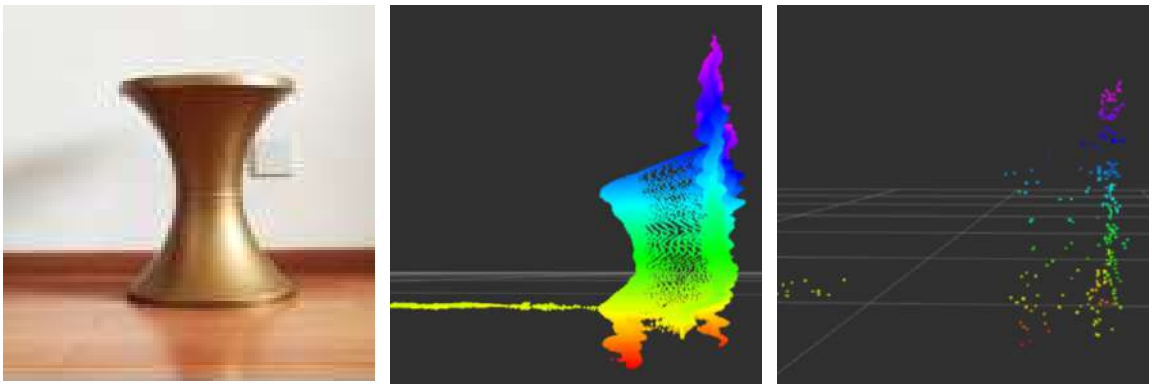
Because of the narrow view field of single depth camera on a UAV, the information of obstacles around is quite limited thus the shortest whole actual flight trajectory is difficult to achieve. Therefore we focus on the time cost of the trajectory planner and safety rather than other factors such as trajectory length or entire energy cost.

In this chapter, we proposed such a computationally efficient obstacle avoidance trajectory planner that can be used in unknown cluttered environments. This planner is mainly composed of a point cloud processor, a waypoint planner with Heuristic Angular Search(HAS) method and a motion planner with minimum acceleration optimization. Fur-

thermore, we propose several techniques to enhance safety by making the possibility of finding a feasible trajectory as large as possible. The proposed approach is implemented to run onboard in real-time and is tested extensively in simulation and the average control output calculating time of iteration steps is less than 18 ms.

In the waypoint planner, we directly find the target point of the drone in the next step on a sparse point cloud. Then in the motion planner, we solve the optimization problem to obtain the motion primitives that the drone needs to perform at the next moment. In order to reduce the amount of calculation for collision detection when searching for a waypoint, we further streamline the point cloud of obstacles in the global map maintained by Octomap. The degree of simplification is related to the drone safety radius r_{safe} we set. Then, the discrete angular search is used to simplify the collision detection to calculate the distance from the point to the straight line.

3.2 Quick Responding and Safe Planner



(a) (b) (c)

Fig. 3.1.: (a) depth camera's RGB output, (b) raw point cloud, (c) filtered point cloud(Pcl_2)

As mentioned above, the collision check is the most time-consuming part of the trajectory generation. To cope with this challenge, we introduce a Heuristic Angular Search(HAS) method with a backup safety plan. The overall algorithm is presented in Algorithm 1, where Pcl is the point cloud, P_{rec} is the list where the planner record p_n in each step, B_E described

in (3.1) is the transformation matrix from body coordinate to earth coordinate, $c()$ is short for $cos()$ and $s()$ is short for $sin()$, ϕ, θ, ψ are Euler angles respectively. We describe Line 2-4 in section 3.2.1 and describe Line 5 in Section 3.2.2. Line 6-7 is described in Section 3.2.3 and Line 9 is described in Section 3.2.4. Overall, the outer loop can be executed at 55-100 Hz, considering the density of obstacles in the simulation tests.

Algorithm 1: our proposed planner

1: **while** *true* : **do**
2: Filter the raw point cloud data, output Pcl_1
3: Transform Pcl_1 in body coordinate (B) to Pcl_2 in earth coordinate(E) by B_E
4: Build a global map represented by point cloud Pcl_3, filter again
5: Find the next waypoint w_p by heuristic angular search
6: **if found a feasible waypoint: then**
7: Run the minimum acceleration motion planner to get motion primitives
8: **else**
9: Run the backup plan for safety, then go to line 5
10: Send the motion primitives to the UAV flight controller
11: Record the current position p_n in list P_{rec}

$$B_E = \begin{bmatrix} \mathrm{c}\psi\mathrm{c}\theta & \mathrm{s}\psi\mathrm{c}\theta & -\mathrm{s}\theta \\ \mathrm{c}\psi\mathrm{s}\theta\mathrm{s}\phi - \mathrm{s}\psi\mathrm{c}\phi & \mathrm{s}\psi\mathrm{s}\theta\mathrm{s}\phi + \mathrm{c}\psi\mathrm{c}\phi & \mathrm{c}\theta\mathrm{s}\phi \\ \mathrm{c}\psi\mathrm{s}\theta\mathrm{c}\phi + \mathrm{s}\psi\mathrm{s}\phi & \mathrm{s}\psi\mathrm{s}\theta\mathrm{c}\phi - \mathrm{c}\psi\mathrm{s}\phi & \mathrm{c}\theta\mathrm{c}\phi \end{bmatrix} \tag{3.1}$$

3.2.1 Processing the Point Cloud

The point cloud data obtained by a real depth camera is often noisy and too dense, and the noise is greater on objects farther from the camera, as shown in Fig. 3.1(a) and Fig. 3.1(b). This is inconvenient for converting the coordinate system of each point in the point cloud and establishing a global map. First, we filter the original point cloud data Pcl_1 through three filters in order to obtain the point cloud data Pcl_2 which is convenient to store and recall. The algorithm of the filter and the point cloud after filtering are shown in Algorithm 2 and Fig. 3.1(c). d_{use} is a parameter. It can be seen that the filtered point cloud

data are more concise and tidy, retaining the basic shape of the obstacle. Then we convert the point cloud into the earth coordinate system and use Octomap to build and maintain a global map. In fact, it is tolerable as long as the gap between the midpoints of the point cloud corresponding to an obstacle is not greater than the safe radius r_{safe} of the drone. But if you do this at beginning, the global map after fusion will be unavailable for visualization. So we filter again after we obtain the point cloud of the global map, the algorithm is also shown in Algorithm 2. q is one of the three axes' value of a point in Pcl_4, p^w is the point in Pcl_4, $L_q \in \mathbb{R}^{n\times 3}$ is the list of all points in Pcl_4 and it is rearranged by the increasing q value according to the order of $x-y-z$.

At last, we only use the point in Pcl_r for collision detection.

Algorithm 2: point cloud filter

$Pcl_1 \Leftarrow$ point cloud raw data
for p^i in Pcl_1 **do**
    Remove p^i which is further than 8m, keep only one point in a 0.2m voxel, remove the outliers
$Pcl_2 \Leftarrow Pcl_1$
for p^m in Pcl_2 **do**
    $p^m = B_E p^m + p_n$
$Pcl_3 \Leftarrow$ center points of Octomap, with Pcl_2 input
$Pcl_4 \Leftarrow Pcl_3$
for q in x, y, z **do**
    for p_q^w in L_q **do**
        if not $((p_q^w - L_q(0,q))\%r_{safe} \approx 0$ **or** no element of $L_q(:,q)$ in range of $[p_q^w, p_q^w + r_{safe}])$ **then**
            Delete p^w from Pcl_4
$Pcl_r \Leftarrow Pcl_4$
for p^t in Pcl_r **do**
    if $\|\overrightarrow{p_n p^t}\|_2 > d_{use}$ **then**
        Delete p^t from Pcl_r
$d_{min} = min(\|\overrightarrow{p_n p^t}\|_2)$

3.2.2 Heuristic Angular Search Method

Different from the previous work in which a complete path needs to be planned on the local map, we only find a target point close to the drone as a guide for motion planning. Because the overall planner's calculation speed is quite fast, such a short predicted trajectory is sufficient to refresh before the drone flight reaches its endpoint. As shown in Fig. 3.2, we use the vector $A_g = (\alpha_g, \beta_g)$ to represent the angle of the navigation target p_g relative to the current position of the drone p_n in E. Based on this, we define a series of line segments with different endpoints P_{d1}-P_{d4} in (3.2), and these line segments have a common endpoint p_n.

Algorithm 3: HAS method

for i in $\{1,2,3,4\}$ **do**
  for α_d in $\{0, \Delta\alpha, 2\Delta\alpha ... m\Delta\alpha\}$ **do**
    if $d_{Pdi} > r_{safe}$ **then**
      $w_p = \mu(P_{di} - p_n) + p_n$
      Break all circle

Algorithm 4: collision check

for p^t in Pcl_r **do**
  if $\|\overrightarrow{p_n p^t}\|_2^2 > \|\overrightarrow{P_{di} p^t}\|_2^2 + \|\overrightarrow{P_{di} p_n}\|_2^2$
or $\|\overrightarrow{P_{di} p^t}\|_2^2 > \|\overrightarrow{p_n p^t}\|_2^2 + \|\overrightarrow{P_{di} p_n}\|_2^2$ **then**
    $d_{Pdi} = \infty$ (Foot drop of p^t is not on $\overline{p_n P_{di}}$)
  else
    $d_{Pdi} = \frac{\|\overrightarrow{p_n p^t} \times \overrightarrow{p_n P_{di}}\|_2}{\left\|\overrightarrow{p_n P_{di}}\right\|_2}$

Algorithm 3 reveals the process of searching for waypoints. We simplify the calculation of collision detection by calculating the perpendicular distance from a point to a line segment, rather than the distance from the obstacle to the sampled curvy path [80]. The specific process of collision checking is shown in Algorithm 4. In most cases in simulation tests, collision detection can be done within 16 ms. The meaning of heuristic search is that the starting point of the search is calculated according to the historical record of the results obtained by this method and the current point cloud information, so as try to obtain an initial

value A_{g0} which is the closest to the final search result, minimize the search time cost. The initial value of A_{g0} is calculated in (3.3) and (3.4), where l_d is the detection radius of UAV for obstacle avoidance check, μ is a relatively small coefficient with a value between 0.1-0.2. A_{last} is the angle corresponding to the waypoint in the last step, n_{obs} is the size of Pcl_r in the current step and n_{avr} is the average of the size of Pcl_r over all past steps.

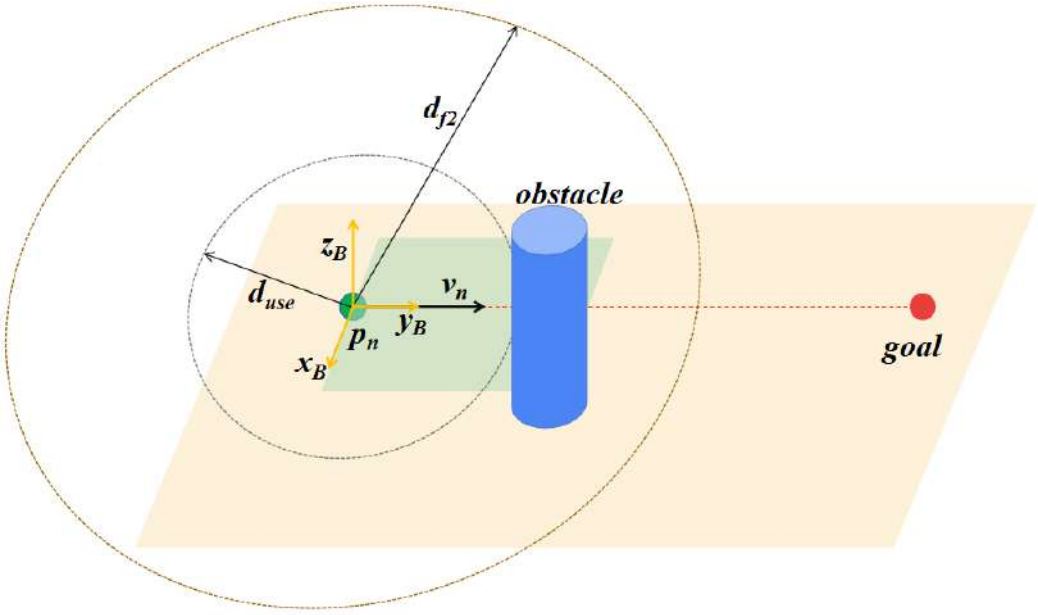
$$\begin{aligned}
P_{d1} &= l_d \left(\mathrm{c}\left(\alpha_{g0}+\alpha_d\right), \mathrm{s}\left(\alpha_{g0}+\alpha_d\right), \mathrm{s}\left(\beta_{g0}\right)\right) + p_n \\
P_{d2} &= l_d \left(\mathrm{c}\left(\alpha_{g0}-\alpha_d\right), \mathrm{s}\left(\alpha_{g0}-\alpha_d\right), \mathrm{s}\left(\beta_{g0}\right)\right) + p_n \\
P_{d3} &= l_d \left(\mathrm{c}\left(\alpha_{g0}\right), \mathrm{s}\left(\alpha_{g0}\right), \mathrm{s}\left(\beta_{g0}+\alpha_d\right)\right) + p_n \\
P_{d4} &= l_d \left(\mathrm{c}\left(\alpha_{g0}\right), \mathrm{s}\left(\alpha_{g0}\right), \mathrm{s}\left(\beta_{g0}-\alpha_d\right)\right) + p_n
\end{aligned} \tag{3.2}$$

$$A_{g0} = \begin{cases} A_g \text{ (others)} \\ A_{\text{last}} \left(\lambda n_{\text{obs}} > n_{avr}\right) \end{cases} \tag{3.3}$$

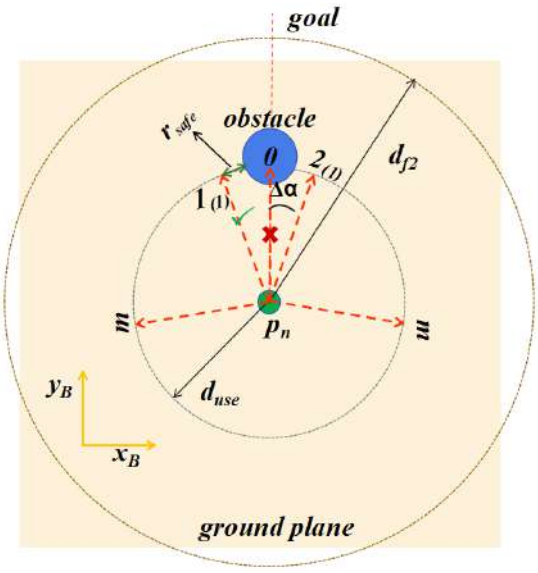
$$\lambda = \frac{\text{times' number for } A_g = A_{last} \text{ in last 3 steps}}{3} \tag{3.4}$$

3.2.3 Motion Planning

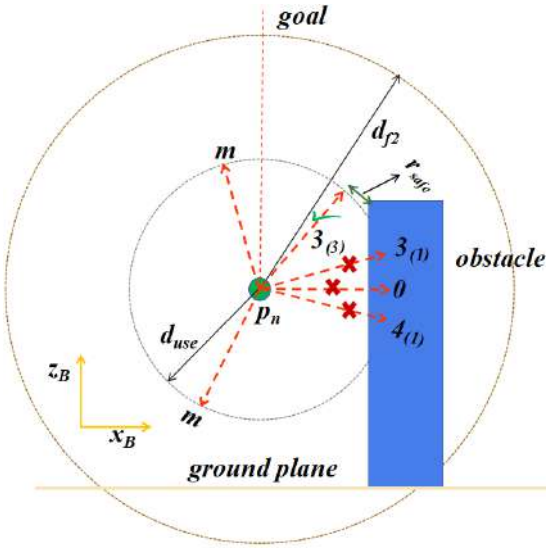
After obtaining the next waypoint w_p, the next step is to calculate the motion primitives (such as position $p = [p_x, p_y, p_z]$, speed $v = [v_x, v_y, v_z]$, acceleration $a = [a_x, a_y, a_z]$) and send the command to the flight controller. Sending w_p directly as the control command may cause the flight to be unstable, and the acceleration magnitude may exceed a_{max}. Because w_p 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. In this way, the kinematic constraints of the drone can be addressed by setting constraints [38]. We take the acceleration of the drone as the variable to be solved, because compared with the use of jerk or snap, acceleration can be



(a)



(b)



(c)

Fig. 3.2.: Illustration about angular search,(a) is a stereogram,(b) and (c) are the projection of (a) to different plane. $B-xyz$ presents the body coordinate. The number in the blanket is the ordinal number of iteration, for example, $2_{(1)}$ presents P_{d2} with $\alpha_d = \Delta\alpha$.

directly sent to the flight control as a control command, and the computational load is less while meeting the kinematic constraints and ensuring the smooth trajectory.

The optimization problem is defined in (3.5), where the subscript n presents the current step in a rolling process of the whole planner, p_{start} is the position of the drone when the planner starts to work [81]. v_{max} and a_{max} are the kinematic constraints for speed and acceleration respectively, t_{max} is the upper bound for the time which can be used to finish the predicted piece of trajectory. ξ is the tolerance for the difference between the end of the predicted trajectory and the w_p, v_{n+1} and p_{n+1} are calculated by the kinematic formula.

$$
\begin{aligned}
\min_{a_n, t_n} \quad & \|a_n\|_2^2 + \eta t_n \\
\text{s.t.} \quad & p_0 = p_{\text{start}} \\
& 0 < \mathrm{t}_n \leq t_{max} \\
& v_n = \dot{p}_n \\
& a_n = \dot{v}_n \\
& \|v_{n+1}\|_\infty \leq v_{max} \\
& \|a_n\|_\infty \leq a_{max} \\
& \|p_{n+1} - w_p\|_2 \leq \xi \\
& v_{n+1} = v_n + a_n t_n \\
& p_{n+1} = p_n + v_n t_n + \frac{1}{2} a_n t_n^2
\end{aligned}
\tag{3.5}
$$

3.2.4 Safety Guarantee

Next, we demonstrate the safety of the trajectory and add additional measures to improve safety based on the above method. As shown in Fig. 3.3, if the trajectory of the aircraft is a straight line that coincides with $\overline{p_n w_p}$ in each step, then this line must be safe because it has undergone collision detection check. However, considering the kinematic constraints of the aircraft, the trajectory of the aircraft in each step is a curve. Assuming that the acceleration a_n solved by the optimizer is in the same plane as the speed v_n and the waypoint of the

drone at the current moment (so that it meets the optimization objective function), then this curve is a parabola in this plane. When a_n is in opposite direction of v_n, the deviation d_{max} between the drone trajectory and line segment $\overline{p_n w_p}$ is the largest. It can be easily proven since $\|v_n\|_2$ and $\|a_n\|_2$ is constant.

We can get d_{max} by solving the optimization problem in (3.6), and we get a close form solution in (3.7).

$$\begin{aligned} d_{\max} &= \max\left(\frac{2\|v_n\|_2^2}{\|a_n\|_2}\right) \\ s.t.\ &\sqrt{\frac{2\left\|w_p - p_n\right\|_2}{\|a_n\|_2}} + \frac{\|v_n\|_2}{\|a_n\|_2} \leq t_{\max} \end{aligned} \tag{3.6}$$

$$d_{\max} = 2\|v_n\|_2\left(t_{\max} - \sqrt{\frac{2\left\|w_p - p_n\right\|_2}{a_{\max}}}\right) \tag{3.7}$$

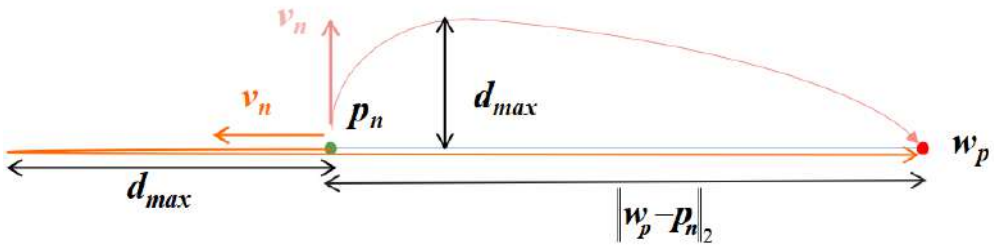


Fig. 3.3.: Illustration of the relationship between d_{max} and the direction of v_n ($\|v_n\|_2$ and $\|a_n\|_2$ are fixed).

The trajectory is safe when we choose parameters to make $d_{max} < r_{safe}$.

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}), HAS will 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 can be proved always smaller than r_{safe}. 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 planning step, and the chosen search direction of the former step will not be considered after the drone has flown back in place. This measure is called the "safety backup plan."

Besides, during the flight, when no feasible w_p is found at the first run of Algorithm 3 we will change l_d to a smaller one and try it again.

3.2.5 Improvements on the Motion Planner

(1) Streamline and Sequence the Input Pointcloud

In our previous work[82], we find that the execution time of the trajectory planner is highly relevant to the size of the input point cloud Pcl_r. The collision check accounts for a large proportion of the total time cost. Every point in Pcl_r is checked for a collision in the loops. Once a point p_{1st} in Pcl_r is first found to collide with the detecting line segment, the collision check loops are terminated. Therefore, we hope to find p_{1st} in a more computation-efficient way to reduce the time cost.

Algorithm 5: Streamline the sorted Pcl_r

for p^t in Pcl_r (j is the iteration number): **do**
  if ($\|p^t - p_n\|_2 \leq 0.5d_{ft}$ or $\angle p_{1st}p_np_g \leq 90^\circ$): **then**
    Put p^t in list Pcl_{use}
if $len(Pcl_{use}) > n_{use}$: **then**
  Remove $len(Pcl_{use}) - n_{use}$ points in Pcl_{use} randomly
else
  Choose $n_{use} - len(Pcl_{use})$ points from $\complement_{Pcl_r}Pcl_{use}$ randomly, add them into Pcl_{use}

By analyzing a large amount of recorded data of the motion planner in simulation and hardware tests, we found the following statistical laws. Let p_g denotes the goal position. p_{1st} has a larger probability to appear in the part that is closer to p_n and $\overline{p_np_g}$ (Pcl_r is first sorted in order of increasing distance to p_n). d_{ft} is the farthest distance from p_n to the points in Pcl_r. For approximately 89.6% of all the recorded p_{1st}, $\overline{p_np_{1st}} \leq 0.5d_{ft}$. For approximately 81.3% of p_{1st}, the angle $\angle p_{1st}p_np_g \leq 90^\circ$. Thus, the part of Pcl_r that is out of the highlight range ($\overline{p_np_{1st}} \leq 0.5d_{ft}$, and $\angle p_{1st}p_np_g \leq 90^\circ$) can be streamlined. In

Algorithm 5, we streamline Pcl_r to remain within at most n_{use} points of it. $len()$ returns the list size. The point that is more likely to collide is stored in list Pcl_{use} to have a higher priority for being checked (line 3-4). When the number of points in Pcl_{use} is more than n_{use}, Pcl_{use} is randomly sampled to limit its size. The limited size of Pcl_{use} reduces and stabilizes the time cost for the collision check. In addition, if n_{use} and r_{safe} are reasonable, safety is not compromised in extensive simulation and hardware tests.

(2) Improve the Motion Primitives Generation Efficiency

Previously [82], we considered the flight time to reach w_p 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 motion planner steps, the planned trajectory deviates considerably from w_p, 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 (3.7) is proposed. It is slightly different from that of [82], to improve the success rate and time cost.

$$
\begin{aligned}
\min_{a_n} \quad & \|a_n\|_2^2 + \eta_1 \|\overrightarrow{w_p p_{n+1}}\|_2 + \eta_2 \frac{\|\overrightarrow{p_n p^*_{n+1}} \times \overrightarrow{p^*_{n+1} w_p}\|_2}{\|\overrightarrow{p_n w_p}\|_2} \\
\text{s.t.} \quad & v_n = \dot{p}_n,\ a_n = \dot{v}_n \\
& \|v_{n+1}\|_2 \le v_{\max},\ \|a_n\|_2 \le a_{\max} \\
& v_{n+1} = v_n + a_n t_{avs} \\
& p_{n+1} = p_n + v_n t_{avs} + \frac{1}{2} a_n t_{avs}^2 \\
& p^*_{n+1} = p_n + 2 v_n t_{avs} + 2 a_n t_{avs}^2
\end{aligned}
\tag{3.8}
$$

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 motion planner. The endpoint

constraint is moved to the objective function. The endpoint of the predicted trajectory need not coincidence with w_p. Because the execution time of the motion planner is always much smaller than the planned time to reach w_p, before the drone reaches w_p, 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 motion planner is sufficient. Therefore, minimizing the distance between the trajectory endpoint at t_{avs} and w_p is reasonable. Given that the current step run time of the motion planner may exceed t_{avs}, the distance from the trajectory endpoint at $2t_{avs}$ to $\overline{p_n w_p}$ should also be optimized. The subscript n presents the current step in a rolling process of the motion planner. $v_n \in \mathbb{R}^3$ and $a_n \in \mathbb{R}^3$ are the current velocity and acceleration of the drone. v_{max} and a_{max} are the kinematic constraints for speed and acceleration, respectively, and v_{n+1}, p_{n+1}, and p_{n+1} are calculated using the kinematic formula. 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_p can be detected. The time cost of the motion planning and safety of the motion planner is greatly improved. We adopt the revised optimization formula instead of the one introduced in section 3.2.3 in the final version of this system.

As a result, the safety criterion changes and is detailed in (3.9) and (3.10).

$$\begin{aligned} d_{\max} &= \max\left(\frac{\|v_n\|_2^2}{2\|a_n\|_2}\right) \\ s.t.\ &\frac{\sqrt{\|v_n\|_2^2 + 2\|a_n\|_2\|w_p - p_n\|_2} + \|v_n\|_2}{\|a_n\|_2} \le t_{avs} \end{aligned} \tag{3.9}$$

$$d_{\max} = \frac{\|v_n\|_2^2 t_{avs}^2}{4\left(t_{avs}\|v_n\|_2 + \left\|w_p - p_n\right\|_2\right)} \tag{3.10}$$

3.3 Experimental Results

3.3.1 Experimental Configuration

Our proposed HAS-based trajectory planner was tested and verified in the Robot Operation System (ROS)/Gazebo simulation environment. Gazebo is a simulation software which can provide a physical simulation environment close to the real world. The model of the drone we use in the simulation is IRIS, the depth camera model is Kinect V2, and the PX4 1.7.4 firmware version is used as the underlying flight controller. Mavros package is deployed for establishing the communication between our planner node and the PX4 control module. The acceleration controller for tracking is provided by the PX4 module by default. The point cloud filters are executed by C++ code and the other parts are executed by Python scripts. All these timing breakdowns were measured using an IntelCore i7-8565U 1.8GHz Processor. Table 3.1 describes the parameter settings of the planner in the simulation test. To make the depth camera observe the environment more efficiently, we control the yaw angle of the drone to keep the camera always heading toward the goal during the flight.

Table 3.1.: Parameters for simulation

Parameter	Value	Parameter	Value
$\Delta\alpha$	10°	d_{use}	3m
a_{max}	$4m/s^2$	r_{safe}	0.8m
voxel size	$0.2m$	v_{max}	$3m/s$
ξ	0.01m	μ	0.1
l_d	$3m$	d_{max}	$0.68m < r_{safe}$
t_{max}	$0.5s$	η	1.2

Two flight tests of increasing difficulty are presented in section B and section C to show the planning trajectory in 3D space and the time cost of each planning step. The obstacles are not known a priori and are unobservable at takeoff.

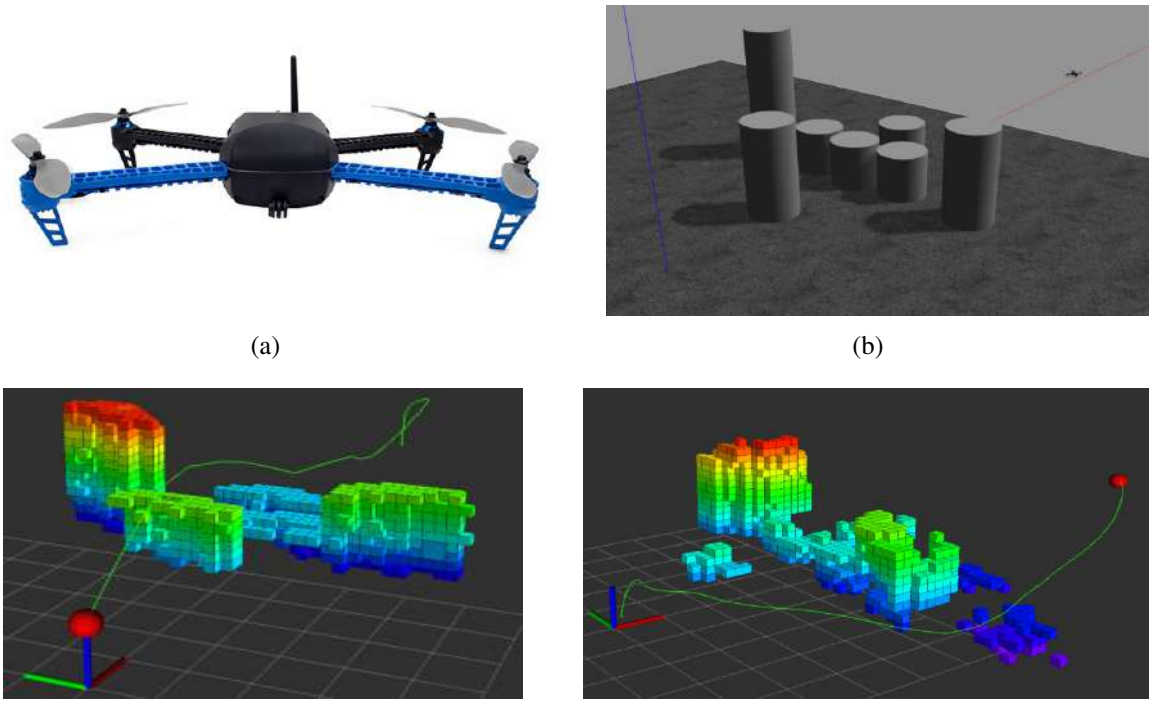
3.3.2 Simulation Flight Test in a Simple Environment

The test results are shown in the Fig. 3.4. In the first flight, the starting point of the drone is (0,0,0), and the red point indicates the navigation target point (12,0,1). After reaching the target point, set the starting point change to (12,0,0) and the endpoint is set to (0,0,1), then another test is performed. This is to test the drone's ability to avoid obstacles in the horizontal and vertical directions. According to the design of the algorithm, the drone will choose the path with the smallest amount of angle change of the flight direction when the obstacle can be avoided both horizontally and vertically. Because turning the drone too fast will increase the noise of the point cloud data obtained by the depth camera and destroy the established map, the attitude angle of the drone should be kept as smooth as possible.

The global 3D map and the flight trajectory of the drone during the flight are displayed in RVIZ, as shown in Fig. 3.4(c)-(d). In the first flight, the drone first raised its height to avoid the obstacles in the face of short obstacles and then chose to fly to the left to avoid the higher obstacles. In the second test, the drone first chose to fly to the left and then chose to continue to the left, because this minimizes the amount of angle change in the flight direction.

3.3.3 Simulation Flight Test in a Complex Environment

In this test, we built a more complex simulation world as shown in Fig. 3.5. Due to the limited space in the Chapter, we show only one flight's results. The start point is set at (12,0,0), the endpoint is set at (-12,0,1). The flight trajectory of the drone and the established global map are shown in Fig. 3.5 and Fig. 3.6. After repeating the flight experiments 10 times, the detailed data of the trajectory and the average running time of each part of the planner are shown in Fig. 3.7 and Fig. 3.8. Table 3.2 compares the calculation time of our proposed planner to the state-of-the-art, where Cond.1 means A_{g0} is fixed to goal direction, and Cond.2 means all points in Pcl_3 are used for collision check. PL(path length) factor is the ratio of the whole path length to the straight distance from the start point to goal. It is worth noting that the difficulty of the simulation tests in this table are different, we show the



(a) (b)

(c) (d)

Fig. 3.4.: (a) an IRIS drone, (b) a simple simulation environment, (c) and (d) show the results of the first and the second flight test respectively. The trajectory is shown in the green line.

data here for a preliminary comparison. We can see that our proposed planner has obvious advantages in computational time, but the whole path length is longer than some of others' works.

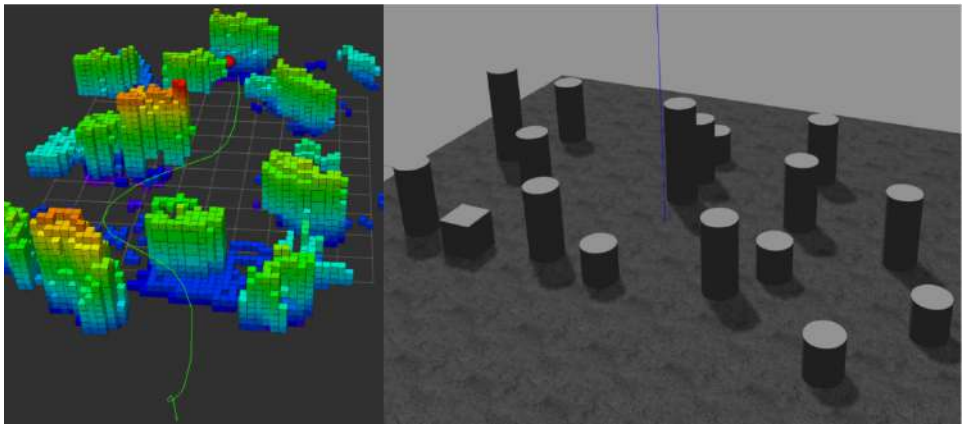
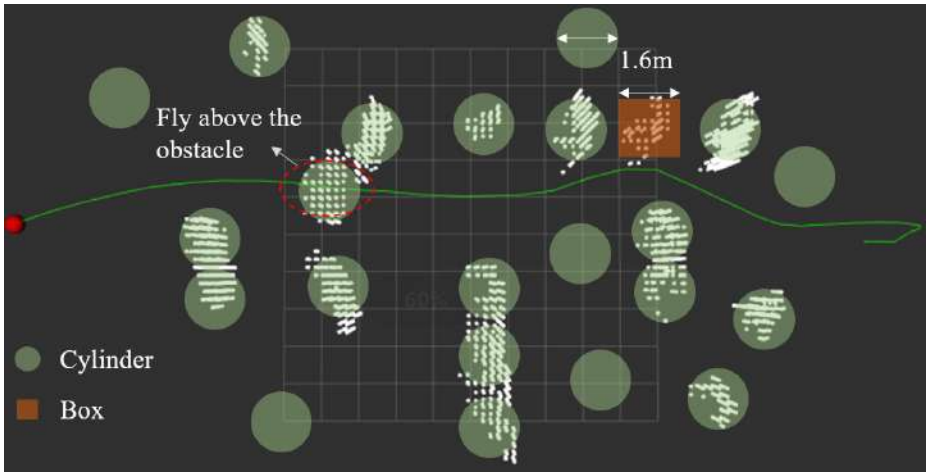


Fig. 3.5.: Our simulation environment and visualized data of results

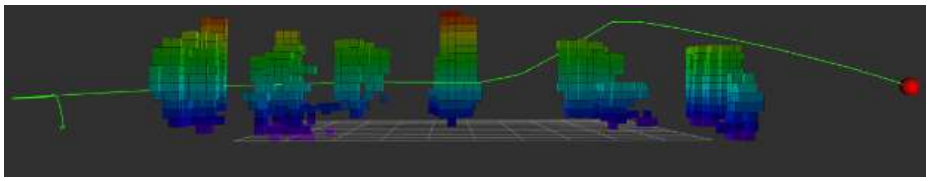
Table 3.2.: Comparison with state-of-the-art algorithms.

Authors	Time(ms)	PL Factor	Authors	Time(ms)	PL Factor
Zhou et al.[48]	>100	1.56	Tordesillas et al.[83]	>25	**1.34**
Liu et al.[40]	>160	-	Chen et al.[59]	>34	-
Burri et al.[84]	>40	1.78	**This Chapter**	**19**	1.48
This Chapter (Cond.1)	29	1.48	This Chapter (Cond.2)	42	1.44

In Fig. 3.7(b)-(c), the curve changes intensely near the end because the drone was switched to position control mode when it is close enough(<0.3 m in this test) to the goal. We can see from the boxplot that the number of iteration time in the angular search is the major influential factor to the time cost. Fig. 3.8(a) shows that in most instances the HAS method can work out the feasible solution with less than 3 steps, so the average step time can be controlled within 20 ms. The time cost also relates to the number of input points to some extent, which means we can decrease the time cost by simplifying the point cloud (Pcl_r) in a more efficient way.



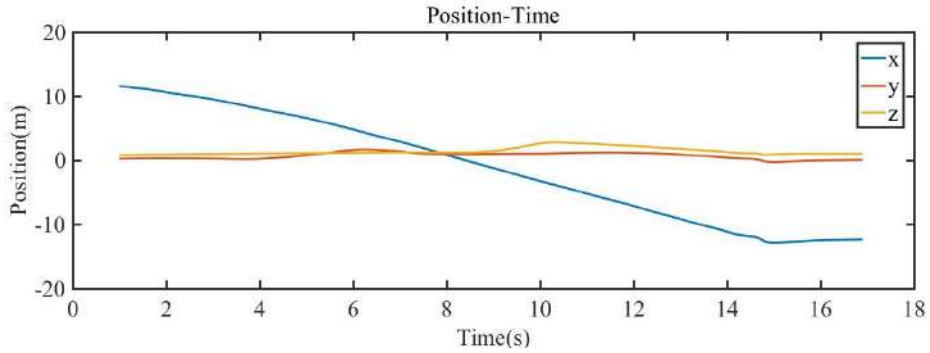
(a)



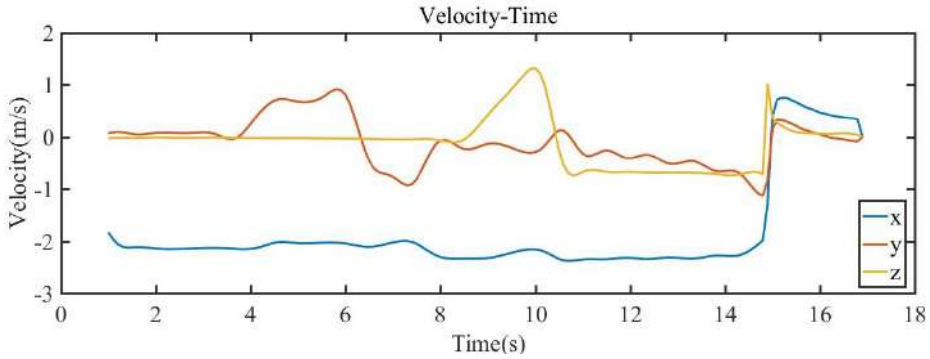
(b)

Fig. 3.6.: Results of the test in a complex environment. (a) shows the point cloud of the global map and the size of the obstacles from the top view, (b) shows the Octomap from a side view. The trajectory is shown in the green line.

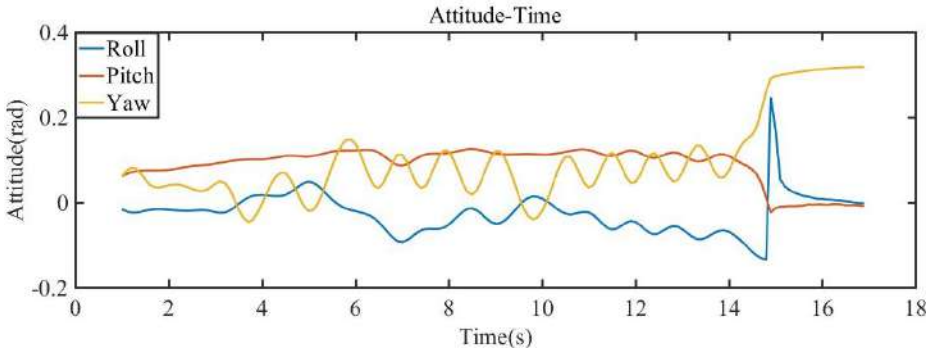
However, in additional series of simulation tests, we found the proposed trajectory planner may fail in two typical scenarios as shown in Fig. 3.9: a room with narrow exit and a forest with dense and tall pillars. To improve the computation efficient, the collision check is not performed in range of 360° on the point cloud. Although the drone returns to the last recorded position when it fails to find a waypoint, it may fail to exit the room if the exit is too narrow. In the dense forest, although the gap between the pillars allows the drone to pass, considering the fixed r_{safe} the drone can't find a collision-free line segment under such condition. In addition, there is still huge room for improvement by changing the code to C++, improving the hardware of the simulation platform.



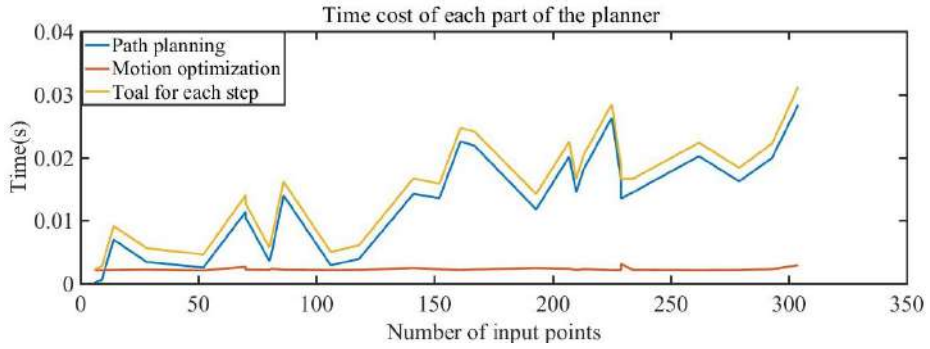
(a)



(b)

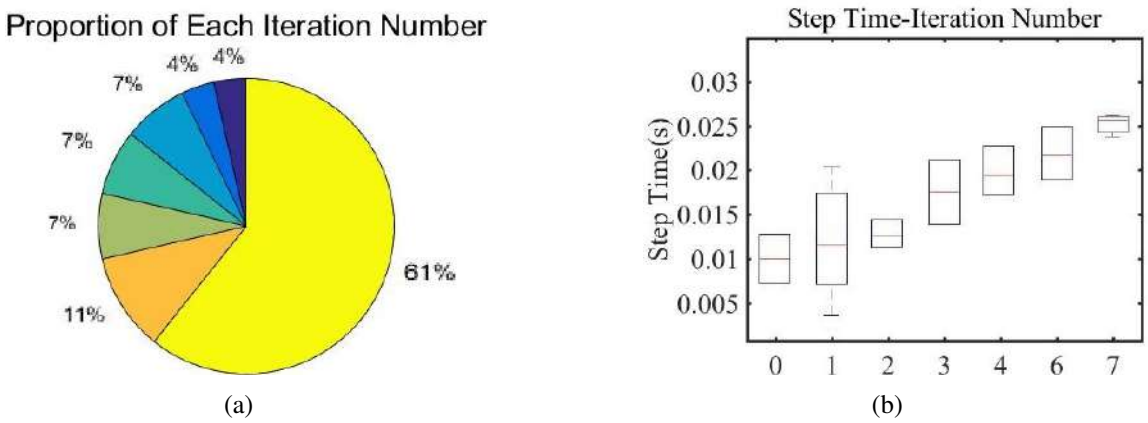


(c)



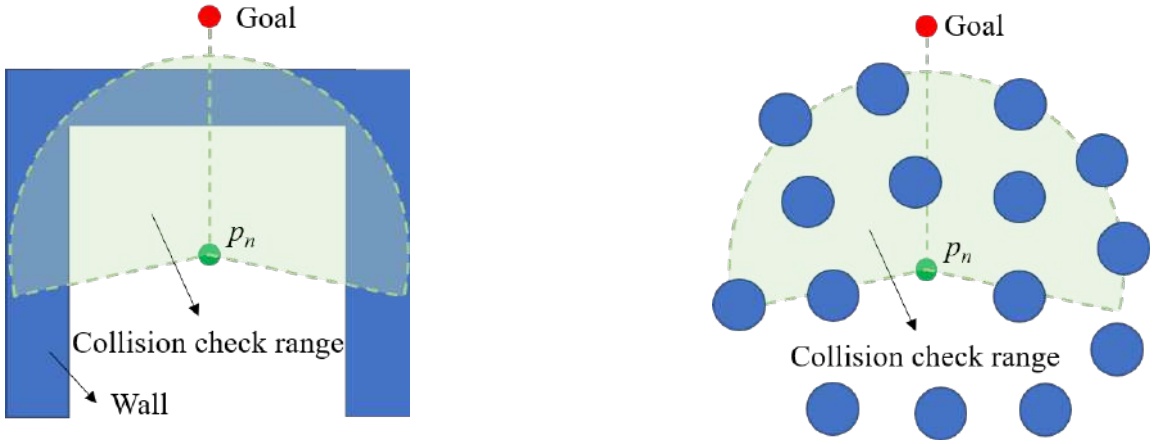
(d)

Fig. 3.7.: (a)-(c) curve of the three-axis coordinate position, flight speed, attitude angle respectively; (d) curve of time cost of each part of the planner versus number of points in Pcl_r.



(a) (b)

Fig. 3.8.: (a) pie chart for the proportion of each iteration number; (b) the boxplot of time cost for each iteration number.



(a) (b)

Fig. 3.9.: The two scenarios in which the drone are most likely fail to find a free path

3.3.4 The Improvements in Optimization Formulation

To validate the improvements in the optimization introduced in equation (3.5), we perform static tests with a large amount of data collected from simulation flight tests. We record all the required data for solving the optimization problem, including p_n, v_n, and w_p, 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 (3.5) and the improved one (3.8) 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 3.3, 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. For the improved optimization formulation, we choose $\eta_1 = 40$, $\eta_2 = 10$.

Table 3.3.: 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**

3.4 Conclusion

This work presented a trajectory planner's framework based on the HAS method, for safe and quick responding flights in unknown environments. The key properties of this planner are that it uses a direct waypoint search method on a simplified point cloud to reduce the time cost and the safety is ensured by restricting $d_{max} < r_{safe}$ by setting parameters and compromise on v_{max} and l_d when necessary. Our proposed planner was tested successfully in different simulation environments, achieving the average step time cost within 18 ms. The time cost is believed to be able to achieve a better level on a higher performance hardware platform with C++ code. In conclusion, the main contributions of this chapter are:

- The combination of a streamlined point cloud of global Octomap and the heuristic discrete angular search makes the computation load of finding a collision-free path much lighter. It improves the efficiency by generating waypoints directly on the point cloud rather than building a grid map and running a static path planning algorithm(such as A* or JPS) on the grid map afterward.

- The collision check can be removed from the motion planning part due to the introduction of r_{safe} and the constraint of maximum speed and acceleration of drone, the drone's position can be well constrained in the free space between the execution time of two contiguous steps of the trajectory planner.

- We propose three techniques to guarantee safety in the autonomous flight based on the mentioned method. Simulation experiments in ROS/Gazebo showing agile flights in completely unknown cluttered environments, with maximal average control output calculating time of iteration steps less than 18 ms.

We also test this local motion planner in real world with a self-assembled quadrotor platform, the results are shown in Chapter 4 where it is connected with a global path planner to build a more versatile system.

4. PARALLEL NAVIGATION FRAMEWORK FOR FLIGHTS IN COMPLEX TERRAIN

4.1 Research Background

Minimizing the reaction time towards environment change 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. The UAV is more likely to detour, resulting in an inefficient flight trajectory. 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. The drone must be able to avoid sudden obstacles in the unknown environment before the map is updated. Thus, to achieve autonomous unmanned aerial vehicle (UAV) navigation in unknown and complex environments, we should combine the two kinds of planner (point cloud planner and map planner) together in a reasonable way. In this chapter, we propose a framework in which a low-frequency path planner and a high-frequency reactive motion planner work in parallel. It makes the drone follow the solved path on map and can avoid the suddenly appearing obstacles nearby.

Considering the path planning on map, two important indicators are usually considered except safety: path length, and calculation time for replanning. However, regarding path length and calculation time, most researchers have only focused on one of the two factors because of the potential conflicts between them. In other words, calculating a shorter global path is always more time-consuming.

In addition, calculating the shortest path in the global 3D map consumes too much time and is not applicable to real-time planning. Although we do not expect the path replanning time cost very short, it should be up-to-date with the map. While flying in an unknown environment, the environmental information sensed by the drone is continuously used to update the map. 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, 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. The computation complexity of the path planning should also be optimized. Because we should consider the limited computing resource of the onboard computer and the path planning node shares the resource with many other running nodes, including the VIO (localization), mapping, and point cloud planner.

To optimize the computation, we propose several techniques. Because path planning with a 3D map is also expensive, the map planner (MP) first determine the 2D path to the local goal with the improved jump point search (JPS) method on the projection map. The double-stage 2D path planning is done with a downsampled coarse 2D map first and then on a fine map to balance the path length and computation cost. 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.

To summarize, we propose a framework in which a low-frequency 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 is used to determine the local goal for the pointcloud-based motion planner. The point cloud planner (PCP) for trajectory planning is based on the design in our previous work [82]. 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.

4.2 Mapping and the Map Planner

The flight system architecture is shown in Fig. 4.1. 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.

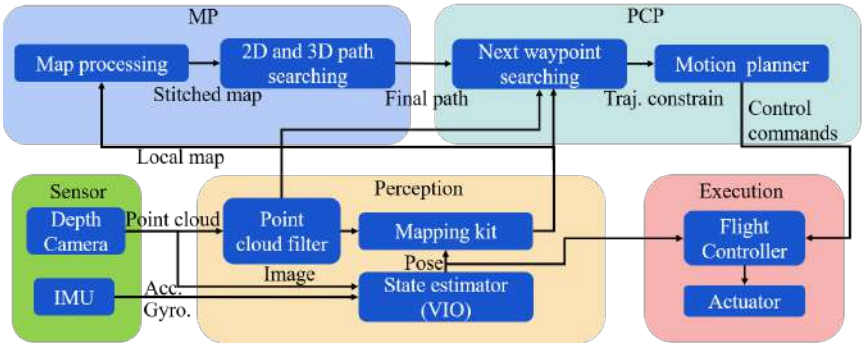


Fig. 4.1.: Architecture of our autonomous navigation system for UAVs.

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 IV. Moreover, the point cloud filter for the raw sensor data preprocessing is introduced at the beginning.

4.2.1 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, as is shown in Fig. 4.2 and Algorithm 2. Pcl_2 is the filtered point cloud in the Earth coordinate system, and we input Pcl_2 to 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. 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.

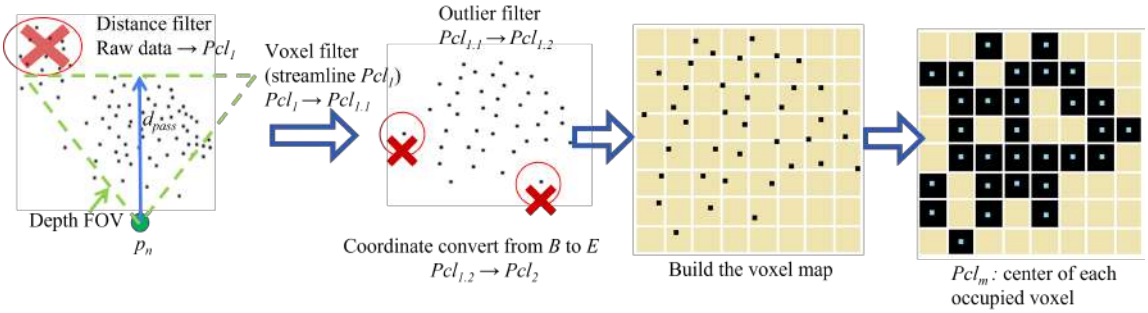


Fig. 4.2.: Process for point cloud filtering, coordinate transformation, and mapping. B denotes the body coordinate and E is for the earth coordinate.

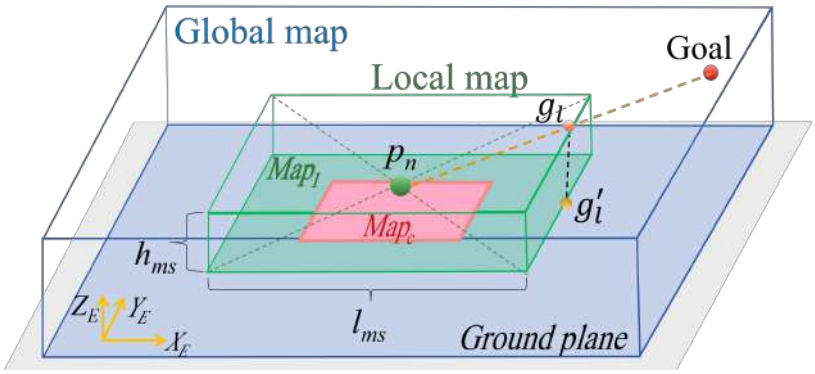


Fig. 4.3.: Local and global maps.

4.2.2 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 chapter. Here, we first

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

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. 4.3, 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 [82], 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. 4.4. 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

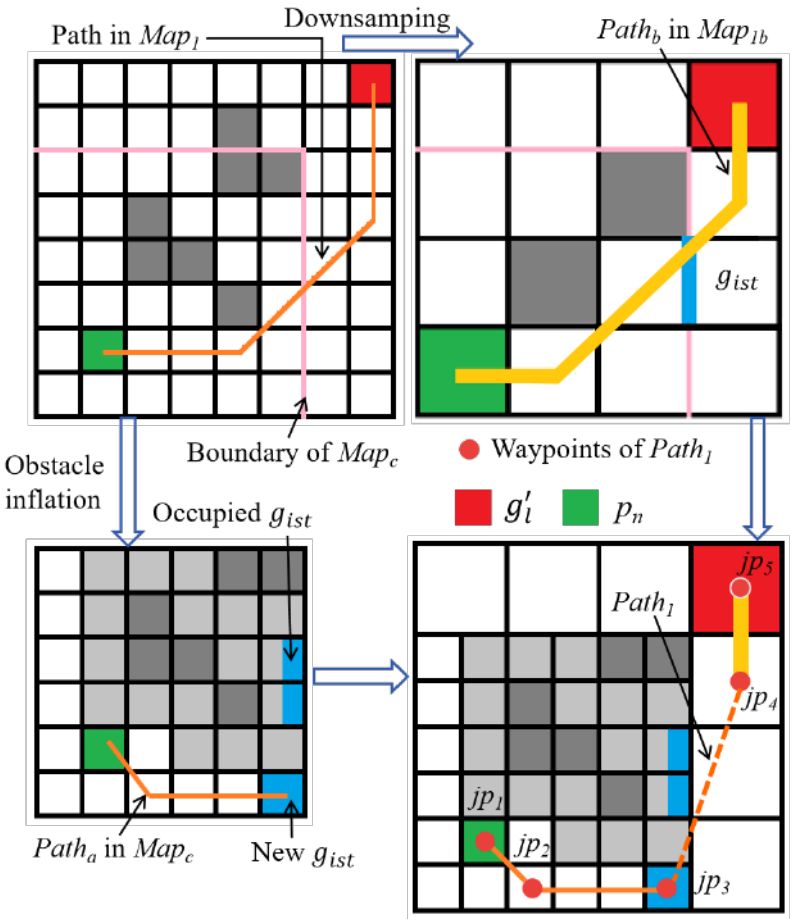


Fig. 4.4.: The map downsampling and obstacle inflating (k=3,h=2), and the path planning in the stiched map.

with Map_1 to reduce the map size and obtain a low-resolution version Map_{1b} from Map_1 (Fig. 4.4, top right). Map_c is segmented from Map_1 afterward as the original resolution map around the drone (Fig. 4.4, 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. 4.4, top 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 $Path_1 = \{jp_1, jp_2, \cdots\}$ (Fig. 4.4, 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.

$$Map_1' = \begin{bmatrix} [Map_1]_{i\times j} & [\mathbf{0}]_{(i+s)\times s} \\ [\mathbf{0}]_{s\times j} & \end{bmatrix} \tag{4.1}$$

$$Map_c' = \begin{bmatrix} & [\mathbf{0}]_{k\times(n+2k)} & \\ [\mathbf{0}]_{(m+2k)\times k} & [Map_c]_{m\times n} & [\mathbf{0}]_{(m+2k)\times k} \\ & [\mathbf{0}]_{k\times(n+2k)} & \end{bmatrix} \tag{4.2}$$

$$Map_c = Sgn(Conv_1([Map_c']_{(m+2k)\times(n+2k)}, I_{k\times k})) \tag{4.3}$$

$$Map_{1b} = \langle Conv_h([Map_1']_{(i+s)\times(j+s)}, \frac{I_{h\times h}}{h^2})\rangle \tag{4.4}$$

$$h = \langle \frac{ij}{2mn} \rangle \ (ij > 3mn \ \&\ (i+s) \text{ is divisible by } h) \tag{4.5}$$

Equations (4.1)-(4.4) 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. (4.3) represents the obstacle inflation process, and (4.4) is for the map downsampling. Fig. 4.4 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 (4.5), and $i,\ j,\ m,\ n,\ s$ should meet the conditions in the bracket.

4.2.3 Improved 2D path

In section III.B, a path $Path_1(jp_1, jp_2, ...)$ 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. 4.5, 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 6. 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 $\{jp'_1, \cdots, jp'_N\}$.

Algorithm 6: Optimize the original JPS path

1: **for** jp_{ck} in $Path_1$ (ck is the iteration number): **do**
2: $ti = ck + 2$
3: **while** $ti < len(Path_1)$ and $len(Path_1) > 2$ **do**
4: **if** $\overline{jp_{ck}jp_{ti}}$ does not collide with the occupied grids in the 2D map: **then**
5: $ti = ti - 1$, delete jp_{ti} from $Path_1$
6: $ti = ti + 1$

4.2.4 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 7, Fig. 4.5, and Fig. 4.6 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 $\{Pcl_{lm}^1, \cdots, Pcl_{lm}^N\}$, 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. 4.5, 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 7, line 3-8, as shown in Fig. 4.6(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 $\overline{p_n tp_1}$ is found (line 9), which has the minimal angle difference with $\overrightarrow{p_n g_l}$ (the yellow grid in Fig. 4.5 and Fig. 4.6). Next, the length of path segment $\overline{p_n tp_1}$ is determined in line 10, and the

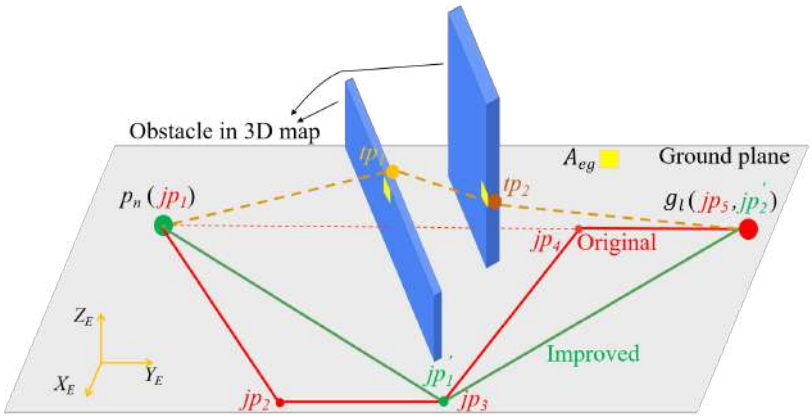
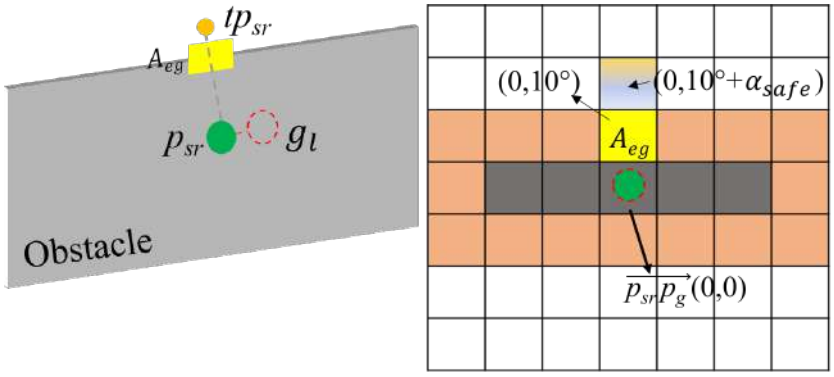


Fig. 4.5.: A scenario where the 3D path is much shorter than the improved 2D path.



(a) (b)

Fig. 4.6.: (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), $\alpha_{res} = 10°$.

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 7, 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 (4.6 (b)) unless the origin point

Algorithm 7: DAGS method

for sr in [1,N] ($sr \in \mathbb{N}$ is the searching rounds number) **do**
  If $sr = 1$, $p_{sr} = p_n$, otherwise $p_{sr} = tp_{sr-1}$
  The angular coordinate of $\overrightarrow{p_{sr}g_l} \rightarrow A_g(\alpha_{g1}, \alpha_{g2})$
  for each point p_{mi} in Pcl_{lm}^{sr}: **do**
    The angular coordinate of $p_{mi} \rightarrow A_{mi}$
    Calculate the angle difference $A_{mi-g} = A_{mi} - A_g$
    $A'_{mi-g} = \lfloor A_{mi-g}/\alpha_{res} \rfloor$ (Discretize A_{mi-g}), build the discrete angular graph (Fig. 4.6(b)) with A'_{mi-g}
  The edge of all A'_{mi-g} in the angular graph $\rightarrow A_{eg-all}$, $A_{eg} \subset A_{eg-all}$ and $\|A_{eg}\|_2 = min(\|A_{eg-all}(1)\|_2, \|A_{eg-all}(2)\|_2, ...)$
  Points in Pcl_{lm}^{sr} corresponding to $A_{eg} \rightarrow Pcl_{eg}$, $p_{eg} \subset Pcl_{eg}$ and p_{eg} has the maximal distance to $\overline{p_{sr}g_l}$, $l_{tp} = \overline{p_n p_{eg}}$
  Get the direction angle of $\overrightarrow{p_{sr}tp_{sr}}$, $A_{tp} = (\|A_{eg}\|_2 + \alpha_{safe})\dfrac{A_{eg}}{\|A_{eg}\|_2}$, $\alpha_{safe} = arcsin(r_{safe}/l_{tp})$
  $tp_{sr} = p_{sr} + l_{tp}(cos(A_{tp}(1)), sin(A_{tp}(1)), sin(A_{tp}(2)))$

p_{sr} in the search is tightly surrounded by obstacles in all directions (can not 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.

4.3 Complete Navigation Framework

This section will introduce how the goal g_n is generated from $Path_{fnl}$ for the PCP's current step n, and the complete navigation framework. First, the discrete angular search (DAS) method specifies the safe waypoint $w_p \in \mathbb{R}^3$ in free space (see Chapter 3), which

the drone should traverse. The motion planner solves the optimization equation to make the drone pass through w_p under the given motion constraints. The PCP also includes an additional safety measure to ensure that no collision will occur, which works when no w_p can be found in an emergency.

4.3.1 Connection between the PCP and MP

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

The final path $Path_{fnl} = [pt_1, pt_2, ..., g_l]$ is received from the MP ($Path_{fnl}$ dose not include p_n), and an optimization problem (4.6) 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_p 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.

$$\begin{aligned} \min_{v_t} \quad & \|v_t \frac{\|v_0\|_2}{\|v_t\|_2} - (p_n + v_0)\|_2 + \|v_t - \kappa_1 a_1\|_2 + \|v_t - \kappa_2 a_2\|_2 \\ \text{s.t.} \quad & a_1 = pt_1 - p_n,\ a_2 = pt_2 - p_n,\ v_t = \overrightarrow{p_n g_n} \end{aligned} \tag{4.6}$$

In (4.6), 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. 4.7 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_p is always on $\overline{p_n g_n}$. We can see from (4.6) and Fig. 4.7 that as the drone approaches the next waypoint pt_1, the influence

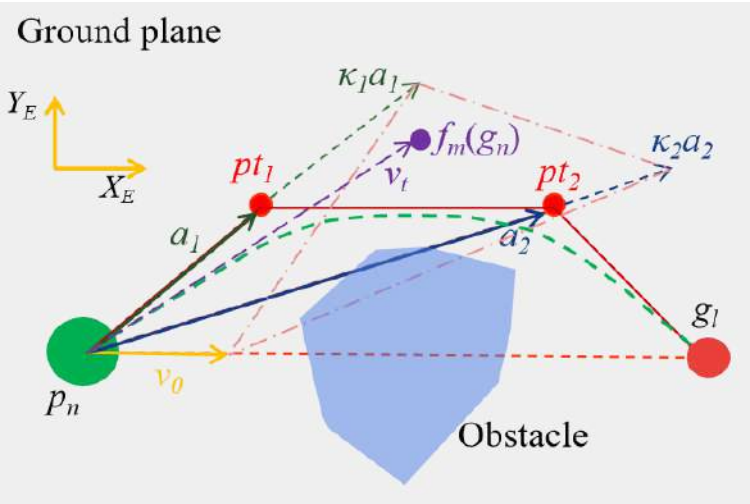


Fig. 4.7.: Geometric illustration of the analytical solution of (4.6). The pink, dashed line marks the Fermat triangle.

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 (4.6), is computationally expensive. From a geometric point of view, the nature of (4.6) 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. 4.7, and $g_n = f_m$. The calculation of f_m is illustrated in (4.7). 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 is the length of the side opposite to the triangle point (x_{f1}, y_{f1}), and so on. $f'_m(x_{fm}, y_{fm})$ is the coordinate of f_m in plane P_{fm}. Finally, f'_m is converted to $E-XYZ$ to obtain f_m.

$$
\begin{aligned}
x_{fm} &= (\sum_{i=1}^{3} x_{fi}(4S_{fm} + \sqrt{3}l_i^2) + g(y))/f_{Sl} \\
y_{fm} &= (\sum_{i=1}^{3} y_{fi}(4S_{fm} + \sqrt{3}l_i^2) + g(x))/f_{Sl} \\
g(x) &= [x_{f1}, x_{f2}, x_{f3}][l_3^2 - l_2^2, l_1^2 - l_3^2, l_2^2 - l_1^2]^T \\
g(y) &= [y_{f1}, y_{f2}, y_{f3}][l_3^2 - l_2^2, l_1^2 - l_3^2, l_2^2 - l_1^2]^T \\
f_{Sl} &= 12S_{fm} + \sqrt{3}(l_1^2 + l_2^2 + l_3^2)
\end{aligned}
\tag{4.7}
$$

4.3.2 The whole framework

To summarize, Algorithm 8 shows the overall proposed framework. The two planners (MP and PCP) are designed to run in ROS parallelly and asynchronously because of their large difference in operation time and share all the data involved in the calculation via the ROS master node. In addition, the point cloud filter and the mapping kit are run in parallel on different threads.

4.4 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.

4.4.1 Algorithm performance static test

Our detailed algorithms and methods are designed to obtain an undiminished or much better path planning performance with a decreased or slightly increased time cost. 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

Algorithm 8: our proposed framework

1: **while** true: (Thread 1) **do**
2: Filter the raw point cloud data, output Pcl_2
3: **while** true: (Thread 2) **do**
4: Build a global 3D voxel map Pcl_m, project the piece with similar height to the drone on ground to obtain Map_1
5: **while** the goal is not reached: (Thread 3) **do**
6: **if** the shorter 3D path has not been found or it collides with the updated Pcl_m: **then**
7: Apply downsampling on Map_1 to get Map_{1b}, find the 2D path $Path_1$ on the stitched map (Map_{1b},Map_c).
8: Try to find a shorter 3D path, output $Path_{fnl}$
9: **while** the goal is not reached: (Thread 4) **do**
10: Calculate the goal g_n from $Path_{fnl}$
11: Find the waypoint w_p by DAS method
12: **if** found a feasible waypoint: **then**
13: Run the motion planner to get motion primitives
14: **else**
15: Run the safety backup plan, and go to line 14
16: Send the motion primitives to the UAV flight controller

Core i7-8565U 1.8-4.6 GHz processor and 8 GB RAM, and Python 2.7 is used as the programming language.

(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. 4.4). A large-scale 2D map is used in the numerical tests, as shown in Fig. 4.8. The map size is 800 m*800 m, and the local map size is tested with

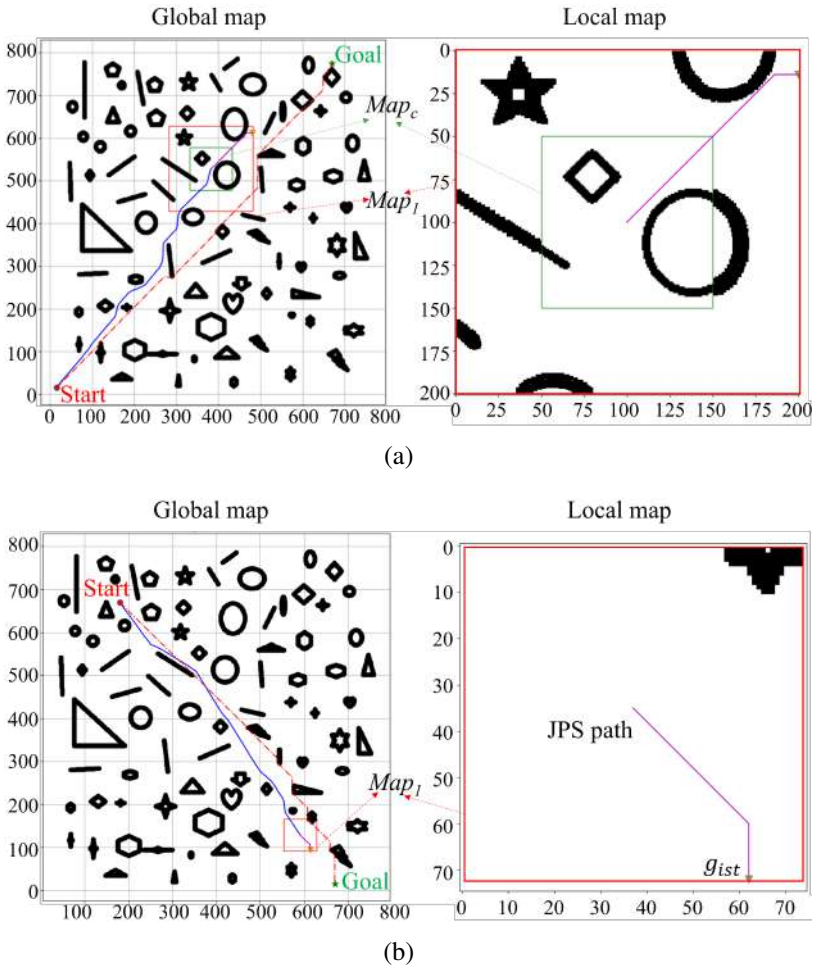


Fig. 4.8.: Visualized result during the numerical simulation. (a): only the sliding local map is used, with the map size of 75 m*75 m, (b): the double layer map is used.

three configurations (unit: m): 75*75, 200*200, and 400*400. The sizes of Map_1 and Map_c in Fig. 4.8(b) are 200*200 and 100*100 (unit: m), and the configuration for the obstacle inflation and map downsampling is the same with Fig. 4.4. 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 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 4.1. 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 4.1 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.

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 4.2, 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.

Table 4.1.: 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 4.2.: 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

(2) 3D path searching

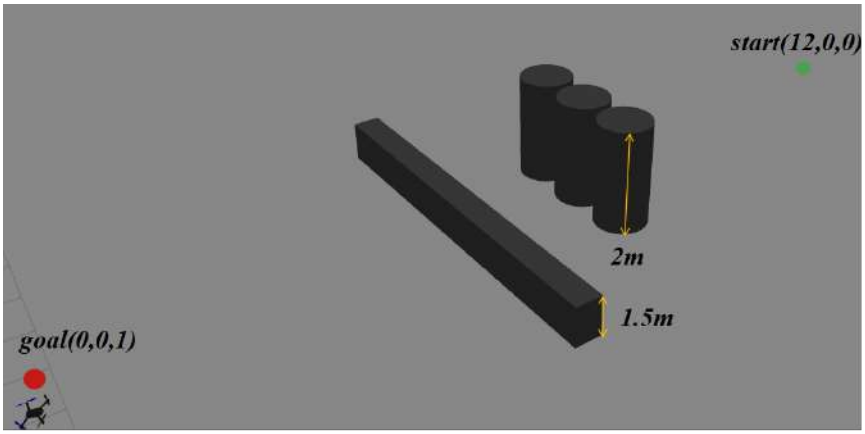
In addition, we compare the path length and computing time with JPS3D [85], RRT, and our proposed method in the local map. For fair comparison, all the methods are implemented with Python. We generate 60 box obstacles randomly distributed in a $20 \times 20 \times 6\ m^3$ 3D local voxel map (resolution 0.2 m) as the input, the goal position is also randomly assigned inside the map, and the initial position is always $(0,0,0)$. The box size varies from $0.4 \times 0.4 \times 0.8$ m^3 to $5 \times 5 \times 2\ m^3$, and each dimension varies independently. For each algorithm we perform 50 tests. For our method (only 2D path planning and DAGS), the average path length and time cost is 19.62 m and 0.031 s. Although JPS3D can search shorter paths of the average length 17.10 m, the time cost mean is 0.655 s, which is far from real-time requirement. RRT achieves 22.81 m with 0.036 s, both the two indicators are worse. Thus, our proposed method achieves the reasonable trade off between the computing time and path length.

4.4.2 Simulated flight tests with real-time planning

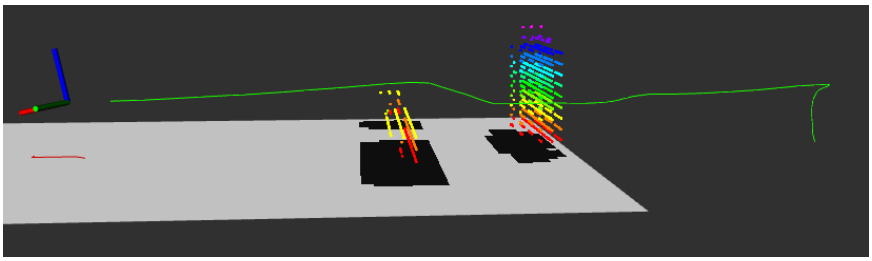
(1) Validity verification

We first test our whole navigation framework in two different scenes in Gazebo simulation. Gazebo is a simulation software that provides a physical simulation environment similar to the real world. Compared to our experimental hardware platform, all the simulation configurations are set up as the same or similar to ensure the credibility of the simulation and the analysis conclusion. The first world consists of a long and low wall and a row of

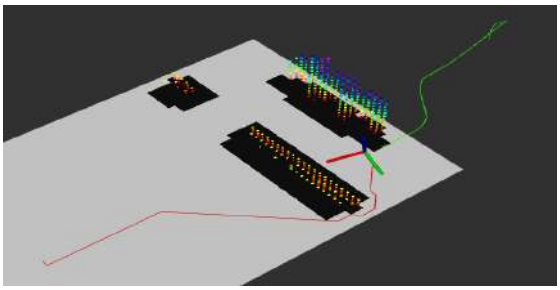
taller cylinders. It is designed to show the case that the 3D planning in 3D map takes effect to make the entire trajectory shorter. As is shown in Fig. 4.9, the drone pass the cylinders by side and fly above the wall. In Fig. 4.9(b), the red line is the 2D path on the ground plane, the green line is the actual trajectory in the past.



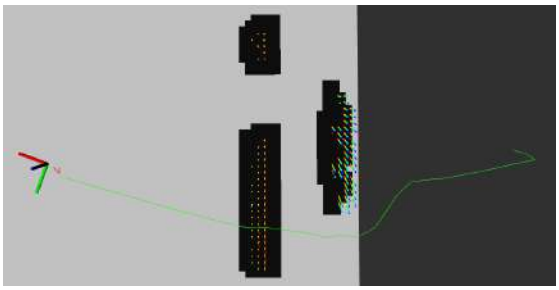
(a)



(b)



(c)



(d)

Fig. 4.9.: Simulation test 1 of the whole framework, (a)is the world in Gazebo, (b)-(d) are visualized data in RVIZ.

Then, we construct a complex world (see Fig. 4.10(a)-(b)) to study the overall performance of the proposed framework. The straight line distance between the start point and

goal is about 33.4m, there are not only obstacles with regular shape on the ground, but also some random items common in life, such as a dummy, table, barricade, wall and weeder. In the first flight (Fig. 4.10(c)) we disable the 3D path planning to see what will happen. The drone entered the box area near the goal because the low wall has not been detected. Then the map updated and MP replanned the path, the drone turned around and finally reached the goal. However, flying over the wall can reach the goal directly and avoid the detour, but MP doesn't plan in z axis. When we enable 3D planning, the flight result is demonstrated in Fig. 4.10(d), apparently the trajectory is much shorter and smoother. Fig. 4.10(e) and (f) are two flight results with different convolution kernel size k. When $k = 5$ in (e), the trajectory is longer than that with $k = 3$ in (f) because some narrow passages are banned by the inflation of obstacle at a higher safety level. However there occurred a tiny collision in (f) because the passage is too narrow, the time delay of the planner, the error in sensors and the control error of Px4 module all contribute to the uncertainty of collision avoidance.

To study the path length shortened by the 3D path search and corresponding extra time cost, the flight data in the simulation environment is analyzed. The visualized data during the flight is shown in Fig. 4.11. The obstacle feature size in the simulation is from 0.5 m to 6 m, which is similar to that of most real scenes. Three local 3D map bottom sizes are used (unit: m): 12*12, 20*20, and 30*30, and the map height is fixed at 6 m with the map resolution of 0.2 m. The drone is at the center of the local map. We also conduct 5 flight tests with different combinations of starting and goal points for each local map size. For each flight test, an additional flight test without the 3D path search procedure is used as the control group to compare the final trajectory length. During the flight simulation, the time cost of the 3D path search and the path length is recorded for statistical analysis. The results can be found in Table 4.3. η_{2D} indicates the mean of shortening percentage of the 3D path compared to the 2D path. T_{3D} is the average time cost for the 3D path search (Algorithm 7). Len_{3D} and Len_{2D} are the actual trajectory average lengths for the flight tests with and without Algorithm 7, respectively.

We can see that η_{2D} and Len_{3D} decrease as the map size increases. When the map size is small, the points in the 3D map reduces. Seemingly, the algorithm is more likely to find a

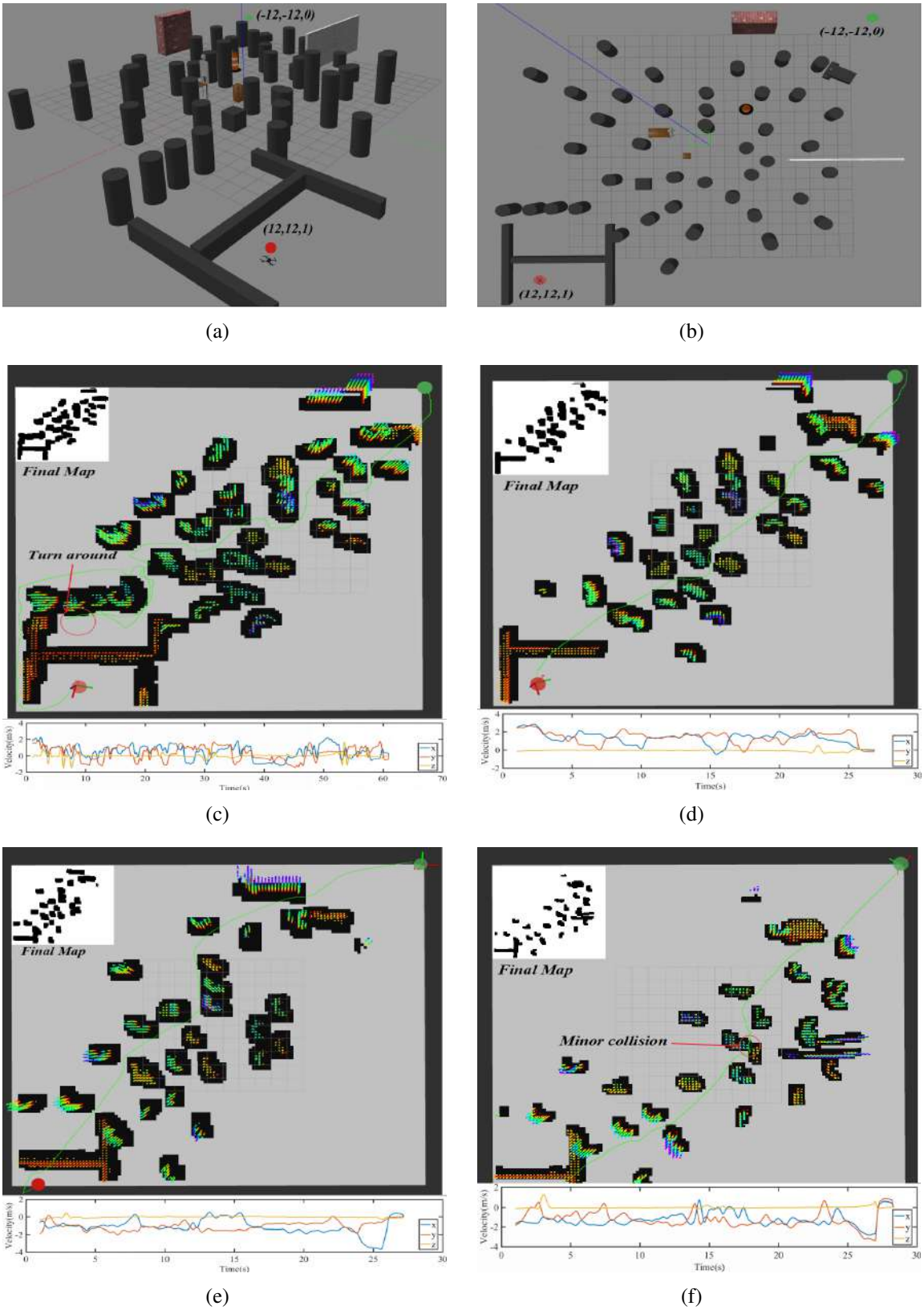


Fig. 4.10.: (a)-(b) are the simulation world in Gazebo. (c)-(d) are visualized data of different flights with different configurations in RVIZ, the final 2D map constructed by the mapper is attached on upper left corner, and curve of velocity is attached on lower right corner. In (c) and (d), the start point is marked in green and the goal is in red, and it is reversed in (e) and (f)

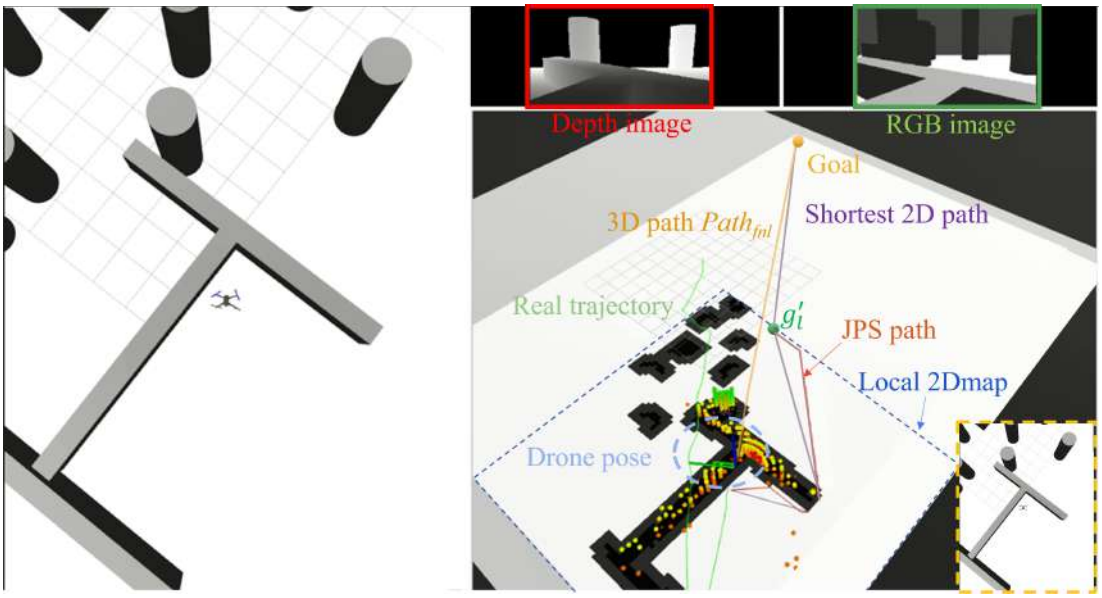


Fig. 4.11.: The visualized data in RVIZ. The Gazebo window when the flight test is ongoing is shown at the lower right corner. The colorful dots is the point cloud of the 3D local map. The black blocks on the ground plane in RVIZ stand for the obstacles, the white part stands for the free or unknown area.

Table 4.3.: Flight test results for the 3D path planning

Map size (m)	T_{3D} (s)	η_{2D} (%)	Len_{3D} (m)	Len_{2D} (m)
12*12	0.011	38.6	39.274	47.263
20*20	**0.032**	**35.7**	**32.635**	**44.582**
30*30	0.059	35.0	30.843	42.341

shorter 3D path. However, the 3D path of a small local map is more likely to collide with the newly appearing obstacles in the updated local map. Therefore, the drone has a high probability of taking a detour while flying along the 3D path, and the actual trajectory length is longer than when we use a large local map. When the local map size is settled as 20 m*20 m, the average outer loop frequency of the path planning is greater than 15 Hz, and the actual trajectory is smooth and natural. We use this map size in the following flight tests.

(2) Comparison with the global optimum

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 while 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. 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 4.4 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 4.4.: 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
$p_n w_p$	0.3 m	n_{use}	70
voxel size	0.2 m	pcl frequency	30 Hz
depth resolution	640*360	$\kappa_1,\ \kappa_2$	4.2, 1.5
r_{dec}	3 m		

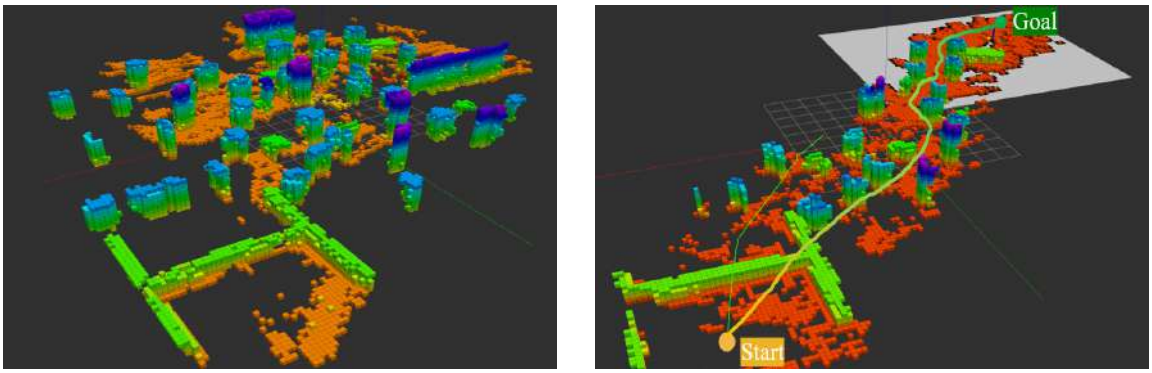
The mean of the 10 flight tests' results are obtained for comparison. The mean of length of the actual trajectory, RRT* path, and JPS path is 37.024 m, 32.985 m, and 32.822 m, respectively. We use the average JPS path length as the globally optimal path length. The actual path length increases by approximately 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. 4.12 illustrates the detailed visualized data of the flight test corresponding to one of the 10 flights. Fig. 4.12(a) demonstrates the entire map of the simulation world in Fig. 4.11. The coordinates of the starting and goal points of the resulted flight trajectory in Fig. 4.12(b) are $(11.4, 14.4, 0.8)$ and $(-12.0, -10.0, 1.2)$, respectively. The global optimal path length for this flight is 33.252 m, and the actual flight trajectory length is 36.057 m. In Fig. 4.12(c) and (d) we show the comparison of the globally optimal paths (found by RRT* and JPS3D) and the actual trajectory in (b), 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.

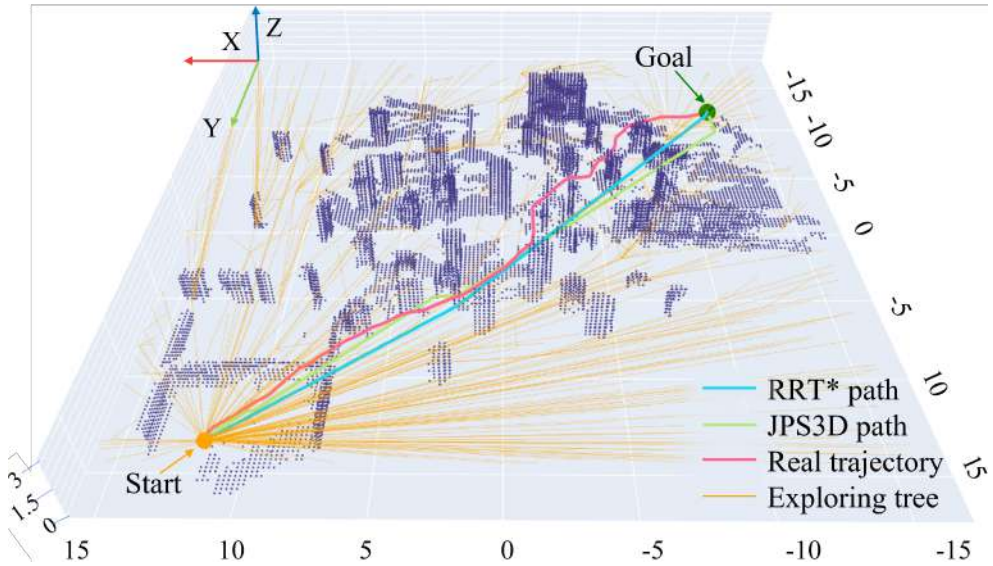
Table 4.5.: 3D path length comparison with the state-of-the-art algorithms

work	η_{3D}	Success rate	failure position
Ours	**12.8%**	**100%**	N/A
[82]	29.5%	85%	High wall
[64]	20.2%	**100%**	N/A

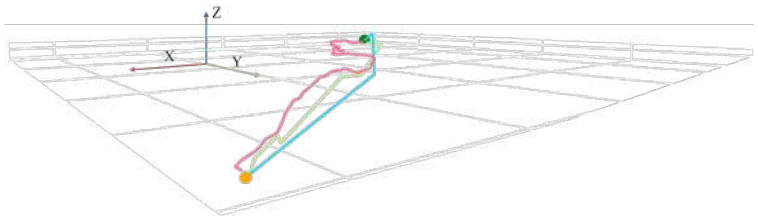
The trajectory length comparison between our proposed framework and the state-of-the-art algorithms is shown in Table 4.5. [82] is our former work proposes a local motion planner. η_{3D} indicates the relative length increment comparing the resulted path with the optimal 3D path. The results are evaluated from the flight test data in the same simulation world, with the open-sourced code [64] and the default parameters. We conduct 20 flights for each



(a) (b)



(c)



(d)

Fig. 4.12.: The visualized results for the flight tests in Gazebo.

work and the initial and goal position is randomly assigned as the above tests, the maximum velocity and acceleration are $1m/s$ and $5m/s^2$, respectively. Our proposed framework can fly the shortest trajectory compared to the works listed in Table 4.5, also the success rate is the highest. [82] 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 [64] is farther, the front-end path has no optimality guarantee even in a local map, thus the resulted trajectory is longer.

4.4.3 Hardware flight tests

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

(1) Introduction of hardware platform

We conduct the hardware tests on a self-assembled quadrotor. The Intel RealSense depth camera D435i is installed 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[2] 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

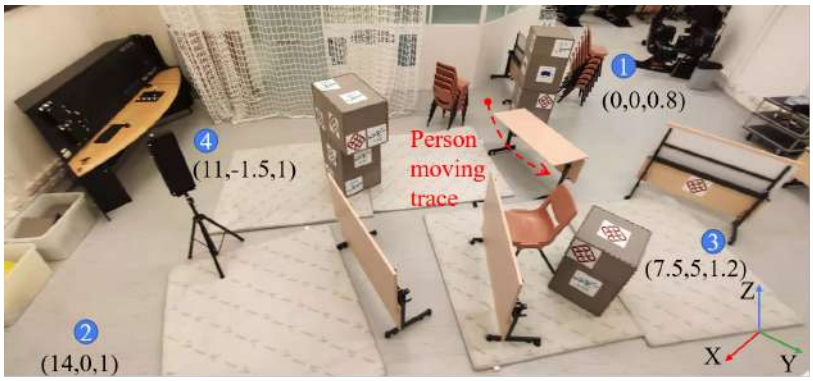
[2]https://github.com/HKPolyU-UAV/FLVIS

identical to the simulation flight tests, as shown in Table 4.4. The yaw angle of the drone is controlled to keep the camera always heading toward the local goal g_l'.

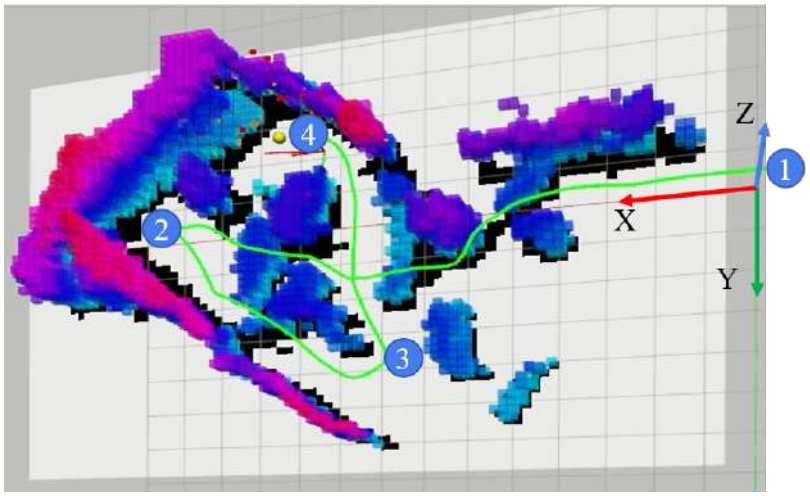
(2) Indoor tests

Fig. 4.13 (a) 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 the 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. 4.13 (b). The position, attitude, and velocity curves of the drone can be found in Fig. 4.14. 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 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 $\|\overrightarrow{p_n w_p}\|_2$ is assigned to be always greater than $\|\overrightarrow{p_n p_{n+1}}\|_2$, in the motion optimization problem defined in chapter 3, minimizing the cost $\eta_1\|\overrightarrow{w_p p_{n+1}}\|_2$ 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.

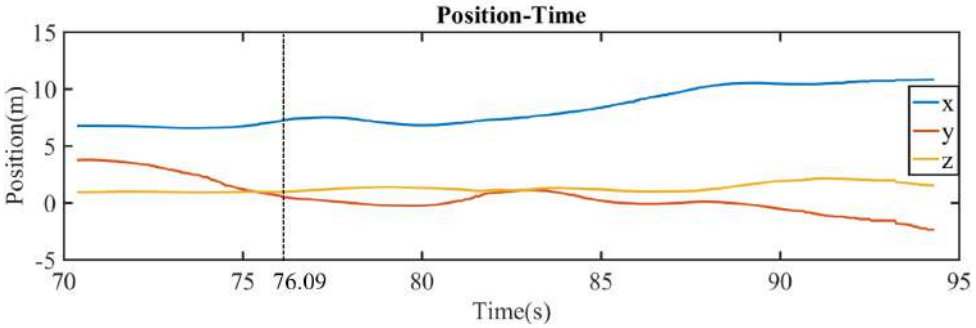


(a)

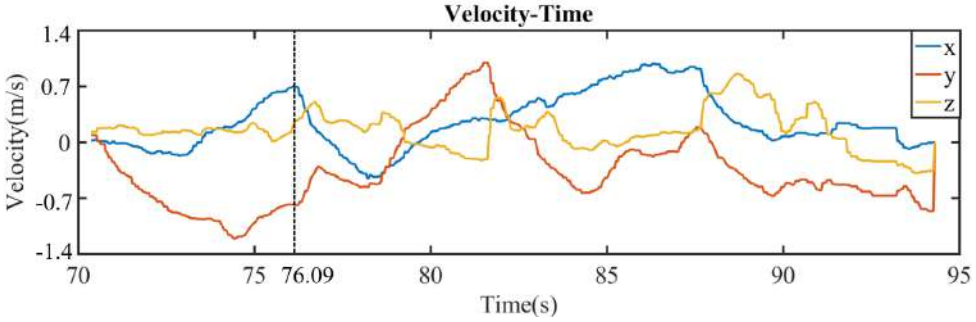


(b)

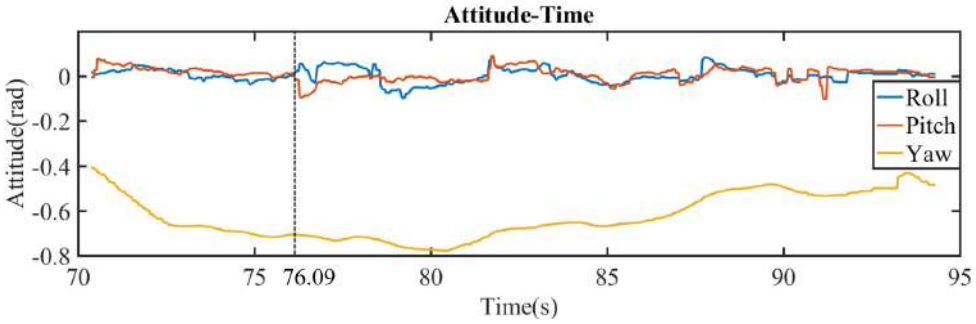
Fig. 4.13.: (a): Indoor environment for the hardware flight tests. (b): Explored map and the drone trajectory after the hardware flight test in the static environment.



(a)



(b)



(c)

Fig. 4.14.: Curves of the three-axis coordinate positions, flight velocities, and attitude angles.

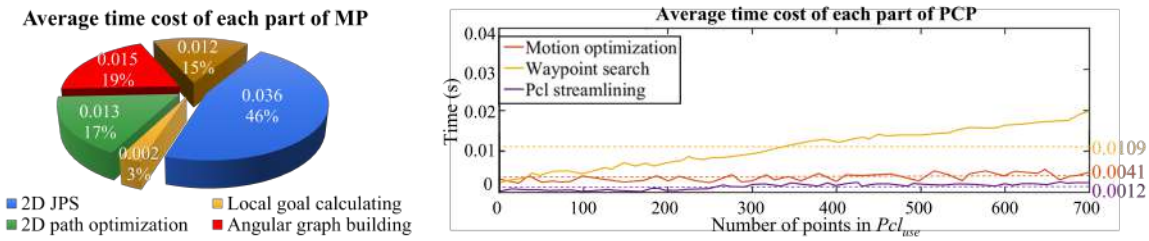
(a) (b)

Fig. 4.15.: Two of the outdoor flight test environments. (a) locates in a campus and (b) is at the sports corner in a park.

(3) Outdoor tests

In addition, the proposed framework is tested in diverse outdoor scenarios. Fig. 4.15 shows the environments of two experiments, and the flight tests in other environments are included in the video. Fig. 4.15(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. 4.15(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 intruding person causes the change of environment so that we can observe the flight performance in such a dynamic environment. 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. 4.16 to show the computational efficiency. In Fig. 4.16(a), the average time cost and the percentage are provided on a pie chart. In Fig. 4.16(b), the relationship between the time cost of each procedure in the PCP and the size of Pcl_{use} is illustrated, with the average time cost shown on the right side. 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



(a) (b)

Fig. 4.16.: (a): Average time cost and the proportion of each submodule of MP. (b): The time cost versus Pcl_{use} size curves of each part of the PCP.

step is 0.078 s, 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, only the time cost of the waypoint searching is relevant to the Pcl_{use} size because the number of the points determines the collision check's circling number. The average time cost of the PCP step is 16.2 ms, of which searching for w_p is the most time-consuming part.

Moreover, the time cost is compared with those of the state-of-the-art algorithms in Table 4.6. 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 4.6 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 state-of-the-art 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 4.7, 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. The test configurations are the same with the test described in subsection V-A, the last paragraph of part 2, and 50 random 3D voxel maps are used. We can conclude that our method always cost less time than JPS3D and can run in real-time even in full-load 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.

Table 4.6.: COMPARISON WITH STATE-OF-THE-ART ALGORITHMS.

Works	Time cost (ms)	MSCF (GHz)
MP	78	3.4
PCP	**16.2**	**3.4**
[48]	>100	3.0
[40]	>160	3.4
[84]	>40	N/A
[82]	19	4.6
[86]	199	3.1
[47]	106	3.0

Table 4.7.: 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

4.5 Conclusion

In this Chapter, 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 chapter can reduce the computing time cost, with the performance basically unchanged or even improved compared to that of the original method [82]. The real-time flight trajectory length outperforms those of the state-of-the-art 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. 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.
- 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.

Supplementary Materials

Demo video: `https://youtu.be/AOENvwf8sfM`, or `https://www.bilibili.com/video/BV1sR4y1s7fh?spm_id_from=333.999.0.0`

5. FLYING AND AVOIDING DYNAMIC OBSTACLES ON POINTCLOUDS

5.1 Research Background

Most existing frameworks that enable drones to autonomously fly in unknown environments consider all obstacles stationary. However, as quadrotors often fly at low altitudes, they are faced with various moving obstacles such as vehicles and pedestrians on the ground. One primary solution to avoid collision is to raise flight altitude to fly above all the obstacles. This method is not feasible for some indoor applications, because the flight altitude is limited in narrow indoor space, and the drones are often requested to interact with humans as well. Another solution is to assume all detected obstacles as static. But this method cannot guarantee the safety of the trajectory [59], considering measurement errors from the sensors and unmissable displacement of the dynamic obstacles.

Although we consider the obstacle avoidance in dynamic environment in chapter 3 and 4, the motion planner only take advantage of its fast computation speed to avoid intruding obstacles. In the algorithms introduced above, the dynamic obstacle and static obstacle are both represented by pointcloud, and a hidden assumption lies behind is that the pointcloud will not move in the future. For an obstacle just suddenly intrude into the vehicle's FOV and then turn to static, the method in Chapter 3 can react in time. However, for objects continuously moving, the error in obstacle position estimation will be not negligible, and may lead to collision. Because the planned trajectory is only guaranteed safe for the static pointcloud within its duration time, if the object is moving, the trajectory is no longer safe for obstacle's changing position. Obviously, the faster the obstacle and the drone, the greater the error caused by the above assumption. For obstacles moving at comparable speed with the drone, the method in Chapter 3 will be hazardous. As shown in Fig. 5.1, the collision may happen if the motion information of obstacles is ignored (the red line). Therefore,

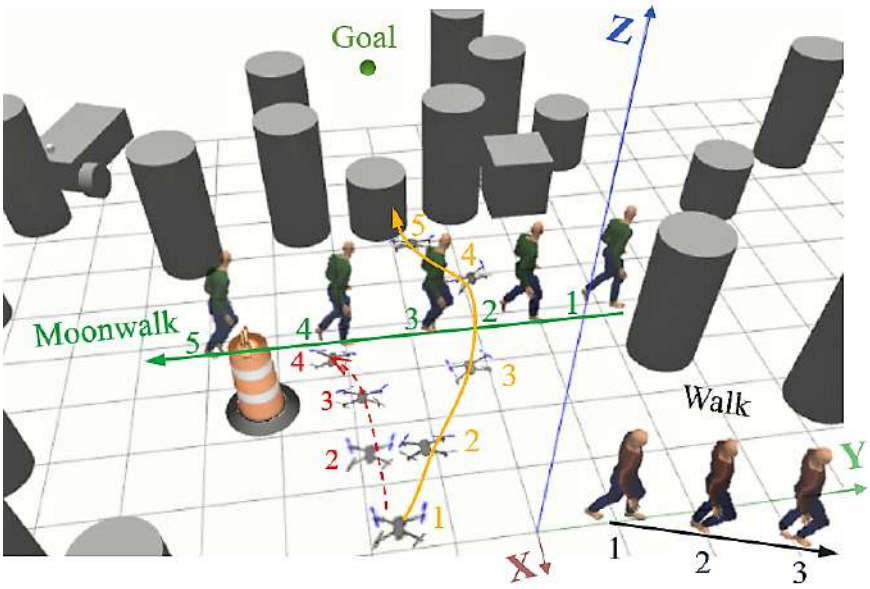


Fig. 5.1.: The composite picture of the simulation in Gazebo for the process that the drone avoids static and dynamic obstacles. 5 screenshots are used for composition and the cut time interval is fixed to 0.7 seconds. The line with an arrowhead shows the moving direction and the numbers mark the corresponding frame, the numbers increase over time. The yellow line is generated by the method in this chapter, while the red line is by the static planning method.

a more efficient and safer way to avoid moving obstacles is to predict and consider the obstacles' position in advance based on the velocity, which can avoid detours or deadlock on some occasions.

To fly in dynamic scenes for micro aerial platforms, we propose a complete system in this chapter, composed of a position and velocity estimator for moving obstacles, an upper-level planner to obtain the desired velocity, and a motion planner to generate final motion primitives. Considering the limited computation resource and demands for low cost, all the computation involved should be light and do not require high-precision sensor data. For the perception of dynamic obstacles, the dynamic ones are identified from static ones by clustering and comparing the displacement from two point cloud data frames. To overcome its narrow field of view (FOV), the static points are memorized to prevent the drone from hitting obstacles that just moved outside the FOV. An RGB-D camera is the only sensor utilized to obtain the point cloud. First, we set up a Kalman filter for each dynamic obstacle

for tracking and output more accurate and continuous estimating results. The feature vector for each obstacle is adopted to improve the obstacle matching accuracy and robustness, thus the dynamic obstacle tracking and position and velocity estimating performance are improved compared to the related existing works. In addition, we introduce the track point to reduce the displacement estimation error involved by the self-occlusion of the obstacle.

Then, with the estimated position and velocity of obstacles and the current states and kinodynamic limitations of a real vehicle, the forbidden pyramids method is applied to plan the desired velocity to avoid obstacles. The desired velocity is obtained from a sampling-based method in the feasible space, and the sampled velocity with the minimal acceleration cost is chosen. Finally, the motion primitives are efficiently solved from a well-designed non-linear optimization problem, where the desired velocity is the constraint. For navigation tasks, the proposed velocity planning method is also flexible to combine with most path planning algorithms for static environments, giving them the ability to avoid dynamic obstacles. The waypoint in the path acts as the trajectory endpoint constraints at a further time horizon. In this chapter, we test it with our former proposed waypoint planning method heuristic angular search (HAS) (Chapter 3) to complete the system and conduct the flight tests. The safety and the lower acceleration cost of this method can be verified by the data from flight tests. The computational efficiency of the whole system also shows great advantages over state-of-the-art (SOTA) works [72]-[76].

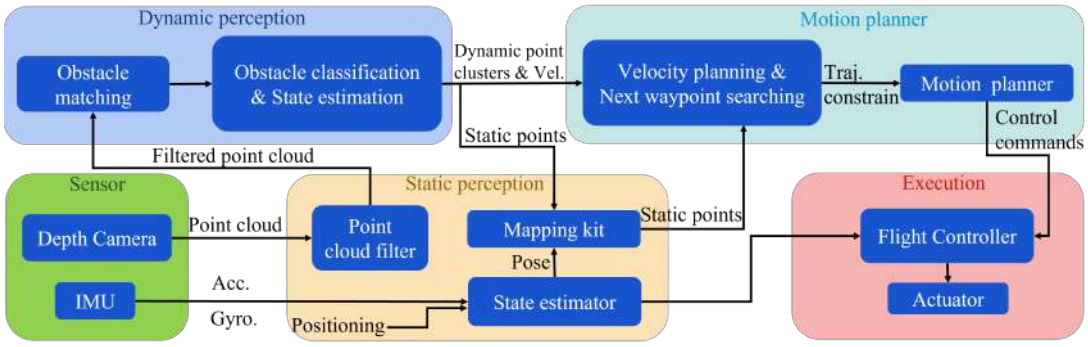


Fig. 5.2.: The proposed system for the autonomous navigation in dynamic environments. The positioning can be done by the outer motion capture system or onboard VIO toolkit.

5.2 Technical Approach

Our proposed framework is composed of two submodules that run parallelly and asynchronously: the obstacle classifier and motion state estimator (section 5.2.1 & 5.2.2) and the waypoint and motion planner (section 5.3.1 & 5.3.2). The additional technical details for improving the accuracy of dynamic perception are introduced in section 5.2. Fig. 5.2 illustrates the whole framework, including the important message flowing between the submodules.

5.2.1 Obstacle tracking

The raw point cloud is first filtered to remove the noise and converted into earth coordinate $E-XYZ$. The details about the filter are in chapter 3. We use Pcl_{t_1} and Pcl_{t_2} to denote two point cloud frames from the sensor at the former timestamp t_1 and the latest timestamp t_2 respectively. $t_2 - t_1 = d_t$ and $d_t > 0$. The time interval d_t is set to be able to make the displacement of the dynamic obstacles obvious enough to be observed, while maintaining an acceptable delay to output the estimation results. Pcl_{t_1} and Pcl_{t_2} are updated continuously while the sensor is operating. To deal with the movement of the camera between t_1 and t_2, Pcl_{t_2} is filtered, only keeping the points in the overlapped area of the camera's FOVs at t_1 and t_2 [76]. The newly appeared obstacles in the latest frame are removed, so only the obstacles appear in both of the two point cloud frames are further analyzed. Pcl_{t_1} and Pcl_{t_2} are clustered into individual objects using density-based spatial clustering of applications with noise (DBSCAN) [87], resulting in two sets of clusters $OB_1 = \{ob_{11}, ob_{12}, ...\}$ and $OB_2 = \{ob_{21}, ob_{22}, ...\}$. Then, matching the two clusters $ob_{2k} \in OB_2$ and $ob_{1j} \in OB_1$ corresponding to the same obstacle is necessary before the dynamic obstacle identification.

At the time when we obtain the first frame of Pcl_{t_2}, a list of Kalman filters with the constant velocity model is initialized for each cluster (obstacle) in OB_2. The position and velocity are updated after the obstacle is matched and the observation values are obtained. To match the obstacle, we preliminarily sort out the two clusters that satisfies the condition $\|pos(ob_{2k}) - pos(ob_{1j})\|_2 < d_m$. d_m is the distance threshold. $pos() \in \mathbb{R}^3$ gives the obstacle

position when the input cluster is from the latest frame Pcl_{t_2}, while it returns the predicted position at t_2 by the corresponding Kalman filter of the cluster from Pcl_{t_1} [76]. It is designed to associate current clusters to the forward propagated Kalman filters rather than clusters in the previous frame. If we cannot find an ob_{1j} for ob_{2k} there, we skip it and turn to the next cluster ($k \leftarrow k+1$, k is the cluster index). For each Kalman filter, it is also necessary to assign a reasonable maximal propagating time t_{kf} before being matched with a new observation. Because the camera FOV is narrow, we hope to predict the clusters which just move out of the FOV for safety consideration, and they are assumed to continue to move at their latest updated velocity in a short period. A Kalman filter is deleted together with its tracking history if it has not been matched for over t_{kf}.

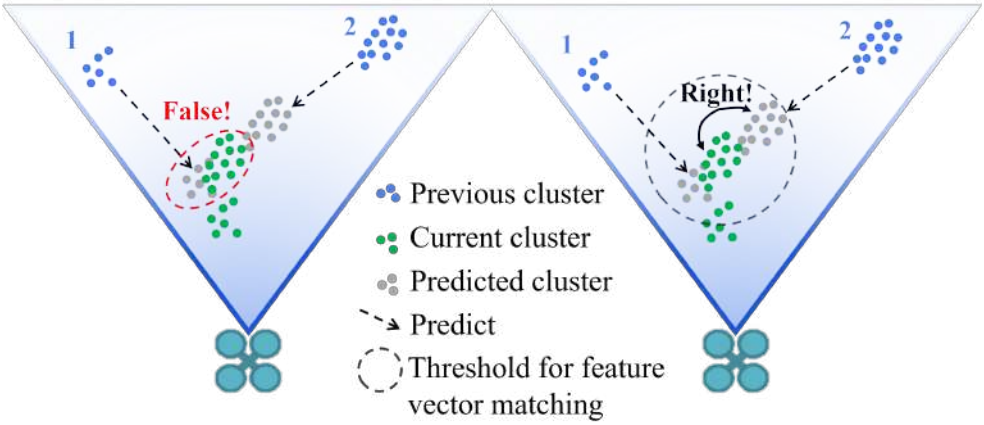


Fig. 5.3.: The left figure shows a situation that two obstacles are mismatched. The predicted cluster of obstacle 1 is closer than the predicted cluster 2 to the current cluster of obstacle 2. By comparing the feature vector, the correct predicted cluster for obstacle 2 can be matched for the current cluster, as shown in the right figure.

In the related works [21, 76], matching the clusters in two frames as the same object is based only on the center position. However, it may fail when obstacles are getting close, as shown in Fig. 5.3. To improve the matching robustness, we design a novel technique based on the feature vector to help match the clusters. The feature vector is composed of several

statistic characters of a point cluster with aligned color information from the obstacle, which is defined as

$$fte(ob) = [len(ob), V_P^A(ob), V(ob), M_C(ob), V_C^A(ob)], \tag{5.1}$$

where ob denotes any point cluster. $len()$ is the function that returns the size of the input cluster. $V_P^A() \in \mathbb{R}^3$ returns the position variance of the cluster, and $V()$ returns the volume of the axis-aligned bounding box (AABB) of the cluster. $M_C() \in \mathbb{R}^3$ and $V_C^A() \in \mathbb{R}^3$ return the mean and variance of the RGB value of points. The idea is: if there is not a significant difference in the shape and color of the two point clusters extracted from two timely close point cloud frames respectively, then they are commonly believed to be the same object. The global features for each obstacle are very cheap to calculate and proved to be effective in tests.

At last, the Euclidean distance $d_{fte} = \|fte(ob_{1j}) - fte(ob_{2k})\|_2$ between feature vectors is calculated with each cluster pair $\{ob_{2k}, ob_{1j}\}$ that satisfies the position threshold. For each cluster $ob_{2k} \in OB_2$, the cluster $ob_{1J} \in OB_1$ results in the minimal d_{fte} is matched with it. The feature vector is normalized to 0-1 since the order of magnitude of each element varies. We use $ob_2^m \in OB_2$ and $ob_1^m \in OB_1$ to represent any two successfully matched clusters.

5.2.2 Obstacle Velocity Estimation and Classification

Here, we will introduce the track point which effectively reduces the velocity estimation error compared to the existing methods.

After the obstacles in two sensor frames are matched in pairs, the velocity of the obstacle ob_2^m can be calculated by $v_2^m = \overrightarrow{p_2^m p_1^m}/(t_2 - t_1)$, where p_2^m and p_1^m are the position of the corresponding obstacle. The obstacles have certain shapes, they are not points. In the related works, the mean of the points in each cluster is adopted as the obstacle position, since it is easy to calculate and close to the centroid for a common obstacle if we ignore the self-occlusion. However, due to the self-occlusion, the backside of the obstacle is invisible, so the mean of points is closer to the camera than the obstacle centroid. In addition, the occluded part of a moving obstacle changes when the relative movement occurs between

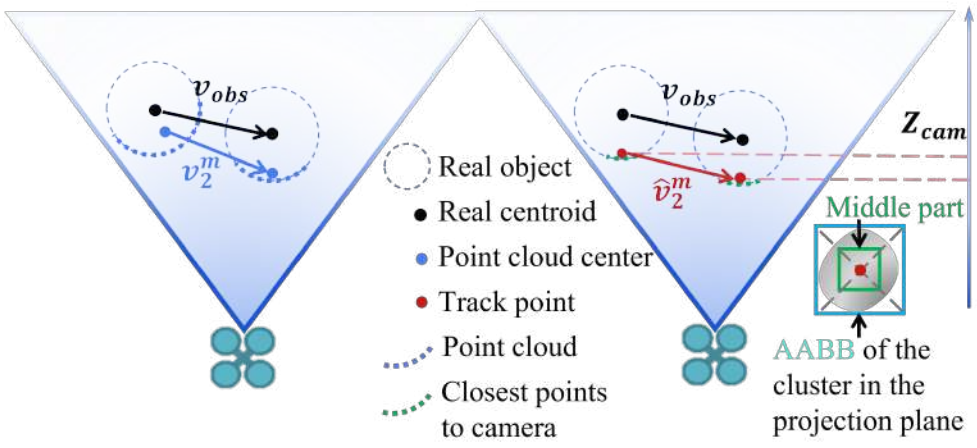


Fig. 5.4.: The left figure illustrates the velocity estimation error caused by the self-occlusion of the obstacle. v_{obs} is the velocity ground truth. When the obstacle approaches the camera, the visible part shrink, resulting in the relative displacement between the point cloud center and obstacle centroid. The track point in the right figure can reduce the velocity estimation error. The middle part of the cluster is bounded by the green box.

the camera and obstacle. Thus, the relative position between the position mean and the real mass center also changes. This will lead to a wrongly estimated velocity, as shown in Fig. 5.4, the error is mainly distributed on the Z axis of the camera Z_{cam} and it is not a fixed error can be estimated. As a consequence, the constant velocity model in the Kalman filters will no longer hold even for constant velocity obstacles. For the position estimation of obstacles, the self-occlusion is not important, considering the visible part only is safe for obstacle avoidance. But the velocity is the key information of moving obstacles in the vehicle velocity planning. Therefore, we propose a method to reduce this velocity estimation error by choosing the appropriate track point for the matched pair of clusters, as illustrated in the right figure of Fig. 5.4.

For a moving obstacle in translational motion, the closest part to the camera is believed not to be self-occluded in ob_2^m and ob_1^m. In addition, the middle part of the cluster is close to the centroid of the common obstacles, so the rotational movement is weak in this part. Thus, we only use the center point $\hat{p}_2^m$ and $\hat{p}_1^m$ of the closest N_c points to the camera in the middle part of the cluster ob_2^m and ob_1^m to estimate the displacement. $\hat{p}_2^m$ and $\hat{p}_1^m$ are named as the track point. Here the distance to the camera is measured only along Z_{cam}.

The middle part of the cluster is divided in the projection plane corresponding to the depth image. The bounding box for the middle part is shrunk from the AABB of the obstacle to the center proportionally. The shrinking scale factor is α ($\alpha < 1$). Considering the common obstacles usually performs slow rotation, and the time gap d_t is small, we can neglect the influence on the closest part caused by rotation in the displacement estimation. To update the Kalman filter, p_2^m (the mean of current cluster) is still the observed position, but the velocity observation is $\hat{v}_2^m = \overrightarrow{\hat{p}_1^m \hat{p}_2^m}/(t_2 - t_1)$. The classification for static and dynamic obstacle is done by comparing the velocity magnitude with a pre-assigned threshold v_{dy}, i.e. $\|\hat{v}_2^m\|_2 > v_{dy}$ indicates a moving obstacle. If an obstacle is classified as static in S_c consecutive times, the corresponding Kalman filter is abandoned and the static point cluster is forwarded for map fusion.

The Kalman filter for one cluster is detailed with

$$\hat{x}_t^- = F_t \hat{x}_{t-1} + B_t a_{t-1}, \tag{5.2}$$

$$P_t^- = F_t P_{t-1} F_t^T + Q, \tag{5.3}$$

$$K_t = P_t^- H^T \left(H P_t^- H^T + R \right)^{-1}, \tag{5.4}$$

$$P_t = \left(I - K_t H \right) P_t^-, \tag{5.5}$$

$$\hat{x}_t = \begin{cases} \hat{x}_t^- + K_t \left(x_t - H \hat{x}_t^- \right) \text{ (found dynamic obstacle)}, \\ \hat{x}_t^- \text{ (no dynamic obstacle)}, \end{cases} \tag{5.6}$$

$$x_t = \begin{bmatrix} p_2^m \\ \hat{v}_2^m \end{bmatrix}, F_t = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, B_t = \begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix}, \tag{5.7}$$

where R, H, Q are the observation noise covariance matrix, observation matrix, and process noise covariance matrix respectively. The superscript $^-$ indicates a matrix is before being updated by the Kalman gain matrix K_t, applicable for the state matrix $\hat{x}_t$ and the posterior error covariance matrix P_t. The subscripts t and $t-1$ distinguish the current and the former step of the Kalman filter. F_t is the state transition matrix and B_t is the control matrix. x_t is

composed of the observation of the obstacle position and velocity, and $\hat{}$ marks the filtered results for x_t. $\hat{x}_t$ equals to the predicted state $\hat{x}_t^-$ if no dynamic obstacle is caught. $\hat{x}_t^-$ is also utilized as the propagated cluster state in the obstacle matching introduced above. Δt is the time interval between each run of the Kalman filter. In the current stage, we assume the moving obstacle performs uniform motion between t_1 and t_2, $a_{t-1} = 0$.

We summarize our proposed dynamic environment perception method in Algorithm 9.

Algorithm 9: Dynamic environment perception

while true: **do**
  Obtain Pcl_{t_1}, Pcl_{t_2} from the point cloud filter, cluster them to OB_1, OB_2
  if it is the first loop **then**
    Initialize the Kalman filters for each cluster in OB_2
  Predict the position of former clusters with the Kalman filters
  for ob_{2k} in OB_2 (k is the iteration number): **do**
    Match ob_{2k} with the predicted clusters
    if successfully match the clusters: **then**
      Estimate the velocity of ob_{2k} with the paired ob_{1j}
      Classify it as static or dynamic and record the class as the history together with the corresponding Kalman filter
      if the cluster marked as static for S_c consecutive times: **then**
        Delete the corresponding Kalman filter, submit ob_{2k} for map fusion
      else if ob_{2k} is dynamic: **then**
        Update the Kalman filter with p_2^m and $\hat{v}_2^m$
  if no dynamic obstacle is found: **then**
    Update the Kalman filter with the predicted state

5.2.3 Ego-motion compensation and neighbor data overlapping

To improve the estimation accuracy, we notice the time gap between the latest point cloud message from the camera and the vehicle state message from the IMU in the flight controller, which is an important detail ignored in the existing works. A constant acceleration

motion model is adopted to describe the vehicle motion in a short period, and the ego-motion compensation can be done with

$$\hat{p}_{cam} = p_{cam} + v_{cam} t_{gap} + \frac{1}{2} a_{cam} t_{gap}^2, \tag{5.8}$$

to result in the compensated camera pose $\hat{p}_{cam}$. p_{cam}, v_{cam} and $a_{cam} \in \mathbb{R}^{2\times 3}$ are the pose, velocity and acceleration of the camera obtained from raw data (translational and rotational motion), translated from the installation matrix of the camera. t_{gap} is the time gap between the message, equals to the timestamp of point cloud minus the timestamp of vehicle state. As a result, the point cloud can be converted to $E-XYZ$ more precisely.

For non-rigid moving obstacles, for example, walking animals (including humans), the body posture is continuously changing. The point cloud deformation may cause additional position and velocity estimation error of obstacle since the waving limbs of a walking human to interfere with the current estimation measurements. We notice that when two neighbor frames of point cloud are overlaid, the point cloud of the human trunk is denser than the other parts which rotate over the trunk. Then, an appropriate point density threshold of DBSCAN can remove the points corresponding to the limbs. So the overlapped point cloud can replace the filtered raw point cloud. For instance, the w^{th} point cloud frame Pcl_w is replaced with $Pcl_w \cup Pcl_{w-1}$. $\hat{p}_{cam}$ is also replaced by the mean of the value from its neighbor data frame, to align with the point cloud.

5.3 Motion Planning

The motion of the drone is more aggressive for avoiding moving obstacles than flying in a static environment. To address the displacement of the drone during the time costed by the trajectory planner and flight controller, position compensation is adopted before the velocity planning, which is another important detail usually not mentioned in the references. The current position of drone p_n is updated by the prediction

$$\hat{p}_n = p_n + (t_{pl} + t_{ct} + t_{pm}) v_n + \frac{1}{2} (t_{pl} + t_{ct} + t_{pm})^2 a_n, \tag{5.9}$$

where t_{pl} is the time cost of the former step of the motion planner. t_{ct} is the estimated fixed responding time for the flight controller. t_{pm} is the timestamp gap between UAV pose and current time. In addition, due to the time cost of obstacle identification and communication delay, the timestamp on the information of dynamic obstacles is always later than that of the pose and velocity message of the drone. Based on the constant velocity assumption, the obstacle position $\hat{p}_2^m$ in the planner at the current time is predicted and updated as

$$\hat{p}_2^m = p_2^m + (t_{pl} + t_{ct} + t_{pm} + t_{dp})\hat{v}_2^m, \tag{5.10}$$

where t_{dp} is the timestamp gap between the UAV pose and the received dynamic clusters. The dynamic clusters published by the perception module share the same timestamp with the latest point cloud pcl_{t2}.

5.3.1 Velocity planning

This subsection will introduce a novel velocity planning method based on the relative velocity and the forbidden pyramids. First, the planner receives the moving clusters and the velocity from the dynamic perception module. The currently unclassified clusters are also conveyed to the planner and treated as static obstacles together with the classified static clusters. In addition, we adopt the mapping kit to offer the static environmental information out of the current FOV, because the FOV of a single camera is narrow. To tackle the autonomous navigation tasks, the drone is required to reach the goal position. The desired velocity of the drone is initialized as v'_{des}, $\|v'_{des}\|_2 = v_{max}$ and v'_{des} heads towards the goal. v_{max} is the maximum speed constraint. Also, considering the path optimality, a path planner is usually adopted in the navigation. Thus, the waypoint w_p generated from the planned path is used to replace the navigation goal if a path planner is required. Otherwise, w_p denotes the navigation goal. w_p can be generated from a guidance law or assigned directly as the first waypoint in the path to enable the drone to follow the path. It is a choice to combine the velocity planning method with the path planning algorithms to adapt to navigation applications better. Fig. 5.5 illustrates the collision check by calculating the

relative velocity of v'_{des} towards the obstacles, and if the check fails the velocity re-planning will be conducted. For dynamic obstacles, the relative velocity equals v'_{des} minus the obstacle velocity. For static obstacles, the relative velocity is v_n itself. If v'_{des} is checked to be safe, the finally desired velocity v_{des} is given by v'_{des}.

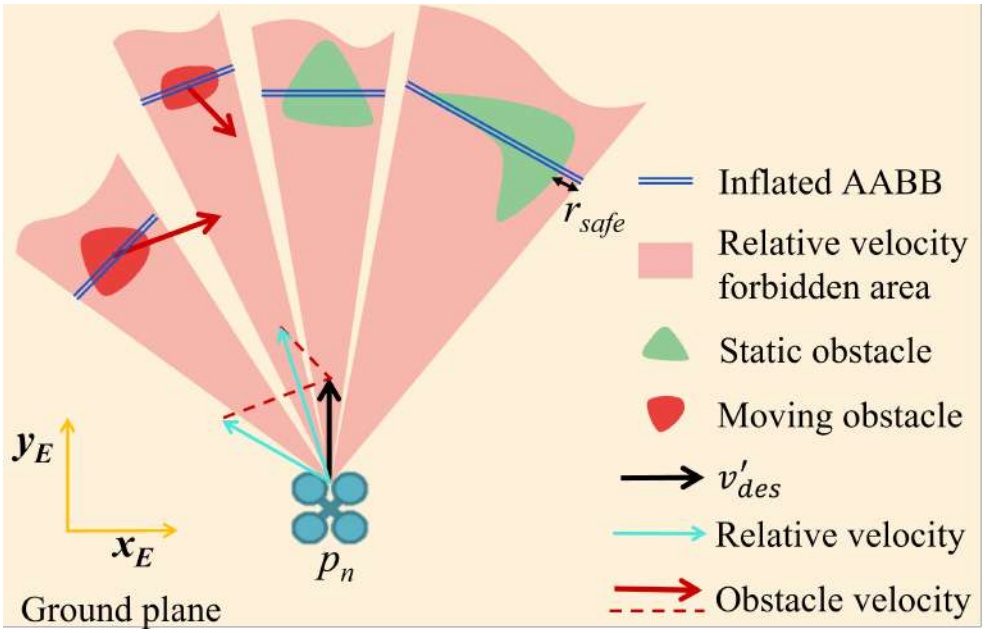


Fig. 5.5.: Check if the current relative velocities towards each obstacle lies in the forbidden area. In this figure, the relative velocity towards one dynamic obstacle and one static obstacle all fail the collision check. The forbidden area is the projection area (space) of the inflated obstacle AABB in the projection plane to the camera. r_{safe} is the inflating size.

Then, the velocity re-planning method is explained in Fig. 5.6-5.8, the forbidden area (space) is extended to a pyramid for the 3D case, different from the triangle for the 2D case. Here, we introduce a hypothesis that the object in the environment does not have a ring topology, thus the flight trajectory through a single object is forbidden. If the relative velocity lies in the corresponding forbidden pyramid, four proposed relative velocity vectors are found by drawing perpendicular lines from the relative velocity vector perpendicular to the four sides of the pyramid. They are the samples to be checked later. The vertical line segments stand for the acceleration cost to control the vehicle to reach the proposed velocity.

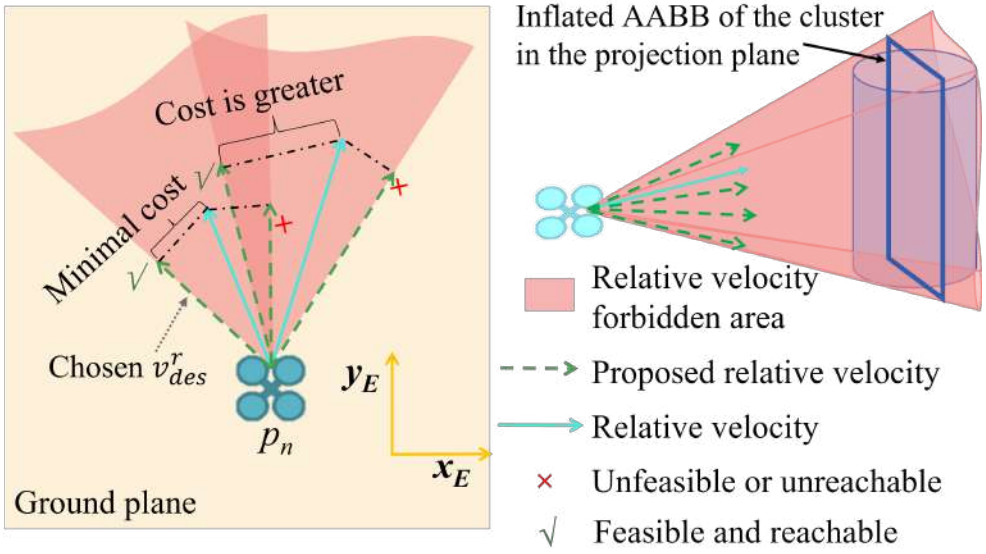


Fig. 5.6.: The left figure explains the velocity planning for multiple forbidden pyramids. We use a floor plan to better demonstrate the method. The right figure is a forbidden pyramid for one obstacle in 3D view, four sides of the pyramid result in four proposed relative velocity vectors. "Unreachable" refers to that a relative velocity is out of the maximal velocity bound of the drone, which is detailed in Fig. 5.8

.

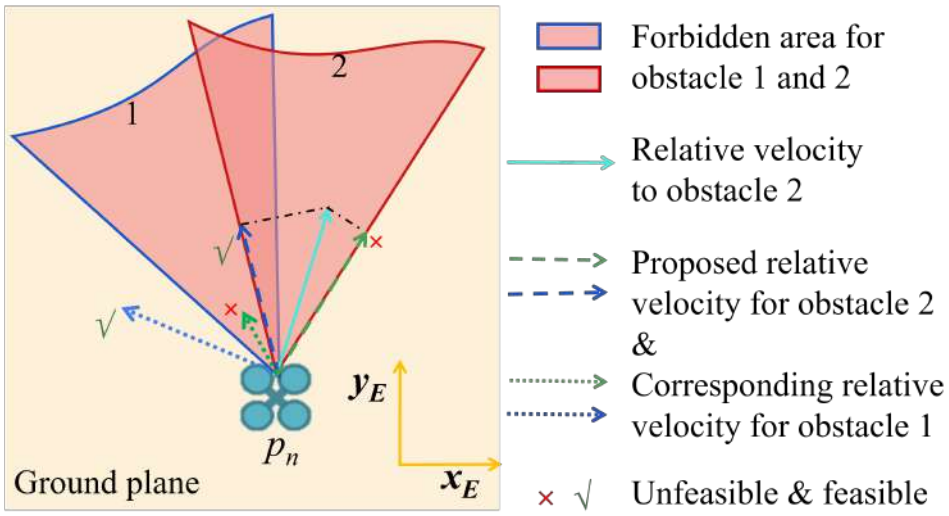


Fig. 5.7.: The feasibility check of the relative velocity for one obstacle, as the supplementary for Fig. 5.6. The proposed relative velocity is checked if feasible for other moving obstacles. In this figure, the left proposed relative velocity for obstacle 2 is also feasible for obstacle 1 (navy blue arrows), while the right one (green arrows) is not.

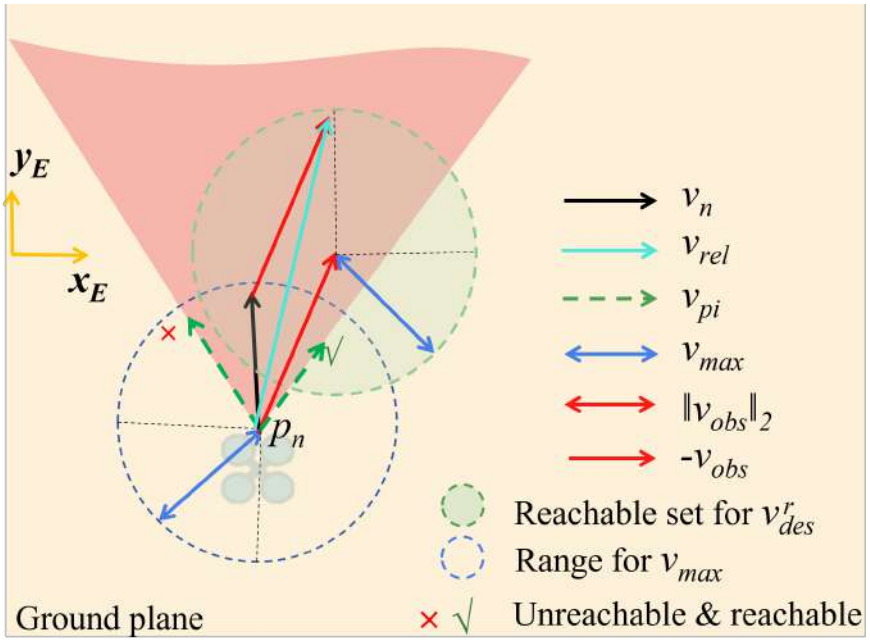


Fig. 5.8.: The reachable check for the proposed relative velocities. v_{obs} is moved to start from p_n, and the endpoint is the center of the spherical reachable set. The possible relative velocity constrained by v_{max} towards this obstacle is included in this set. Only the relative velocity vectors in the reachable set are chosen.

For any cluster, v_{rel} denotes the relative velocity, the five vertices of the forbidden pyramid are

$$\{\hat{p}_n(x_0,y_0,z_0), vt_1(x_1,y_1,z_1), ..., vt_4(x_4,y_4,z_4)\}. \tag{5.11}$$

The acceleration cost a_{ci} of the proposed relative velocity vectors v_{pi} $(i \in \{1,2,3,4\})$ can be calculated by solving the 3D geometric equations, as follows:

$$C^p = \overrightarrow{\hat{p}_n vt_1} \times \overrightarrow{\hat{p}_n vt_2}, \tag{5.12}$$

$$c_d^p = -C^p \hat{p}_n^T, \tag{5.13}$$

$$a_{ci} = \frac{|C^p(v_{rel} + \hat{p}_n)^T + c_d^p|}{\|C^p\|_2}, \tag{5.14}$$

$$v_{pi} = v_{rel} - C^p \frac{C^p (v_{rel} + \hat{p}_n)^T + c_d^p}{\|C^p\|_2^2}. \tag{5.15}$$

In (5.12)-(5.15), triangle $\{\hat{p}_n, vt_1, vt_2\}$ is taken as the example, $C^p[x,y,z]^T + c_d^p = 0$ is the corresponding plane equation, $\hat{p}_n$ is the common vertex of all the 4 triangles.

Obviously, for only one obstacle, the desired relative velocity with the minimal cost is from the four proposed ones. For multiple obstacles and forbidden pyramids, the desired relative velocity is chosen by comparing the feasibility, reachability, and cost. Among all the proposed relative velocity vectors, the one checked to be feasible and reachable and with the minimal cost is selected (v_{des}^r), and the desired vehicle velocity $v_{des} = v_{des}^r + v_{obs}$. v_{obs} is the velocity for the corresponding obstacle. Although the globally optimal solution for acceleration cost cannot be guaranteed within the samples, the computation complexity is greatly reduced compared to solving the optimal solution. We use the inflated bounding box because the character radius of the vehicle r_{uav} can not be neglected.

The feasibility check is to guarantee v_{des} is safe for all obstacles, not for only one of them, which is described in Fig. 5.7. Besides the feasibility check, v_{des} should satisfy the maximum speed constraints v_{max}. We introduce the reachable set for the relative velocity vector to check if the proposed relative velocity v_{pi} $(i \in \{1,2,3,4\})$ is reachable, as Fig. 5.8 indicates. For the relative velocity towards one obstacle, $v_{rel} = v_n - v_{obs}$ always hold. v_{rel} is the current relative velocity towards the obstacle, v_{obs} is the obstacle velocity.

In addition, the lag error of the velocity planning caused by the time cost to reach the desired velocity is also considerable. The relative displacement between the moving obstacle and vehicle during this time gap should be estimated, because the forbidden pyramid is also directly related to the relative position. We can assume the solved jerk J_n is very close to its boundary J_{max}, because the time cost t_v is minimized in the optimization problem (5.20). Thus, the time cost is estimated as

$$\begin{aligned} v_n + a_n t_v + \frac{1}{2} J_n t_v^2 &= v_{des} \\ \Rightarrow t_v = \min_{t_v} \{ \| \frac{2(v_{des} - v_n - a_n t_v)}{t_v^2} \|_\infty &= J_{max} \wedge t_v > 0 \}, \end{aligned} \tag{5.16}$$

and the displacement of the vehicle and obstacle is calculated by

$$d_{uav} = v_n t_v + \frac{1}{2} a_n t_v^2 + \frac{1}{6} J_n t_v^3, \tag{5.17}$$

$$d_{obs} = \hat{v}_2^m t_v. \tag{5.18}$$

At last, the estimated displacement d_{uav} and d_{obs} are added to the position after v_{des} is obtained, and a new v_{des} is planned in iteration until it is checked to be safe. The accurate time cost t_v can only be determined after solving the motion optimization problem. However, involving the optimization problem in the iteration will be time-consuming, so we use a closed-form solution as the approximate value. To speed up the convergence, the safety radius is inflated by a small value ε (equivalent to the tolerance in the safety check) to calculate v_{pi}:

$$r_{safe} = r_{uav} + \varepsilon \ (\varepsilon > 0). \tag{5.19}$$

In a situation where the obstacles are too dense, the forbidden area may cover all the space around the vehicle. We first sort all the clusters with the increasing order of distance to p_n, the farther obstacles are considered less threatening for the drone. Then, the last j clusters are excluded, j is the iteration number increasing from 0. Algorithm 10 reveals the process of velocity planning. As a result, the vehicle can always quickly plan the velocity to avoid static and dynamic obstacles and follow the path to meet the different task requirements.

5.3.2 Motion planning

After the desired velocity v_{des} is obtained, it appears as the constraint in the motion planning and will be reached in a short time. The waypoint constraints w_p is also considered to follow the path, as shown in Fig. 5.9.

The optimization problem to obtain motion primitives is constructed as

Algorithm 10: Velocity planning

1: $v'_{des} \leftarrow v_{max} \dfrac{\overrightarrow{\hat{p}_n w_p}}{|w_p - \hat{p}_n|}$
2: **if** v'_{des} is unsafe (Fig. 5.5): **then**
3: $j \leftarrow 0$
4: Sort the clusters with the distance to p_n
5: **while** v_{des} is not found: **do**
6: Remove the last j clusters from original sequence
7: Get all the feasible relative velocity vectors for the remained clusters
8: **if** feasible and reachable relative velocity exist: **then**
9: Choose v^r_{des} with the minimal acceleration cost, and $v_{des} \leftarrow v^r_{des} + v_{obs}$
10: $j \leftarrow j + 1$
11: **repeat**
12: $\hat{p}_n \leftarrow \hat{p}_n + d_{uav}$, $\hat{p}^m_2 \leftarrow \hat{p}^m_2 + d_{obs}$
13: Repeat line 7-10 with updated forbidden pyramids
14: **until** v_{des} is safe
15: **else**
16: $v_{des} \leftarrow v'_{des}$

$$
\begin{aligned}
\min_{J_n, t_v} \quad & \eta_1 t_v + \eta_2 d_{trj} \\
\text{s.t.} \quad & a_{n+1} = a_n + J_n t_v, \\
& v_{des} = v_n + a_n t_v + \frac{1}{2} J_n t_v^2, \\
& d_{trj} = \frac{\|\overrightarrow{\hat{p}_n p_{end}} \times \overrightarrow{w_p p_{end}}\|_2}{|w_p - \hat{p}_n|}, \\
& p_{end} = \hat{p}_n + v_n K t_v + \frac{1}{2} a_n (K t_v)^2 + \frac{1}{6} J_n (K t_v)^3, \\
& 0 < t_v, \ \|a_{n+1}\|_2 \leq a_{\max}, \ \|J_n\|_2 < J_{max},
\end{aligned}
\tag{5.20}
$$

where the jerk of the vehicle J_n is the variable to be optimized. t_v is the time required to reach v_{des}, which is the variable and the optimization object at the same time. a_{n+1} and p_{end} are calculated by the kinematic formula. a_{max} and J_{max} are the kinodynamic constraints of acceleration and jerk of the vehicle respectively. The velocity constraint v_{max} is satisfied in the equality constraint with v_{des}. η_1, η_2 are coefficients. The default values are shown in

Table 5.1. After the desired trajectory piece is solved, a default cascade PID controller of PX4 is utilized to track this trajectory in position, velocity, and acceleration.

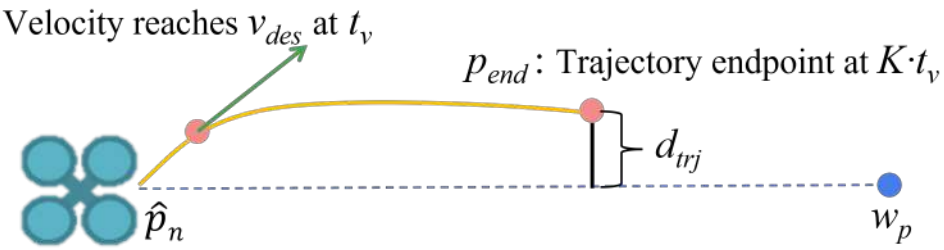


Fig. 5.9.: The proposed motion planning method. The objective function is designed to minimize the time cost to reach the desired velocity and the distance from trajectory endpoint p_{end} to the path line. The solid yellow line represents the predicted trajectory.

5.4 Experimental Implementation and Results

5.4.1 Point cloud filters

For the dynamic environment perception, filtering the raw point cloud is necessary, because the obstacle state estimation is sensitive to the noise. The noise should be eliminated strictly, even losing a few true object points is acceptable. The filter has the same structure as our former work [82], as shown in Fig. 5.10, but the parameters are different. The distance filter removes the points too far ($\geq 6.5\ m$) from the camera, the voxel filter keeps only one point in one fixed-size ($0.1\ m$) voxel, outlier filter removes the point that does not have enough neighbors (≤ 13) in a certain radius ($0.25\ m$). Based on such configuration, the density threshold for DBSCAN is at least 18 points in the radius of $0.3\ m$. These metrics are tuned manually during extensive tests on the hardware platform introduced in the next subsection, to balance the point cloud quality and the depth detection distance. They are proved satisfactory for obstacle position estimation. The point cloud filtering also reduces the message size by one to two orders of magnitude, so the computation efficiency is much improved, while the reliability of the collision check is not affected.

However, when the drone performs an aggressive maneuver, the pose estimation of the camera (including the ego-motion compensation) is not accurate enough for dynamic obsta-

cle perception. To solve this problem, we propose a practical and effective measurement: The filtered point cloud is accepted only when the angular velocity of the three Euler angles of the drone is within the limit ω_{max}.

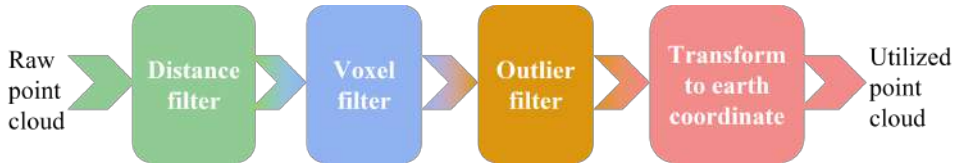


Fig. 5.10.: The filtering process for the raw point cloud.

5.4.2 Map building

In our implementation, we adopt a simple method to store the static points in a list ("mapping kit" in Fig. 5.2), and visualize the points as the point cloud map. After the first dynamic obstacle identification, we push all the static clusters into a list for initialization. When a new static point cluster is found afterward, we compare the distance between the new static cluster center with the existing clusters' centers in the list. If the new cluster is very close to the existing static clusters, it will not be added to the list to avoid duplicating and saving the RAM of the onboard computer. In the velocity planning, only the static clusters in the list in the range of 6.5 m (the cut-off depth of the distance filter) to the vehicle are considered. To obtain a high-quality map, the existing mapping toolkits (such as Octomap) are also capable in our system.

5.4.3 Experimental Configuration

The detection and avoidance of obstacles are tested and verified in the Robot Operation System (ROS)/Gazebo simulation environment first and then in the hardware experiment. The drone model used in the simulation is 3DR IRIS, and the underlying flight controller is the PX4 1.10.1 firmware version. The depth camera model is an Intel Realsense D435i with a resolution of 424*240 (30 fps). For hardware experiments, we use a self-assembled

quadrotor with a Q250 frame and a LattePanda Alpha 864s with an Intel m3-8100y processor, other configuration keeps unchanged. A motion capture system VICON is adopted to obtain the pose of the drone. Table 5.1 shows the parameter settings for the simulation tests. The supplementary videos for the tests have been uploaded online[1][2].

Table 5.1.: Parameters for the tests

Parameter	Value	Parameter	Value
S_c	3	t_{ct}	0.01 s
v_{dy}	0.3 m/s	d_t	0.2 s
t_{kf}	0.7 s	d_m	0.9 m
N_c	12	α	0.5
η_1	10	η_2	6
K	3	ε	0.05 m
a_{max}	6 m/s^2	ω_{max}	1.5 rad/s
J_{max}	12 m/s^3	v_{max}	1.5 m/s

5.4.4 Simulation Test

(1) Dynamic perception module test

First, the accuracy and stability of the estimation method for the obstacle position and velocity are verified.

In the simulation world depicted in Fig. 5.11(a), there is one moving ball, two moving human models, and some static objects. The moving obstacles reciprocate on different straight trajectories. The camera is fixed on the head of the drone, facing forward straightly. Since the point cloud from the simulated sensor is clean and the noise is very light, the distance filter threshold is extended to 8 m. The drone is hovering around the point $(-6, 0, 1.2)$. Fig. 5.11(b) depicts the visualized estimation results in Rviz. The Euclidean distance of the feature vectors utilized for obstacle tracking is illustrated in Fig. 5.12. The estimation

[1] https://youtu.be/1g9vHfoycs0, https://www.bilibili.com/video/BV1oU4y1N7yp?spm_id_from=333.999.0.0

[2] https://www.youtube.com/watch?v=5CwFATodSvU, https://www.bilibili.com/video/BV1T44y1a7rR?spm_id_from=333.999.0.0

numeral results are shown in Fig. 5.13, and they are compared with the ground truth. The dynamic perception performance is also compared with SOTA works in Table 5.2, where the metrics Multiple Object Tracking Precision (MOTP) and Multiple Object Tracking Accuracy (MOTA) are adopted as defined in the work of Bernardin [88]. MOTP is the average position estimation error in this test. Only walking or running pedestrians are tested in Table 5.2. The second line marked with * is for our method without using the track point to correct the velocity estimation, and the third line marked with # is for our method without the neighbor frame overlapping. We record the point cloud (at 30 Hz) and UAV states data (at 100 Hz) for about 600 s, and repeat the test on the data 5 times to give the average results.

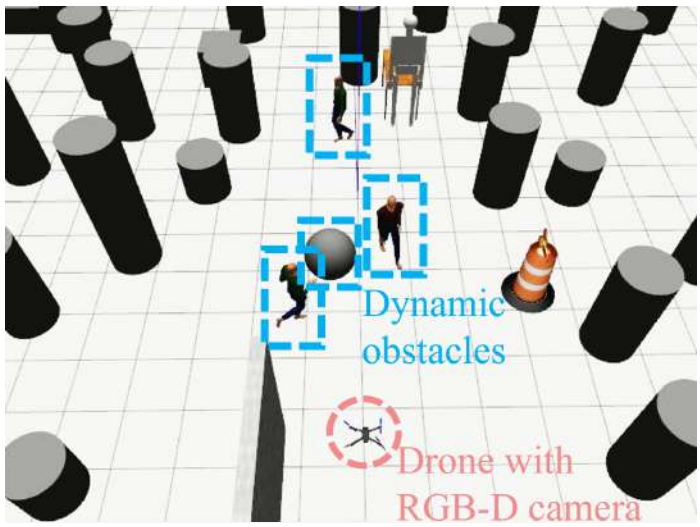
The estimation test results in Gazebo simulation demonstrate that our estimation algorithm is practical for dynamic obstacle avoidance. In addition, our method efficiently improves the estimation accuracy and robustness in the clustered environment. For our MOTA, it is composed of a false negatives rate $f_n = 6.7\%$ (covering non-detected dynamic objects and dynamic objects erroneously classified as static or uncertain), a false positives rate $f_p = 6.9\%$ (static objects misclassified as dynamic), and a mismatch rate $f_m = 2.1\%$.

Table 5.2.: Obstacle State Estimation Comparison

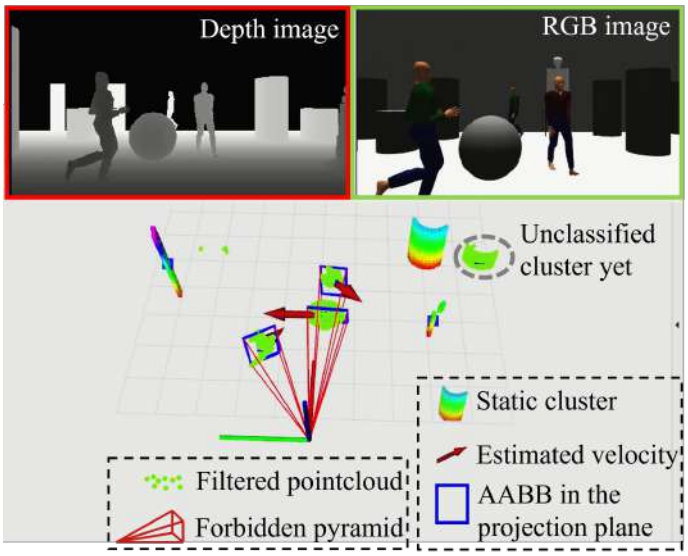
Method	$error_{vel}(m/s)$	MOTA (%)	MOTP
Ours	**0.21**	**84.3**	**0.15**
Ours*	0.29	83.9	0.16
Ours#	0.25	83.6	0.18
[76]	0.37	76.4	0.28
[72]	0.41	70.1	0.30

(2) Motion planning module test

In addition, we compare the motion planning method with [76] and [89] in Table 5.3. The metrics a_{mean}, v_{mean}, l_{traj} and t_{opt} stand for the average acceleration, the average velocity, the average flight trajectory length and the time cost for the motion optimization part. The time costs are measured on a laptop computer with an Intel i7-8565U CPU and 8 GB



(a)



(b)

Fig. 5.11.: (a): The simulation environment for the moving obstacles' position and velocity estimation test. (b): The visualized estimation results in RVIZ, corresponding to (a). Only the forbidden pyramids for dynamic clusters are visualized. The pedestrians always face their moving direction. It can be seen that the obstacles are correctly tracked even though they are very close.

RAM. Similarly in [76], we consider the environment with only dynamic obstacles, because the locations and velocities of all obstacles are known and considered as dynamic in [89]. The obstacles are ellipsoids with human-like size (0.5×0.5×1.8m) and move at constant

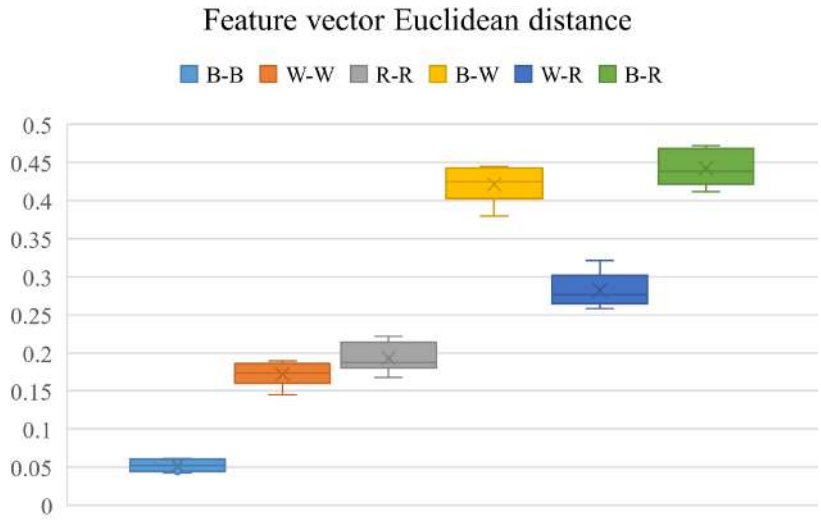


Fig. 5.12.: The box chart of the Euclidean distance of the feature vector $fte()$ between obstacles from OB_1 and OB_2. B, W, and R represent the moving **B**all, **W**alking and **R**unning person in Fig. 5.11 respectively. The distance of the same obstacle is obviously lower than that of different obstacles, so the obstacles are matched correctly.

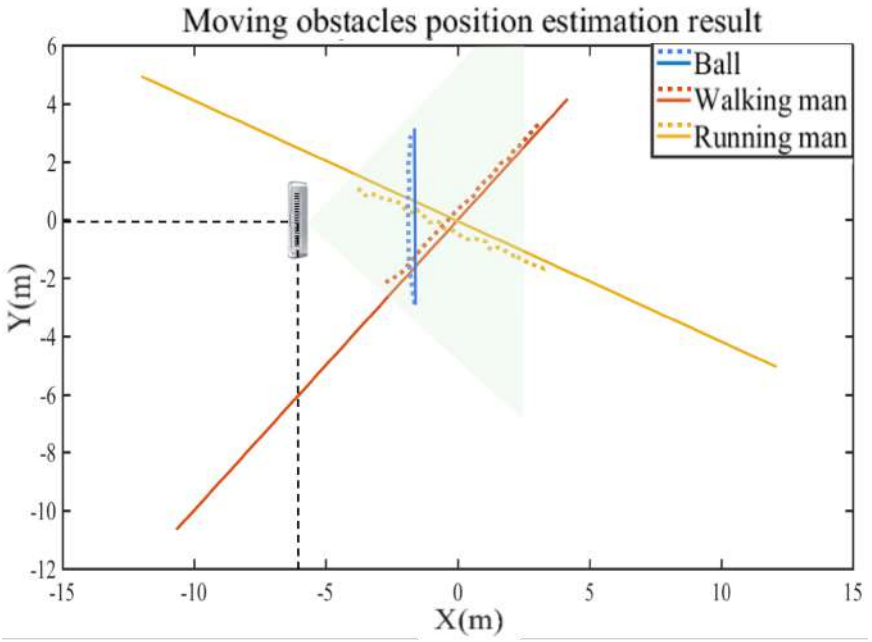


Fig. 5.13.: The estimation results of the moving obstacles' position. The FOV of the camera is represented with a light green area. The dotted line is the estimated result, while the solid line is the ground truth.

Table 5.3.: Dynamic Planning Comparison

Method	$a_{mean}(m/s^2)$	$v_{mean}(m/s)$	$l_{traj}(m)$	$t_{opt}(ms)$
Ours	**2.96**	2.24	25.21	**3.15**
[76]	3.43	2.37	23.65	8.61
[89]	3.18	2.33	22.96	31.23

velocities, as shown in Fig. 5.14. For each planner, the drone flies between two points $(0,0,1.2)$ and $(20,0,1.2)$ back and forth for 10 times, 20 obstacles with velocities at 1-3 m/s cross this straight path disorderly. The camera FOV is also considered, and simulated to be $85.2^\circ \times 58^\circ$ with the maximal sensing depth 8 m. From Table 5.3 we conclude that the average acceleration cost of our motion planning method is smaller because our velocity planning method considers the minimal acceleration cost in all the sample velocities for the current time. Also, the computing time is much shorter thanks to the simple but efficient object function and constraints, which shows the potential to avoid faster obstacles. In these tests, our planning approach produces a longer trajectory because the farther obstacles are more likely to be ignored when the obstacles are dense. Only the obstacle close to the drone is considered sometimes, the trajectory optimality in length from the global view is weak compared to the compared works, as they optimize the trajectory with all the obstacles in the sensing range. In addition, the average speed is smaller because our velocity planning approach is based on sampling and the unreachable samples are simply abandoned, the drone's movement capability is not fully used.

(3) System test

To test the whole framework for navigation tasks, we utilize the HAS method [82] as the path planning algorithm at the front end to generate the waypoint w_p for equation (5.20). The flight simulation world is revealed in Fig. 5.1 and 5.11(a), there are four walking or running pedestrians, one moving ball, and many static pillars and boxes. In Fig. 5.1, the necessity for estimating the obstacle's velocity is illustrated: To avoid the moving man

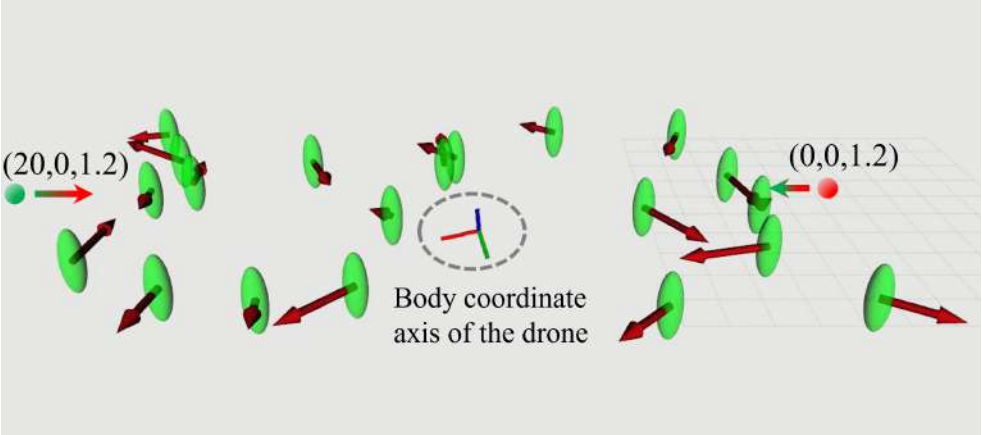


Fig. 5.14.: The simulated test environment for the motion planning module. The drone flies between the two points for the assigned times. The red arrows represent the velocity vectors of dynamic obstacles.

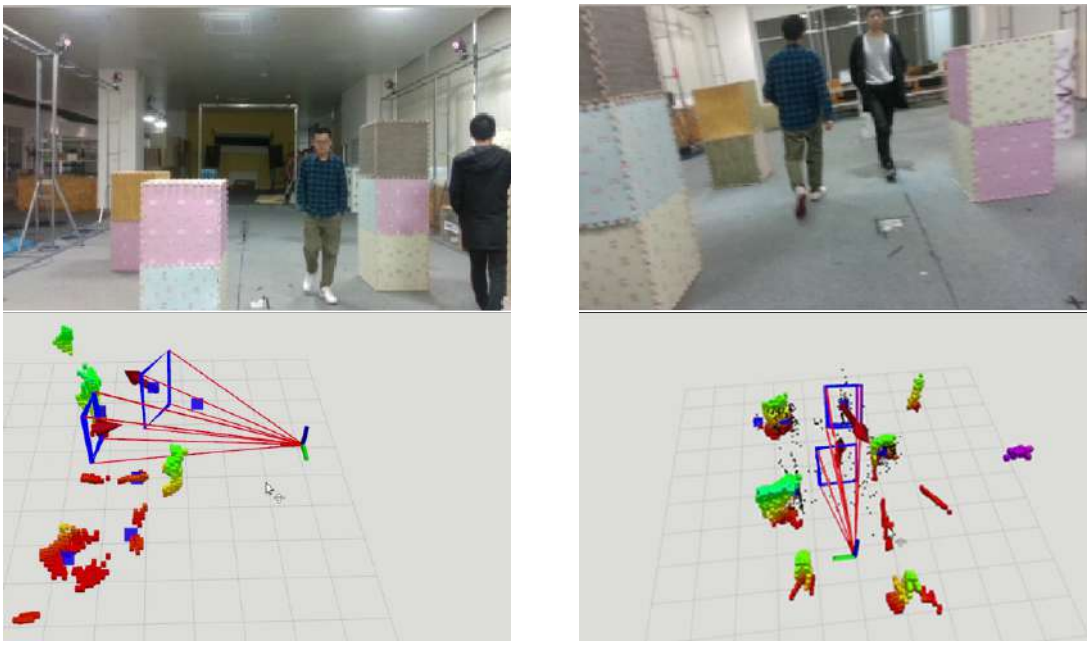
which is at a similar speed to the drone, the aircraft choose to fly in the “opposite” direction from the man so the threat is removed easily. If only the static HAS method [82] is utilized in the same situation, the drone decelerates and flies alongside the man (red line), which is very inefficient and dangerous.

5.4.5 Hardware Test

(1) Perception module evaluation

In the above simulation tests, the proposed perception module is verified with the simulated depth camera. However, the noise of the point cloud from a real depth camera is much more severe. In subsection 5.1, the parameters of the point cloud filters for the real camera are tuned to eliminate the ghost points (the points that do not correspond to any real obstacles), but still a few ghost points remained. Also, the points for real obstacles are not accurate as those in simulation, especially the depth error is greater for the farther objects. The vehicle state estimation also has a greater error than that in simulation, which adds additional error when transforming the points from the body to the earth coordinate. Therefore, some parameters for perception should be adjusted before flight tests. We collect over 430 s data of the raw point cloud (at 30 Hz), raw RGB image (at 30 Hz), and the vehicle states (at 100 Hz) under VICON in different scenarios, and study the influence on the

dynamic obstacle detecting and tracking performance from the parameters. As a result, v_{dy} and d_t are found to be more influential than other parameters. In Table 5.4, we use MOTA (%) to evaluate the performance under different configurations with the collected data, and repeat the test 5 times for each configuration. Other metrics are discarded since the position and velocity ground truth of a moving pedestrian are complex to obtain. (a), (b), (c) in the first row refers to the different test scenarios, and the scenarios are introduced in Fig 5.15.



(a) (b)

Fig. 5.15.: The images from the onboard camera of the dynamic perception test scenarios and the visualized results. Two pedestrians are walking among several boxes and pillars. (a): The camera is fixed, for MOTA(a). (b): The camera is held by hands and moving at around 1.5 m/s and 2.5 m/s, for MOTA(b) and MOTA(c) respectively.

We can conclude from Table 5.4 that greater v_{dy} and d_t are helpful to suppress the noise and depth error in obstacle tracking. However, v_{dy} should be much smaller than the slowest object in the environment to make the classification robust to the velocity estimation error. If d_t is too large, due to the limited camera FOV, an object may be neglected since the continuously observed time is even shorter than d_t. According to the results, we choose $v_{dy} = 0.5$ m/s and $d_t = 0.3$ s, other parameters of the perception module stay unchanged.

Table 5.4.: Obstacle tracking performance under different parameters

v_{dy}(m/s), d_t(s)	MOTA(a)	MOTA(b)	MOTA(c)
0.2, 0.2	69.43	65.26	59.68
0.3, 0.2	71.91	67.71	60.26
0.5, 0.2	75.86	71.12	64.11
0.9, 0.2	74.37	70.59	59.14
0.5, 0.1	71.65	68.24	64.87
0.5, 0.3	78.45	76.12	71.36
0.5, 0.5	74.57	69.11	55.73
0.5, 0.7	66.62	62.78	48.83

(2) Flight test

We set up a hardware test environment as Fig. 5.16, the drone takes off behind the boxes and then a person enters the FOV of the camera and walks straight towards the drone during the flight to test the effectiveness of our method. In Fig. 5.17, the camera is fixed and takes photos every 0.33 s during the flight. Eight photos are composed together. In the first frame the pedestrian appeared, the orange line shows the trajectory of the drone while the yellow line is for the person. It can be concluded that the reaction of the drone is similar to the simulation above. The visualized data for this hardware test is shown in Fig. 5.18, the planned velocity and the predicted trajectory react promptly once the moving obstacle appears. In Fig. 5.19, the gray line is only for the path planning algorithm (HAS), and the blue line represents the whole planning with Algorithm 10. The gray line has a strong positive linear correlation with time because the number of the input points of the collision check procedure determines the distance calculation times. The blue and red lines show the irregularity, because the moving obstacle brings an external computation burden to Algorithm 10, and the number of moving obstacles has no relation to the point cloud size. The single-step time cost of our proposed method (excluding the path planning) is even smaller than 0.01 s, indicating the fast-reacting ability towards moving obstacles.

At last, we compare our work with SOTA works on the system level in Table 5.5. Since most related works differ significantly from ours in terms of application background and test

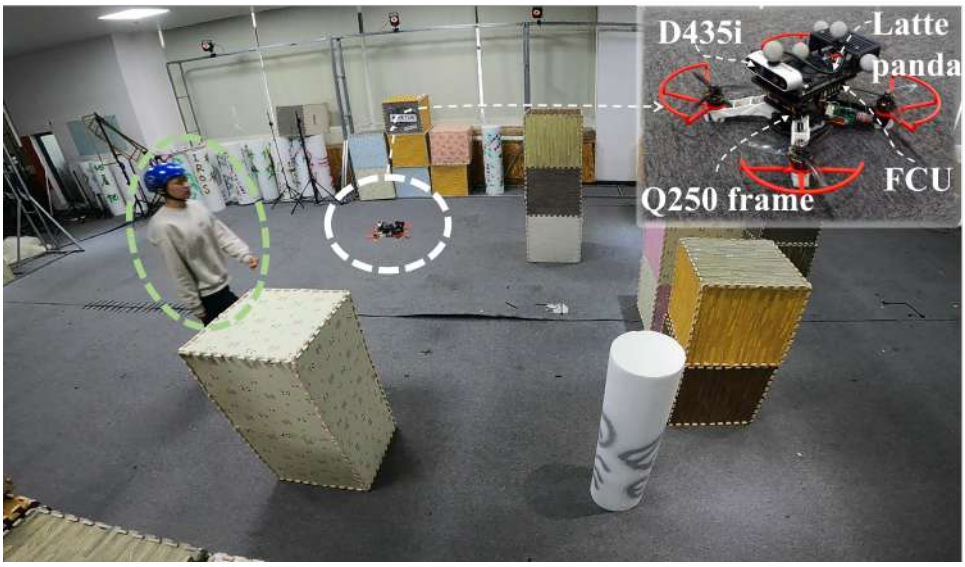


Fig. 5.16.: The dynamic hardware test environment. The aerial platform is introduced in the upper right corner. The pedestrian walks directly through the area while the drone is flying among the static obstacles.

Fig. 5.17.: The composed image of one of the hardware flight tests. The drone takes off from the right side, and the goal is located at the left side, denoted by a green dot. The numbers mark the corresponding frames, increasing with time.

platform, for numeral indicators we only compare the total time cost for reference. The time cost is obtained from the references directly, to compare roughly at orders of magnitude. The abbreviations stand for: obs (obstacle), cam (camera), UUV (underwater unmanned vehicle), N/A (not applicable). “N/A” in the sensor type column refers to the work that gets obstacle information from an external source and does not include environment perception. Most works have severe restrictions on the obstacle type or incompleteness in environment perception, and the computing time cost is not satisfactory for real-time applications. Our

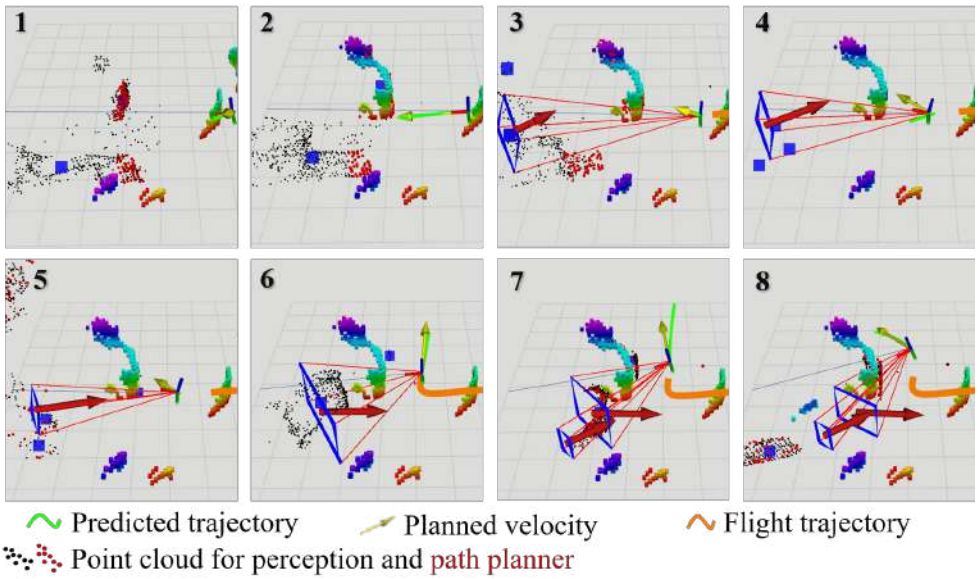


Fig. 5.18.: The corresponding visualized data in RVIZ for the frames in Fig. 5.17

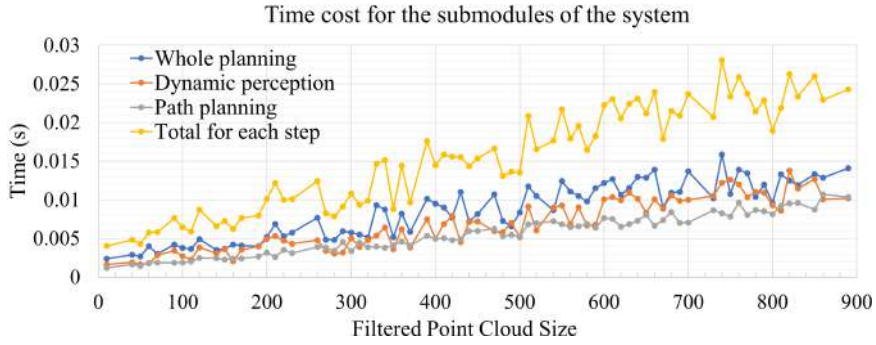


Fig. 5.19.: The time cost for different modules under different filtered point cloud size.

work has a great advantage in generality and system completeness, the computing efficiency is also at the top level.

5.5 Conclusion and Future Work

In this chapter, we present a computationally efficient algorithm framework for both static and dynamic obstacle avoidance for UAVs based only on point clouds. The test results indicate our work is feasible and shows great promise in practical applications. In summary, the main contributions of the chapter are as follows:

Table 5.5.: System comparison between different works

Work	Sensor type	Vehicle	Obs limits	Time cost (s)
[90]	Sonar	UUV	dynamic obs	1-2
[70]	N/A	Robots	dynamic obs	0.045-0.13
[24]	Cam & Lidar	Car	**N/A**	0.1
[16]	N/A	UAV	human	0.2-0.3
[25]	Event cam	UAV	dynamic obs	**0.0035**
[72]	RGB-D cam	UAV	**N/A**	0.024
Ours	RGB-D cam	UAV	**N/A**	**0.015**

- The feature vector is introduced to help match the corresponding obstacle in two point cloud frames. It is proved to be more robust than existing works that match the obstacles with only position information predicted by the Kalman filter. The neighbor frame overlapping and ego-motion compensation techniques are further introduced to reduce the estimating errors of the obstacle's position.

- To compensate for the resultant displacement estimation error from the self-occlusion of obstacles, the object track point is proposed.

- Based on the relative velocity, the forbidden pyramids method is designed to efficiently plan the safe desired velocity to avoid both dynamic and static obstacles. The various time gaps which may cause control lag error are also well compensated.

- We integrate those proposed methods and a path planning method into a complete quadrotor system, demonstrating its reliable performance in flight tests as shown in Fig. 5.17.

However, when the speed or angular velocity of the drone is high, and also because of the narrow FOV of a single camera, the dynamic perception becomes significantly unreliable. In future research, we intend to improve the robustness of our method in aggressive flights and test it with different sensors such as lidar.

6. AN ENHANCED SYSTEM: ROBUST PERCEPTION AND THREAT-AWARE PLANNING

6.1 Research Background

One major potential application scenario of our proposed dynamic obstacle avoidance system is autonomous aerial cinematography (AAC). Microdrones have been broadly applied in aerial cinematography in recent years, and some popular commercial products of cinematography drones are well known to the public, such as DJI Mavic 2 and 3DR Solo Drone. From the viewpoint of the developers of such products, the safety of the drone's flight should be the primary concern. For some inexperienced drone users, flying in complex environments is challenging and dangerous, such as in a narrow indoor scenario with crowds. Thus, to better ensure safety, the drone is supposed to avoid obstacles automatically when encountering urgent incidents, even under the remote control of humans. In addition, autonomous aerial cinematography is a popular function in the most recent photography drone products. Some consumers may want the drone to follow them when doing sports or driving and make a movie. When under the photography tasks, the drones may have to fly in a complex environment, including many dynamic obstacles such as pedestrians. As a consequence, robust autonomous obstacle avoidance ability is urgently required. More importantly, the AAC drones are usually required to interact with people, the crash on the human body will cause serious injury accidents.

After continuous research in recent years, the robust and agile flight in an unknown static environment is no longer a vital challenge. Many existed works have demonstrated the elegant and smooth flight through clustered obstacles. However, autonomous navigation in complex dynamic environments is still recognized as one of the grand challenges in today's robotics [1]. As introduced in Chapter 5, autonomous flight in a dynamic environment is preliminarily achieved. However, the method in Chapter 5 is not safe and robust enough

in a complex dynamic environment. The dynamic object classifier and tracker are not robust to depth information noise. When the camera is moving, the velocity estimation error of static objects is heavy and static obstacles are often wrongly classified as dynamic. Thus, the dynamic environment still severely threatens autonomous vehicles. The motion planning method also shows a deficiency in generating smooth motion primitives for aerial cinematography. To avoid the dynamic obstacles smoothly, we need to detect the dynamic obstacles in advance, robustly distinguish them from the background, and predict the obstacle's trajectory. In addition, the planned trajectory should cover a longer time duration to improve the global optimality and camera stationarity (the unnecessary re-planning is repressed).

Although we have proposed a complete navigation system in a dynamic environment for micro UAVs, the dynamic object perception method embedded is not robust in complex dynamic scenarios because only depth information is used to avoid the heavy computation burden. Under such a condition, the classification of dynamic and static objects can only refer to the estimated velocity of the objects, but the depth map from the RGBD camera is too noisy, and the state estimation of the drone (camera is fixed on the drone) also has an error. As a result, the initial velocity estimation has an unignorable error, and the resulted classification is often wrong, especially since the static object is likely to be classified as dynamic. This is also an untackled problem in related works [72, 76]. To address this problem, we propose an RGB-image-based object classification framework that outputs robust and continuous dynamic object identification results, which is also cheap in computation and real-time on a micro on-board computer. A pointcloud-based object velocity estimation module receives the object classification and outputs the velocity of the dynamic objects only.

In trajectory planning, we adopt a two-stage framework consisting of a front-end kino-dynamic path planner and a back-end trajectory optimizer. Although the existed works use the constant velocity model to predict the obstacle's future trajectory, we should note that it is not accurate. The pedestrians may decelerate or accelerate when facing the drone for their safety considerations. It is also similar to other human-driven vehicles such as cars or

bicycles. Hence, we predict the future position distribution of the objects by their maximum possible acceleration and assign the cost for the trajectory optimization objective function if the predicted vehicle's trajectory conflicts with the obstacle's future potential region.

6.2 Dynamic Object Perception

For autonomous flight in a dynamic environment, the dynamic object perception result plays a vital role in obstacle avoidance performance. Because the classification based on the estimated object velocity is heavily disturbed by the depth noise, we leverage the robust RGB image-based object detecting algorithm to give a reliable category. Then, the velocity of the dynamic object is estimated with the point cloud input after the clustering, and the velocity is further optimized via the Kalman Filters (KFs). The architecture of our proposed framework is illustrated in Fig. 6.1. The black arrows with brief descriptions represent the information flow.

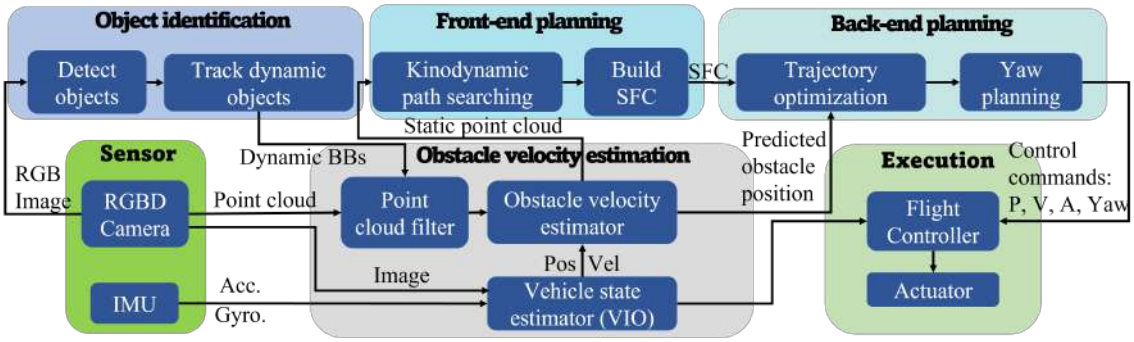


Fig. 6.1.: The flowchart of the whole navigation system.

6.2.1 Detect and track objects on image

In the common scenarios where AAC drones are used, dynamic objects, such as pedestrians or cars, are also common in everyday life. Then, the dynamic objects can be distinguished by prior knowledge with the help of a universal object detector. We list some potential dynamic object types, and once the object detected from the image matches the list, the dynamic object is found and forwarded with the corresponding bounding boxes to the object

tracker. We should note that although the detected "dynamic objects" are not guaranteed moving at the moment, for example, a person sitting on a chair. We argue that the object type that has a considerable probability of being dynamic and may collide with the drone is a valuable target to estimate its velocity.

The object tracker is initialized with the bounding box proposed by the object detector. Since the object tracking framework is designed for multi objects, we initialize $I \in \mathbb{N}$ trackers $\mathbf{T}_{img} = \{\mathscr{T}_0, \cdots, \mathscr{T}_I\}$ for I dynamic objects. $\mathbf{B}_{img} = \{\mathbf{b}_0, \cdots, \mathbf{b}_I\}$ is the list of the bounding boxes of all the tracked targets. Considering that the object tracker may fail during the tracking iterations caused by the object disappearing, we design the recovering mechanism for the tracking failure. Once a tracker fails when it updates with the most recent image, the object detector is executed once to get all the possible $\hat{I}$ dynamic objects again. The $\hat{I}$ newly detected objects is then associated with the currently tracked targets with the IoU, the newly appeared ΔI objects are used to initialize ΔI trackers and update $\mathbf{T}_{img}, \mathbf{B}_{img}, I$, the detected objects associated with the previously tracked object are used to update the ROI of the corresponding trackers. For each tracker in $\mathbf{T}_{img}$, we assign a unique sequence number increasing from zero when it is initialized, and it also inherits the object type tag from the object detector. The image-based object detecting and tracking framework is detailed in Algorithm 11.

In this chapter, a recent light version developed from YOLO object detector series, YOLO-fastest-V2[1], is utilized because of its significant improvement on the computation efficiency. With little sacrificed accuracy, the run time is shortened by about $80\% - 90\%$ compared to YOLO4-tiny when run on CPU, which is very favorable for tiny onboard computers without GPU. For the object tracking algorithm, we adopt the fDSST algorithm [91], which can adjust the BBs' scale compared to the original KCF method [92] and the run time is also very short on our onboard computer. Because the run time of the detector is much longer than the tracker, we only call the detector when any tracker fails and utilize the tracker to fulfill the "gaps" between the runs of the detector and output continuous tracking results, instead of the "tracking by detecting" approach. The trackers are stored in a

[1] `https://github.com/dog-qiuqiu/Yolo-FastestV2`

Algorithm 11: Image-based object detecting and tracking

Input: Raw RGB image $\mathscr{F}$
if Initial run **then**
  Detect dynamic objects with $\mathscr{F}$, initialize I trackers $\mathbf{T}_{img}$ and the bounding boxes (BBs) $\mathbf{B}_{img}$
else
  Update $\mathbf{T}_{img}$ and $\mathbf{B}_{img}$ with $\mathscr{F}$
  if any tracker in $\mathbf{T}_{img}$ fails **then**
    Delete the failed trackers and the BBs from $\mathbf{T}_{img}$ and $\mathbf{B}_{img}$
    Detect $\hat{I}$ objects with $\mathscr{F}$, denoted as $\hat{\mathbf{B}}_{img}$
    for $\hat{\mathbf{b}}_i$ in $\hat{\mathbf{B}}_{img}$ **do**
      Calculate the maximal IoU m_{iou} between $\hat{\mathbf{b}}_i$ and $\mathbf{b}_{mc} \in \mathbf{B}_{img}$
      if $m_{iou} < l_{iou}$ **then**
        Initialize one new tracker with $\hat{\mathbf{b}}_i$, push it in $\mathbf{T}_{img}$ and push $\hat{\mathbf{b}}_i$ in $\mathbf{B}_{img}$
      else
        Update the tracker (corresponding to $\mathbf{b}_{mc}$) with the accurate BB $\hat{\mathbf{b}}_i$
  Publish $\mathbf{B}_{img}$ and the corresponding sequence number and object type.

container after initialized from the newly detected object, and each tracker always updates the object bounding box and confidence indicator when a new RGB frame is input. If the confidence (the peak value of the correlation matrix of the object) is smaller than a threshold, the tracker is considered to fail and deleted from the container.

6.2.2 Estimate the object velocity

The object velocity can be estimated with the help of depth information. The RGBD camera can provide the aligned depth map with the RGB image. The pixel's value corresponds to the real depth multiplied by a scale factor for each pixel in the depth map. The object's 3D position in the camera coordinates can be calculated by the camera intrinsics and the 2D coordinate in the depth map.

We use $\mathscr{F}_{t1}$ and $\mathscr{F}_{t2}$ to represent two depth map frames at the former timestamp t_1 and the latest timestamp t_2 respectively. $t_2 - t_1 = d_t$ and $d_t > 0$. The time interval d_t is determined in the same way as introduced in the last chapter. $\mathscr{F}_{t1}$ and $\mathscr{F}_{t2}$ are updated continuously while the sensor is operating. To utilize the object classification (introduced in section 6.2.1)

and reduce the computation to our best, the depth map is segmented into different dynamic objects by the object BBs and static background (static obstacles) first and then converted to the point cloud in the earth coordinate separately. The clustering procession for the whole point cloud, as introduced in [76, 93], is no longer needed, which takes considerable time. We utilize $\mathbf{C}^s$ to denote the point cloud of the static obstacles (background), and $\mathscr{C}^d_{t1} = \{\mathbf{C}^0_{t1}, \cdots, \mathbf{C}^{I-}_{t1}\}$, $\mathscr{C}^d_{t2} = \{\mathbf{C}^0_{t2}, \cdots, \mathbf{C}^I_{t2}\}$ stands for the point clusters of dynamic objects at t_1 and t_2 respectively. $\mathscr{C}^d_{t1}$ is aligned to $\mathscr{C}^d_{t2}$ following the object sequence numbers, if in t_1 frame there is no corresponding object for $\mathbf{C}^i_{t2}$, we remain a blank cluster at the i^{th} position in $\mathscr{C}^d_{t1}$. For each point cluster $\mathbf{C}^i_{t2}$, we build its 3D axis-aligned bounding box (AABB) $\mathbf{b}^i_o$ with origin $o^i_B \in \mathbb{R}^3$ and size $[l^i_x, l^i_y, l^i_z]$. Then, we build a wider observing box $\mathbf{b}^i_{ob}$ out of the AABB with the origin $o^i = o^i_B - l_{dp}$ and size $l^i = [l^i_x, l^i_y, l^i_z] + 2l_{dp}$. $l_{dp} \in \mathbb{R}^{+3}$ is the inflating size of the observing box, which is related to the maximal possible displacement of the dynamic object within d_t.

Here, the observing box is a similar concept to the "window" in the original PIV method. We translate the coordinate of $\mathbf{C}^i_{t2}$ and change its origin at o^i, as well as $\mathbf{C}^i_{t1}$. To perform the cross-correlation algorithm with $\mathbf{C}^i_{t2}$ and $\mathbf{C}^i_{t1}$ and calculate the displacement of the object, we need to convert the point cluster into a discrete space with the same size. Given the voxel size l_v, the discretized coordinate of any point p in $\mathbf{C}^i_{t1}$ or $\mathbf{C}^i_{t2}$ is calculated by $\lfloor (p - o^i)/l_v \rfloor$. Here comes the most important difference between our method to the original PIV method. The original PIV method is developed to estimate the velocity of the particles that physically exist. However, the point in the point cloud does not relate to a physical particle. The point is generated from the depth map, and the depth value of a specific pixel has been post-processed (may result from interpolation algorithms). The depth calculated from the depth map of a point in 3D space may be slightly different in the continuous frames because of the lightning change. Thus, we extend the 3D discrete space into four dimensions, a feature vector $\mathbf{f}_e \in \mathbb{R}^N$ replace the original binary state (0 represents no particle in this pixel, 1 is for the contrary condition). N is the number of local features of the points fall in the same voxel. The feature vector of each voxel is more robust to the noise and error in the

point cloud, and it is proved in the simulation tests. Let $\mathbf{C}_{vx}$ denote the points in any voxel of the discrete space of $\mathbf{B}^i_{ob}$, the feature vector is defined as:

$$\mathbf{f}_e(\mathbf{C}_{vx}) = [len(\mathbf{C}_{vx}), M_C(\mathbf{C}_{vx})], \tag{6.1}$$

where $len()$ is the size of the input cluster, $M_C() \in \mathbb{R}^3$ returns the mean of the RGB value of the input cluster, and $N = 4$. We use two tensors $\mathscr{X}^i_{t2},\ \mathscr{X}^i_{t1} \in \mathbb{R}^{[N^i_x \times N^i_y \times N^i_z \times N]}$ to denote all the feature vectors in the 3D discrete space, corresponding to point cluster $\mathbf{C}^i_{t2}$ and $\mathbf{C}^i_{t1}$ respectively. The displacement of the i^{th} object can be estimated by the cross-correlation algorithm:

$$\begin{aligned}
F^i_{t1}(\omega_x,\omega_y,\omega_z,\omega_f) &= \frac{1}{2\pi}\iiiint \mathscr{X}^i_{t1} e^{j(\omega_x x+\omega_y y+\omega_z z+\omega_f f)} \mathrm{d}\omega_x\,\mathrm{d}\omega_y\,\mathrm{d}\omega_z\,\mathrm{d}\omega_f \\
F^i_{t2}(\omega_x,\omega_y,\omega_z,\omega_f) &= \frac{1}{2\pi}\iiiint \mathscr{X}^i_{t2} e^{j(\omega_x x+\omega_y y+\omega_z z+\omega_f f)} \mathrm{d}\omega_x\,\mathrm{d}\omega_y\,\mathrm{d}\omega_z\,\mathrm{d}\omega_f. \\
R^i_{12}(x,y,z,f) &= \frac{1}{2\pi}\iiiint F^{i*}_{t1} F^i_{t2} e^{-j(\omega_x x+\omega_y y+\omega_z z+\omega_f f)} \mathrm{d}\omega_x\,\mathrm{d}\omega_y\,\mathrm{d}\omega_z\,\mathrm{d}\omega_f
\end{aligned} \tag{6.2}$$

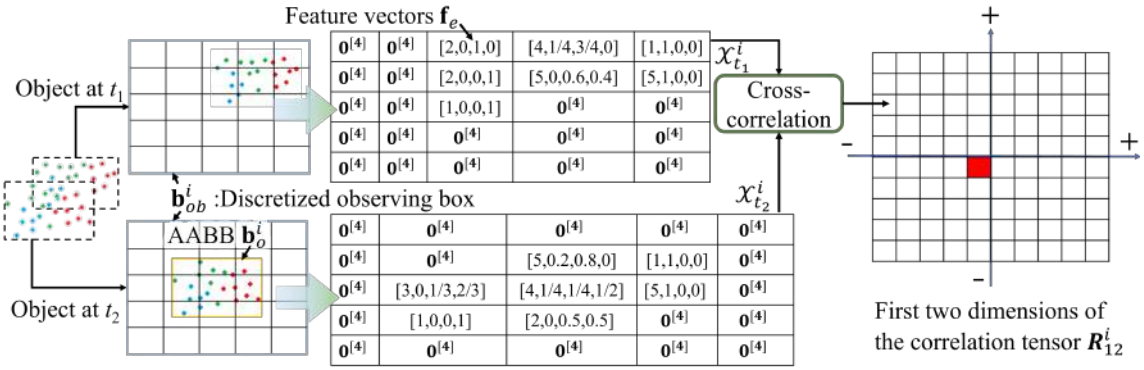


Fig. 6.2.: Illustration of the displacement estimation method. The red grid is the peak in the cross-correlation tensor, whose index represents the displacement of the object. In this case, the displacement of the object from t_1 to t_2 is $[-1,-1]l_v$.

$F^i_{t1}, F^i_{t2} \in \mathbb{C}^{[N^i_x \times N^i_y \times N^i_z \times N]}$ are the resulted complex tensor after the fast Fourier transform (FFT) of $\mathscr{X}^i_{t1}, \mathscr{X}^i_{t2}$. F^{i*}_{t1} is conjugate with F^i_{t1}. $R^i_{12} \in \mathbb{R}^{[N^i_x \times N^i_y \times N^i_z \times N]}$ is the cross-correlation function, it calculates the inverse fast Fourier transform of $F^{i*}_{t1} F^i_{t2}$, and the index d^i_{pk} of

its peak value is the estimated displacement of i^{th} object in the unit of voxel size l_v. Fig. 6.2 shows the process of the displacement estimation with a simple 2D case for a better explanation, while our method is for the 3D case. Then, the velocity can be obtained with

$$v_{t2}^{i} = \frac{d_{pk}^{i} l_v}{d_t}. \tag{6.3}$$

At last, we use KFs to track each dynamic object according to the sequence number and output the optimal position and estimation for each object. The constant velocity model is used in the state transition equation. The KF will forward the prediction for a short time when no latest observation values can be used to update the KF, dealing with the occasion when the tracking of one object on an image is interrupted by short occlusion. If the tracking on the image can recover in a short time, the update for the corresponding KF can resume. The Kalman filter for one object is detailed as below:

$$\hat{x}_t^{i-} = F_t^i \hat{x}_{t-1}^i \tag{6.4}$$

$$P_t^{i-} = F_t^i P_{t-1}^i (F_t^i)^T + Q, \tag{6.5}$$

$$K_t^i = P_t^{i-} H^T \left(H P_t^{i-} H^T + R \right)^{-1}, \tag{6.6}$$

$$P_t^i = \left(I - K_t^i H \right) P_t^{i-}, \tag{6.7}$$

$$\hat{x}_t^i = \begin{cases} \hat{x}_t^{i-} + K_t^i \left(x_t^i - H \hat{x}_t^{i-} \right) \text{ (observation is available)}, \\ \hat{x}_t^{i-} \text{ (no observation)}, \end{cases} \tag{6.8}$$

$$x_t^i = \begin{bmatrix} p_{t2}^i \\ v_{t2}^i \end{bmatrix}, F_t^i = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}, \tag{6.9}$$

where R, H, Q are the observation noise covariance matrix, observation matrix, and process noise covariance matrix respectively. The superscript $^-$ indicates a matrix is before being updated by the Kalman gain matrix K_t^i, applicable for the state matrix $\hat{x}_t^i$ and the posterior error covariance matrix P_t^i. The subscripts t and $t-1$ distinguish the Kalman filter's current and former steps of the Kalman filter. F_t^i is the state transition matrix. x_t^i is composed of the

observation of the obstacle position and velocity, and $\hat{}$ marks the filtered results for x_t^i. $\hat{x}_t^i$ equals to the predicted state $\hat{x}_t^{i-}$ if no dynamic obstacle is caught. $\hat{x}_t^{i-}$ is also utilized as the propagated cluster state in the obstacle matching introduced above. Δt is the time interval between each run of the Kalman filter. The utilization of KF and the related symbols are the same as chapter 5, we repeat them for the reader's convenience. A minor difference is that we match the KFs with the observation value at each depth frame with a more strict condition: The matching is accepted if the observed object position and velocity is close (Euclidean metric) to one forwarded KF, and the observed object ID is identical to one KF's ID. This is a more robust measurement when the object tracker fail to give the correct tracking results. For example, when a tracked object moves outside the camera FOV, the tracker may wrongly keep tracking on some background pattern.

The whole algorithm for object velocity estimation is summarized in Algorithm 12.

6.2.3 Uncertainty evaluation

The KF requires a reasonable estimation of observation noise covariance matrix $R \in \mathbb{R}^2$ (assume the error is isotropic), since the depth noise from the camera is heavy, and it has a severe impact on the position and velocity estimation based on our former tests. First, we analyze the error in pointcloud cluster $\mathbf{C}_{t2}^i$ in the earth coordinate and estimate the variance Var_p^i in the observed object position p_{t2}^i.

Var_p^i mainly comes from three aspects, the depth map variance Var_{p1}^i of the object i, the variance in the vehicle's state Var_{p2}^i (because the error is transferred into $\mathbf{C}_{t2}^i$ during the coordinate conversion from body frame to earth frame), and the variance caused by the changing ego-occlusion and the object body deformation (the waving limbs of a moving person). According to the official document of Intel Realsense D435i depth camera, the standard deviation of depth increases with the measuring distance, as shown in Fig. 6.3. As introduced in the document, the standard deviation of depth is approximately proportional to the square of the distance, thus we can fit the curve with a parabola and estimate Var_{p1}^i with

Algorithm 12: 3D Object Velocimetry

Input: Depth maps $\mathscr{F}_{t1}$ and $\mathscr{F}_{t2}$, Drone's position, orientation, Dynamic Object BBs $\mathbf{B}$ and the object types and sequence number at time t_1 and t_2

Segment $\mathscr{F}_{t1}$ and $\mathscr{F}_{t2}$ with bounding boxes $\mathbf{B}_{t1}$, $\mathbf{B}_{t2}$
Convert the depth maps into the point cloud clusters $\mathbf{C}^s$, $\mathscr{C}^d_{t1}$, $\mathscr{C}^d_{t2}$ in the earth coordinate
Re-sort $\mathscr{C}^d_{t1}$ to align with $\mathscr{C}^d_{t2}$ by the object sequence number, leave empty at position i in $\mathscr{C}^d_{t1}$ if at t_1 no object can match $\mathbf{C}^i_{t2}$.
for i in range 0 to I **do**
  if $\mathbf{C}^i_{t1}$ is empty **then**
    if the KF of object i is timeout **then**
      Delete the KF of object i
    else
      Update the KF with the predicted values in the last step as the observation (equation (6.7))
    Continue to the next loop
  else
    Build the observing box $\mathbf{B}^i_{ob}$ for $\mathbf{C}^i_{t2}$
    Discretize the space of $\mathbf{B}^i_{ob}$, calculate the integer index of each point in $\mathbf{C}^i_{t1}$ and $\mathbf{C}^i_{t2}$ in the subspace $\mathbf{B}^i_{ob}$
    Calculate all the feature vectors for each voxel in $\mathbf{B}^i_{ob}$, for both $\mathbf{C}^i_{t2}$, $\mathbf{C}^i_{t2}$, build $\mathscr{X}^i_{t2}$, $\mathscr{X}^i_{t1}$
    Estimate the velocity, as detailed in equation (6.2) and (6.3)
    if object i's observation cannot match with existed KFs **then**
      Initialize the KF for object i with the observed state x^i_t, $\hat{x}^i_t \leftarrow x^i_t$
    else
      Update the KF with the observation values, obtain the optimal prediction $\hat{x}^i_t$
Publish $\hat{x}^i_t$ as the position and velocity of object i, as well as the object type, with timestamp t_2

$$Var^i_{p1} = (0.00375\|p^i_{t2} - p_{c0}\|^2)^2, \tag{6.10}$$

where p_{c0} denotes the current position of the camera. Var^i_{p2} is composed of the estimation error of both vehicle orientation and position. The position estimation variance Var_{pos} can be directly accumulated in Var^i_{p2}, while the error caused by orientation estimation variance Var_{ori} is calculated by the error propagation law:

$$Var^i_{p2} = Var^{3\times1}_{pos} + R^{3\times3}_{er} Var^{3\times1}_{ori}, \tag{6.11}$$

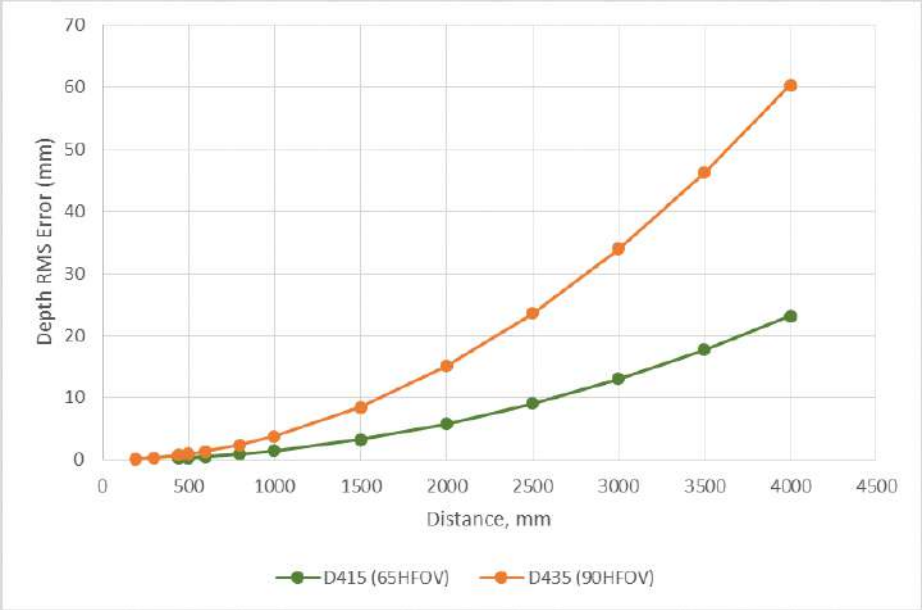


Fig. 6.3.: The relationship between depth standard deviation and the distance. This picture is from the Intel official online document.

$$R_{er}^{3\times 3} = \left[R_{er}^{1}\, R_{er}^{2}\, R_{er}^{3}\right] \tag{6.12}$$

$$R_{er}^{1} = \begin{bmatrix} ((C\psi S\theta C\phi + S\psi S\phi)p_B^i[1] + (-C\psi S\theta S\phi + S\psi C\phi)p_B^i[2]) \\ ((S\psi S\theta C\phi - C\psi S\phi)p_B^i[1] + (-S\psi S\theta S\phi - C\psi C\phi)p_B^i[2]) \\ C\theta C\phi\, p_B^i[1] - C\theta S\phi\, p_B^i[2] \end{bmatrix}, \tag{6.13}$$

$$R_{er}^{2} = \begin{bmatrix} (-C\psi S\theta\, p_B^i[0] + C\psi C\theta S\phi\, p_B^i[1] + C\psi C\theta C\phi\, p_B^i[2]) \\ (-S\psi S\theta\, p_B^i[0] + S\psi C\theta S\phi\, p_B^i[1] + S\psi C\theta C\phi\, p_B^i[2]) \\ -C\theta\, p_B^i[0] - S\theta S\phi\, p_B^i[1] - S\theta C\phi\, p_B^i[2] \end{bmatrix}, \tag{6.14}$$

$$R_{er}^{3} = \begin{bmatrix} (-S\psi C\theta\, p_B^i[0] + (-S\psi S\theta S\phi - C\psi C\phi)p_B^i[1] + (-S\psi S\theta C\phi + C\psi S\phi)p_B^i[2]) \\ (C\psi C\theta\, p_B^i[0] + (C\psi S\theta S\phi - S\psi C\phi)p_B^i[1] + (C\psi S\theta C\phi + S\psi S\phi)p_B^i[2]) \\ 0 \end{bmatrix}, \tag{6.15}$$

where $C()$ is short for $cos()$ and $S()$ is short for $sin()$. $p_B^i \in \mathbb{R}^3$ is the body coordinate of p_{t2}^i. Here, please note that the variance of vehicle position and orientation estimation is assumed to obtain from the localization submodule. If a KF or EKF is used for localization,

the variance we need can be directly copied from its posterior error covariance matrix. For Var^i_{p3}, it can be estimated by analyzing the data from Gazebo simulation since the ground truth value can be easily collected. At last, we have the variance of object position estimation

$$Var^i_p = Var^i_{p3} + mean(Var^i_{p2}) + Var^i_{p1}. \tag{6.16}$$

The observed velocity variance Var^i_v is also related to position variance Var^i_p, the voxel size l_v in our proposed object velocity method also contribute to Var^i_v. Since the displacement result is also discretized by l_v, the result will loose the precision and an additional displacement error of variance $(l_v)^2/12$ (variance of continuous uniform distribution with interval length l_v) is introduced. According to the error propagation law, we have

$$Var^i_v = 2Var^i_p(\frac{1}{d_t^2} - 1) + \frac{(l_v)^2}{12}. \tag{6.17}$$

Thus, the matrix R is obtained:

$$R = \begin{bmatrix} Var^i_p & \sqrt{Var^i_p Var^i_v} \\ \sqrt{Var^i_p Var^i_v} & Var^i_v \end{bmatrix}. \tag{6.18}$$

At last, please be reminded that we use the constant velocity model in KF to estimate the dynamic obstacle's motion. However, it is not precise for real-world common moving objects. The velocity fluctuation is very common such as pedestrians. For the process error covariance matrix Q, we should consider the disturbance from the unmodeled factor, such as the acceleration variance $Var^i_a \in \mathbb{R}$[79]. If we assume Var^i_a is known or can be roughly estimated for a specific type of dynamic object, then matrix Q is given by

$$Q = \begin{bmatrix} \frac{\delta t^2}{4} & \frac{\delta t^3}{2} \\ \frac{\delta t^3}{2} & \delta t^2 \end{bmatrix} Var^i_a. \tag{6.19}$$

Let's take pedestrians as an example. As illustrated in one research[94], the total acceleration root means square (RMS) of a walking human is about $3.15 m/s^2$, for a running

human, the RMS is about $3.72m/s^2$. Thus, we can approximately pre-assign $Var_a^i = 10$ for "person" object to give matrix Q the reasonable value.

6.3 Trajectory Planning

The trajectory planning framework comprises three major parts: front-end path planning, back-end trajectory optimization, and yaw planning. If we plan the initial trajectory at the beginning of the flight, we first run the front-end path planning (hybrid A*) algorithm to obtain a kinodynamic-feasible path for the drone. Then, we build a safe flight corridor (SFC) based on the path. The SFC comprises a series of convex polyhedras connected end to end, and no static obstacle is in the SFC. SFC can give concise geometry constraints to facilitate trajectory optimization, and the optimized trajectory is supposed to stay inside the SFC to avoid all the static obstacles. The predicted trajectories of the dynamic objects are converted to the geometry constraints in the optimization. At last, the yaw along the trajectory is planned to improve the obstacle visibility, and ultimately achieve the purpose of improving flight safety.

To re-plan the trajectory, we design a hierarchical safety check mechanism in the framework to minimize the computation burden. Fig. 6.4 illustrates the trajectory re-planning framework after the initial trajectory has been obtained. The results from the last planning step are well reused in the current circle. The hybrid A* planning, SFC building, and trajectory planning are called only when the corresponding safety check fails (collision detected). We adopt the KD-tree for quick collision check in the hybrid A* planning and SFC building on the point cloud, and the KD-tree construction with the point cloud in the local environment is also very fast [95].

6.3.1 FOV constrained hybrid A* algorithm

The 3D hybrid A* algorithm [48] can search a kinodynamic-feasible path efficiently, and the kinodynamic path contains the primary coarse spatial-temporal information to predict

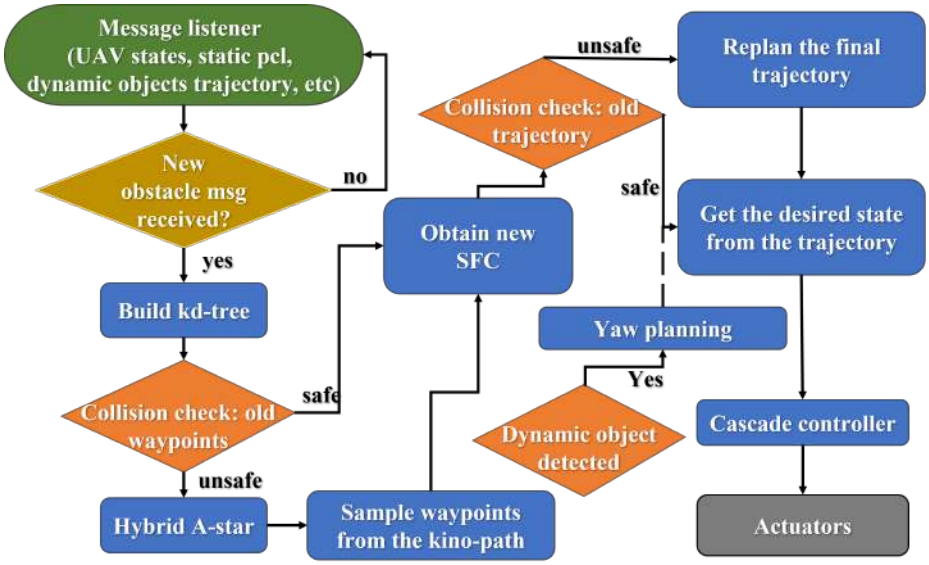


Fig. 6.4.: The flowchart of the re-planning framework

the vehicle position at any future time, thus we can check the path collision with dynamic objects as well.

Given the initial vehicle state (the position and velocity) $[p_{init}, \dot{p}_{init}]$ and goal state $[p_{goal}, \dot{p}_{goal}]$, the hybrid A* algorithm adopts the acceleration as the control input. With a fixed time interval T_{node} and the samples in the control space, the algorithm transfer the initial state to many middle states (nodes). The control sampling and state transition are repeated with the nodes within time horizon T_{node}, until a node is close enough to the goal state. The algorithm inherits a similar operation with an open list and close list from A* algorithm, and adopts the heuristic cost function to speed up the convergence as well. We propose modifications to the original hybrid A* algorithm [48], enabling it to work directly on the point cloud with the dynamic objects existing, and the resulted path can be limited in the camera FOV to enhance safety. The modified **CheckFeasible()** function in the original Algorithm 1 of [48] is described in Algorithm 13. n_{sp} is the sample number in T_{node} for safety check.

The camera FOV is considered as the soft constraint in the path searching, we add an additional large cost c_{fov} in the original **Heuristic()** function [48] when the node is found outside the current FOV of the camera. A regular quadrangular pyramid can represent the

Algorithm 13: Modified functions of hybrid A*

CheckFeasible(): Input: Current state (node) $[p_c, \dot{p}_c]$, current control sample a_c, state $[p_{tmp}, \dot{p}_{tmp}]$ in the open list tries to connect to $[p_c, \dot{p}_c]$, kd-tree $\mathscr{K}$, Dynamic objects' position $\{p_{t2}^0, \cdots, p_{t2}^I\}$ and velocity $\{v_{t2}^0, \cdots, v_{t2}^I\}$.

```
t_c ← current global time
for t_s in [T_node/n_sp, 2T_node/n_sp, ..., T_node] do
    Query position p_c + ṗ_c t_s + a_c t_s^2/2 in 𝒦, d_m ← the distance to the nearest neighbor
    if d_m < d_safe ∨ ‖ṗ_c + a_c t_s‖ > v_max then
        Return false
    else
        for i in range [0,I] do
            if ‖p_t2^i + v_t2^i (t_c − t_2 + t_s) − (p_c + ṗ_c t_s + a_c t_s^2/2)‖ < d_safe then
                Return false
Return true
```

camera FOV. Let the current position of the camera p_{c0} be one vertex of the pyramid. p_{c1} to p_{c4} represent the other four vertexes, and p_{c0} to p_{c4} are in the earth coordinate. We first use the combination of vector inner and outer products to judge one point p_{tmp} is at the front or back side of a plane:

$$d_{f0} = (p_{tmp} - p_{c0}) \cdot ((p_{c0} - p_{c1}) \times (p_{c0} - p_{c2})). \tag{6.20}$$

If $d_{f0} \in \mathbb{R}$ is negative, p_{tmp} is at the inner side of plane $\triangle p_{c0} p_{c1} p_{c2}$ of the pyramid. We calculate d_{f1} to d_{f4} for the other four sides of the pyramid as (6.9), and p_{tmp} is outside the FOV if one of d_{f0} to d_{f4} is positive. If p_{tmp} is outside the FOV, an additional cost is added:

$$h_c = \bar{h}_c + c_{fov} \tag{6.21}$$

h_c is the modified heuristic cost we use, $\bar{h}_c$ is the original in [48]. We do not strictly forbid the planned path to go beyond the FOV, considering the drone may face wide obstacles occupying all the FOV and block the way to the goal. Our approach will find the path inside the FOV if possible and try the unknown space outside the FOV when the FOV is fully occupied.

6.3.2 Trajectory optimization and the time-varying safety margin

Assume that we have the second order kinodynamic path, the path is discretized into a series waypoints and the waypoints are streamlined, denoted as $\mathscr{P}_{ori} = \{wp_1, wp_2, \cdots, wp_m\}$. For any wp_{m_*}. $wp_{m_*} \in \mathscr{P}_{sp}(0 < m_* < m)$, $wp_{m_*-1}wp_{m_*+1}$ collides with obstacles, and $wp_{m_*-1}wp_{m_*}$ and $wp_{m_*}wp_{m_*+1}$ are all collision-free. Before we optimize the trajectory, a safety corridor is required to offer the geometry constraints for the trajectory in order to ensure static obstacle avoidance. In this chapter, we use polyhedrons to construct the SFC rather than spheres because more free space is utilized. The safety corridor generation procedure is the same as described in [96]. The polyhedron $\mathscr{H}_j$ $(0 < j < m)$ in the SFC is defined as:

$$\mathscr{H}_j = \left\{ \mathbf{x} \in \mathbb{R}^3 \mid \left(\mathbf{x} - \hat{p}_j^k\right)^{\mathrm{T}} \vec{n}_j^k \leq 0, k = 1, 2, \cdots, N_j \right\}, \tag{6.22}$$

where N_j is the number of faces of the j^{th} polyhedron, and $\hat{p}_j^k, \vec{n}_j^k$ are the point and the corresponding normal vector on the face.

Thanks to MINCO (minimum control) class [97], we can directly control a trajectory's spatial and temporal profile, making the trajectory optimization with geometry constraints very efficient. We use a series of polynomials to define the trajectory in each dimension (x, y, and z) in 3D space, denoted as:

$$\mathbf{p}(t) = \begin{cases} p_1(t) = \mathbf{C}_1^{\mathrm{T}} \beta\left(t - T_0\right) & T_0 \leq t < T_1 \\ \vdots & \vdots \\ p_j(t) = \mathbf{C}_j^{\mathrm{T}} \beta\left(t - T_{j-1}\right) & T_{j-1} \leq t < T_j \\ \vdots & \vdots \\ p_{m-1}(t) = \mathbf{C}_{m-1}^{\mathrm{T}} \beta\left(t - T_{m-2}\right) & T_{m-2} \leq t < T_{m-1} \end{cases} \tag{6.23}$$

where $\mathbf{C}_j \in \mathbb{R}^{6\times 3}$ is the coefficient matrix of the j^{th} piece and $\beta(t) = \left[1, t, \cdots, t^5\right]^{\mathrm{T}}$ is the time vector. The order of polynomials is 5, and the accelerations at the joint point between polynomial pieces are continuous. Thus, the quadrotor's body angles along the planned

trajectory are continuous according to the differential flatness property. The perception and VIO system all benefit from the smoothly changing body angles because the camera is fixed at the frame, and the jitters in body angles will bring motion blur to the image.

We define the optimization problem formulation as follows:

$$\begin{aligned}
&\min G = S_e + \rho\left(T_{m-1} - T_0\right) + \lambda_v S_v + \lambda_a S_a + \lambda_c S_c + \lambda_d S_d \\
&\text{s.t. } p_j^{[s]}\left(T_j\right) = p_{j+1}^{[s]}(0) = \bar{p}_j, \\
&\quad p_j\left(T_j\right) \in \mathscr{H}_j \cap \mathscr{H}_{j+1}, T_j - T_{j-1} > 0, \\
&\quad \forall j \in \{1, \cdots, m-1\}, \\
&\quad p_1^{[s]}(0) = \bar{p}_s, p_{m-1}^{[s]}\left(T_{m-1}\right) = \bar{p}_f,
\end{aligned} \tag{6.24}$$

where $\mathscr{C}(x) = max(x,0)^3$ is a cubic function, ρ is the weight of the flight time of the whole trajectory. $[s]$ represents a set of derivatives of the highest order s. We choose $s = 2$, so the position, velocity, and acceleration of the joint point between polynomial segments are all equal. $\bar{p}_j \in \mathbb{R}^{(s+1)\times 3}$ is the interval condition, $\bar{p}_s, \bar{p}_f \in \mathbb{R}^{(s+1)\times 3}$ are the start and final states, respectively. S_e, S_v, S_a, S_c, S_d are the cost for the energy, the maximum velocity constraint, the maximum acceleration constraint, the collision with safety corridor, and the collision with dynamic objects respectively. λ_v, λ_a, λ_c, and λ_d are the corresponding weights. We define the time allocation vector $\mathbf{T}$ as:

$$\begin{aligned}
\mathbf{T} &= \left(T_1 - T_0, T_2 - T_1, \cdots, T_{m-1} - T_{m-2}\right)^{\mathrm{T}} \\
&= \left(\delta_1, \delta_2, \cdots, \delta_{m-1}\right)^{\mathrm{T}} \in \mathbb{R}_+^{m-1}.
\end{aligned} \tag{6.25}$$

The detailed definition of the costs are as follows:

$$S_e = \sum_{j=1}^{m-1} \sum_{l=0}^{L-1} \left\| \mathbf{C}_j^{\mathrm{T}} \beta^{(3)}\left(\frac{l}{L}\delta_j\right) \right\|_2^2 \frac{\delta_j}{L}, \tag{6.26}$$

$$S_v = \sum_{j=1}^{m-1} \sum_{l=0}^{L-1} \mathscr{C}\left(\left\| \mathbf{C}_j^{\mathrm{T}} \beta^{(1)}\left(\frac{l}{L}\delta_j\right) \right\|_2^2 - v_{\max}^2 \right) \frac{\delta_j}{L}, \tag{6.27}$$

$$S_a = \sum_{j=1}^{m-1} \sum_{l=0}^{L-1} \mathscr{C} \left(\left\| \mathbf{C}_j^{\mathrm{T}} \beta^{(2)} \left(\frac{l}{L} \delta_j \right) \right\|_2^2 - a_{\max}^2 \right) \frac{\delta_j}{L}, \tag{6.28}$$

$$S_c = \sum_{j=1}^{m-1} \sum_{l=0}^{L-1} \sum_{k=1}^{N_j} \mathscr{C} \left(\left(\mathbf{C}_j^{\mathrm{T}} \beta^{(0)} \left(\frac{l}{L} \delta_j \right) - \hat{p}_j^k \right)^{\mathrm{T}} \vec{n}_j^k \right) \frac{\delta_j}{L}, \tag{6.29}$$

where L is the sample number on each piece of the trajectory, and I is the number of dynamic objects. d_{safe} is the pre-assigned safety radius to guarantee the safety between the vehicle trajectory and the obstacle at any moment. v_{max} and a_{max} are the upper bound of the velocity and acceleration magnitude, respectively. The trajectory optimization is also illustrated in Fig. 6.5, all the sample points along the trajectory will be pushed inside the SFC, and they are guaranteed not to have a spatial-temporal intersection with the predicted dynamic obstacle's position.

Here we introduce the time-varying safety radius toward the dynamic objects. Most existing related works assume the dynamic obstacles move at a constant velocity when planning the trajectory. Since most dynamic obstacles such as pedestrians in the real-world can not suits the constant velocity model precisely, especially when human meet a aerial vehicle their reactive behavior is difficult to predict, the motion uncertainty should be considered carefully. We hence propose a future position distribution estimating method for dynamic objects based on their maximal acceleration from prior knowledge. Let $a_{max}^d \in \mathbb{R}^3$ denotes the maximal acceleration can be reached for a certain object type. At time $t_2 + \Delta t (\Delta t > 0)$, the position of dynamic object i is distributed in

$$\mathscr{S}_{t_2+\Delta t}^i = \left\{ \mathbf{x} \in \mathbb{R}^3 \mid |\mathbf{x}[r] - \hat{p}_d^i[r] - \mathbf{b}_o^i[r]| \leq \frac{a_{max}^d[r](\Delta t^2)}{2} \right\}, \tag{6.30}$$

where $r = 1, 2, 3$ represents the x, y, z coordinate respectively. The dynamic object's predicted position $\hat{p}_d^i$ is calculated by the velocity and the global time t_{opt} when the optimization starts:

$$\hat{p}_d^i = \hat{p}_{t2}^i + \hat{v}_{t2}^i (t_{opt} - t_2 + \frac{l}{L} \delta_j + \sum_{q=1}^{j-1} \delta_q). \tag{6.31}$$

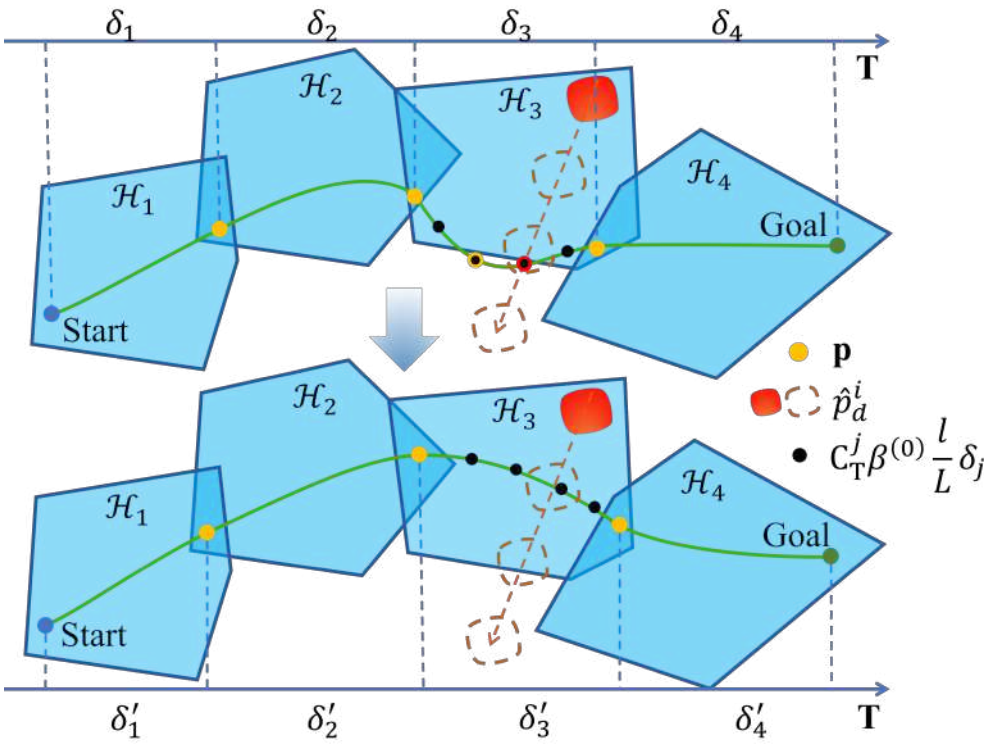


Fig. 6.5.: The process of trajectory optimization. Before the optimization (upper case), the sample points on the trajectory (black dots) are found outside the SFC (with yellow circle) and collides with dynamic object (with red circle). The four trajectory samples correspond to the four predicted object positions. After the optimization, the joint points (yellow dots) of the trajectory and the time allocatinon **T** are adjusted to deform the trajectory and make it safe.

If there is the spatial-temporal intersection between the vehicle's future trajectory and $\mathscr{S}^i_{t_2+\Delta t}$, the vehicle is considered of collision risk. The distributed space is of cuboid shape with the half-edge lengths $l^i_{\Delta t} = a^d_{max}(\Delta t^2)/2 + l^i/2 \in \mathbb{R}^3$, as shown in Fig. 6.5. We assume that the position closer to p^i_d has a higher probability of being occupied by the object i at time $t_2 + \Delta t$, S_d is defined as ($\odot$ is Hadamard product)

$$S_d = \sum_{j=1}^{m-1} \sum_{l=0}^{L-1} \sum_{i=1}^{I} \left(\lambda_{ds} \mathscr{C} \left(1 - \left\| v_d \odot^{-} l^i_{\Delta t} \right\|_2^2 \right) + \mathscr{C} \left(1 - \left\| v_d \odot^{-} \hat{l}^i \right\|_2^2 \right) \right) \frac{\delta_j}{L} \tag{6.32}$$

where $^{-}l^i_{\Delta t}$ represents a vector has the reciprocals of each element in $l^i_{\Delta t}$, the same for $\hat{l}^i$. To make the formula more concise, we define the distance vector $v_d = \mathbf{C}_j^{\mathrm{T}}\beta^{(0)}\left(\frac{l}{L}\delta_j\right) - \hat{p}_b^i$. $\hat{l}^i$ is the inflated AABB obtained by further introduce the safety margin d_s into the original object 3D AABB of size $\hat{l}^i = [l_x^i, l_y^i, l_z^i] + 2d_s$. $d_s = e_{ctl} + e_{pos} + r_{sum}$, it is the sum of characteristic radius of the dynamic object and vehicle r_{sum}, the estimated responding error of the entire control system e_{ctl}, and the object position estimation error e_{pos}. e_{pos} can be estimated from posterior estimated covariance matrix $P_t^i \in \mathbb{R}^3$ of the KF, $e_{pos} = \sqrt{P_t^i[1,1] + P_t^i[2,2]t_f + P_t^i[3,3]t_f^2/2}$ ($t_f = \frac{l}{L}\delta_j + \sum_{q=1}^{j-1}\delta_q$). We use the ellipsoid to approximate the box to generate the convex and differentiable cost. $\lambda_{ds} \ll 1$ is the soft factor for the cost in the extended position distribution region for the first term in S_d, while the second term acts as the harder restriction. When there is ample space in the SFC to avoid the dynamic obstacles, the optimized trajectory will avoid the all the obstacle distribution $\mathscr{S}^i_{t_2+\Delta t}$, as the green curve shown in Fig. 6.6. If there are many dynamic obstacles and the position distribution may occupy too much space, the optimized trajectory is guaranteed only to avoid the predicted AABBs of the objects, and keeps the distance to the predicted dynamic objects' position as long as possible. In addition, S_e is essentially the integration of a polynomial, and it is easy to calculate the closed-form result in each optimization iteration. The other costs are accumulated in discrete form in our code by iterating the sample points along the trajectory.

To solve the optimization problem efficiently, we need the gradient w.r.t the joint points $\mathbf{p}$ between the trajectory pieces, written as $\partial G(\mathbf{p},\mathbf{T})/\partial\mathbf{p}$. Also, we use $\partial G(\mathbf{p},\mathbf{T})/\partial\mathbf{T}$ to represent the gradient w.r.t the time vector $\mathbf{T}$. The Gradient Propagation Law can derive the required gradients:

$$\begin{aligned}\frac{\partial G(\mathbf{p},\mathbf{T})}{\partial\mathbf{p}} &= \frac{\partial G}{\partial\mathbf{C}}\frac{\partial\mathbf{C}}{\partial\mathbf{p}}\\ \frac{\partial G(\mathbf{p},\mathbf{T})}{\partial\mathbf{T}} &= \frac{\partial G}{\partial\mathbf{T}} + \frac{\partial G}{\partial\mathbf{C}}\frac{\partial\mathbf{C}}{\partial\mathbf{T}},\end{aligned} \tag{6.33}$$

where $\mathbf{C}$ is the overall coefficient matrix representing $\left(\mathbf{C}_1^{\mathrm{T}}, \mathbf{C}_2^{\mathrm{T}}, \cdots, \mathbf{C}_{m-1}^{\mathrm{T}}\right)^{\mathrm{T}} \in \mathbb{R}^{2(m-1)s\times 3}$. Since we can calculate $\partial\mathbf{C}/\partial\mathbf{p}$ and $\partial\mathbf{C}/\partial\mathbf{T}$ efficiently referring [97], and the gradient

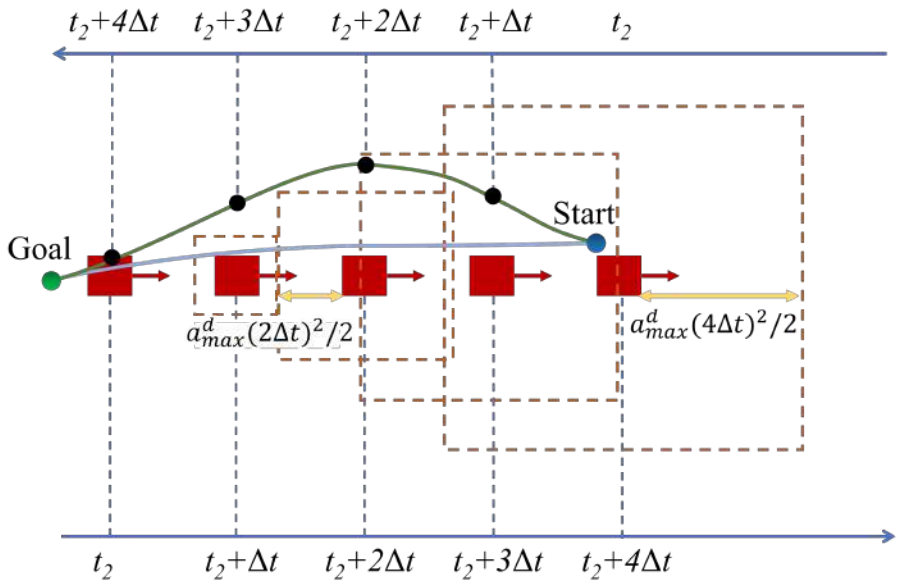


Fig. 6.6.: Illustration of the predicted dynamic objects' position distribution and the optimized trajectory. To make a clear expression, we put the 2D case. The dashed boxes are the position distribution region for the dynamic obstacle (red boxes with arrows) at some future timestamps. The blue curve stands for the trajectory and does not consider the collision cost brought by the unpredictable acceleration of moving objects, while the green curve does.

$\partial G/\partial \mathbf{C}$, $\partial G/\partial \mathbf{T}$ can be derived following the cost definition in (6.16)-(6.19) and (6.22), the gradients of the objective function are finally obtained. In addition, the constraints in (6.14) can all be eliminated by the method in [97], the optimization problem is transformed into an unconstrained one, and we can solve it efficiently with optimization algorithms such as L-BFGS.

6.3.3 Perception Enhanced Planning

In the related works, the desired yaw of the drone is often determined as the orientation angle of the desired velocity. However, the potential connection between dynamic perception and motion planning is not realized sufficiently. The camera fixed on the drone is likely to lose the tracked objects during their relative motion. We hope to track those obstacles with a high probability of collision longer to obtain a more precise velocity estimation from the KF. Fig. 6.7 demonstrates a situation when a dynamic obstacle moving towards the vehicle and

a “safe” trajectory has been planned. The safety of the trajectory is under the assumption that the estimated object velocity is precise, however, the initial velocity estimation often has an unignorable error unless the result can be optimized after several KF iterations. If the yaw follows the desired velocity[51] (left case in Fig. 6.7), the object's appearing time in the FOV is very short. The left case shows that the vehicle will collide with the dynamic obstacle due to the velocity estimation error. After the yaw is planned to actively track the object, as shown in the right figure, the object will be tracked for a longer time and the estimated velocity is more accurate. Thus, the object velocity estimation is enhanced, and the safety of the planned trajectory is improved.

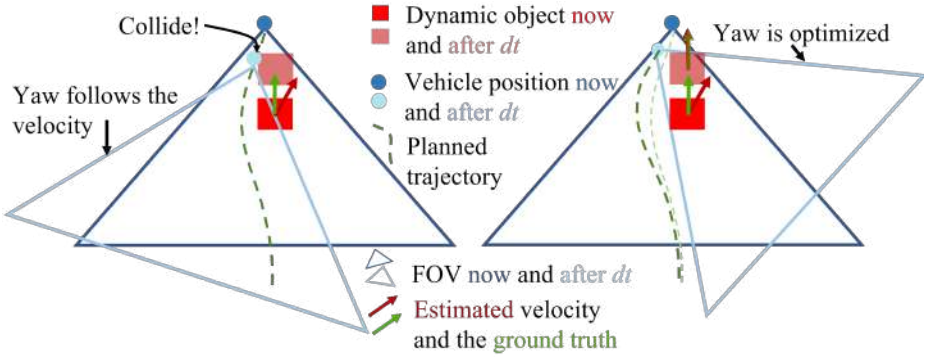


Fig. 6.7.: The importance of yaw planning. *dt* represents a short time period.

Let $\psi_{t_y} \in [\psi^v_{t_y} - \psi_{max}, \psi^v_{t_y} + \psi_{max}]$ denotes the yaw to be optimized at future time $t_y \in [t_{y0}, T_{m-1} + t_{y0}]$ (t_{y0} is the global time when the yaw planning starts), $\psi^v_{t_y}$ is the yaw of vehicle desired velocity at t_y, $p^{veh}_{t_y}$ is the vehicle position according to the optimized trajectory. $\hat{\mathbf{p}}_{t_y} = [\hat{p}^0_{t_y}, \cdots, \hat{p}^I_{t_y}]$ is the position of all the dynamic objects at time t_y.

First, we sample the trajectory evenly in the time horizon $[t_{y0}, T_{m-1} + t_{y0}]$ into N_t samples $\mathbf{T}_s = [t_{y0} + \frac{T_{m-1}}{N_t}, \cdots, t_{y0} + T_{m-1}]$, also sample yaw angle in range $[-\psi_{max}, \psi_{max}]$ evenly and obtain N_y samples $\Psi^{t_y}_s = [\psi^v_{t_y} - \psi_{max}, \psi^v_{t_y} - \psi_{max} + \frac{2\psi_{max}}{N_y - 1}, \cdots, \psi_{max} + \psi^v_{t_y}]$. For any $\psi^{n_y}_{t_y} \in \Psi^{t_y}_s$ ($0 < n_y \leq N_y$), we can obtain the corresponding desired thrust U_{t_y}, pitch $\theta^{n_y}_{t_y}$, and roll $\phi^{n_y}_{t_y}$ of the vehicle according to the desired acceleration $a_{t_y} = [a^x_{t_y}, a^y_{t_y}, a^z_{t_y}]$ and the known gravitational acceleration G:

$$\begin{aligned}
\theta_{t_y}^{n_y} &= arctan\left(\frac{a_{t_y}^x + a_{t_y}^y tan\psi_{t_y}^{n_y}}{(a_{t_y}^z + G)(cos\psi_{t_y}^{n_y} + sin\psi_{t_y}^{n_y} tan\psi_{t_y}^{n_y})}\right), \\
\phi_{t_y}^{n_y} &= arccos\left(\frac{a_{t_y}^z + G}{U_{t_y} cos\theta_{t_y}^{n_y}}\right), \\
U_{t_y} &= \|a_{t_y}\|_2.
\end{aligned} \tag{6.34}$$

Then, we have the FOV pyramid at t_y on earth coordinate since the vehicle position and orientation at t_y are all known. For each $\hat{p}_{t_y}^i \in \hat{\mathbf{p}}_{t_y}$ we calculate the visibility of all dynamic objects according to the method introduced in section 6.2, and we add 1 to the total score s_{t_y} of the current $\psi_{t_y}^{n_y}$ every once $\hat{p}_{t_y}^i$ is found in the FOV. Also, considering the obstacle close to the vehicle poses more threat, an additional score $1/\|\hat{p}_{t_y}^i - p_{t_y}^{veh}\|$ is added for each $\hat{p}_{t_y}^i$ in the FOV. The score related with the object visibility is denoted as $s^{vis}(t_y, n_y)$. We also hope the yaw angle gaps between each two adjacent time samples to be minimal. A large yaw gap may exceed the vehicle dynamic limit and disturb the height control. Thus, for $\psi_{t_y}^{n_y}$ we design the state transition score $s^{gap}(n_y, n_*, t_y) = \Delta\psi_{max} - |\psi_{t_y^-} - \psi_{t_y}|$ and $s(n_y, n_*, t_y) = s^{vis}(n_y, t_y) + s^{gap}(n_y, n_*, t_y)$, where $t_y^- = t_y - \frac{T_{m-1}}{N_t}$ is the former time sample, n_* stands for the sample index at time $t_y^-(0 < n_* \leq N_y)$, $\Delta\psi_{max}$ is the maximal yaw gap limit. If $s^{gap} < 0$ the transition between two yaw samples will not be considered. At last, we plan a series of yaw angles $\hat{\Psi} = [\hat{\psi}^1, \cdots, \hat{\psi}^{N_t}]$ which accumulate the highest score $\sum_{n=1}^{N_t} s$. The complete planning method is described in Algorithm 14, and we demonstrate an example for the planning in Fig. 6.8 in a more intuitive approach. In Algorithm 14, the node $\mathscr{N}$ is a structure and composes of several members such as yaw angle, score, and the pointer to its parent node.

We get the desired yaw at the specific time with linear interpolation on $\hat{\Psi}$, and forward the desired yaw together with the desired position, velocity, and acceleration to the flight controller.

Algorithm 14: Yaw angle planning

Predict the vehicle and dynamic objects' position with N_t time samples
Calculate s^{vis} for each time sample and each yaw angle sample in Ψ_s, initialize a node $\mathcal{N}$ for each sample, $\mathcal{N}_{t_y}^{n_y}.\psi \leftarrow \psi_{t_y}^{n_y}$, $\mathcal{N}_{t_y}^{n_y}.s^{vis} \leftarrow s^{vis}(n_y, t_y)$
for t_y in $\mathbf{T}_s$ **do**
    for $\psi_{t_y}^{n_y}$ in $\Psi_s^{t_y}$ **do**
        $\mathcal{N}_{t_y}^{n_y}.parent \leftarrow \mathcal{N}_{t_y^-}^{n_{max}}$, $\mathcal{N}_{t_y}^{n_y}.s \leftarrow \mathcal{N}_{t_y^-}^{n_{max}}.s + s_{gap}(n_y, n_{max}, t_y)$, n_{max} is the node index of the former time sample t_y^- that maximize $\mathcal{N}_{t_y}^{n_y}.s$.
Find the node with the maximal accumulated score s at the last time sample $T_{m-1} + t_{y0}$, denoted as $\mathcal{N}_{opt}$. Push $\mathcal{N}_{opt}.\psi$ in list $\hat{\Psi}$.
while $\mathcal{N}_{opt}.parent$ exists **do**
    Push $\mathcal{N}_{opt}.parent.\psi$ in empty list $\hat{\Psi}$.
    $\mathcal{N}_{opt} \leftarrow \mathcal{N}_{opt}.parent$
Return the planned yaw angles $\hat{\Psi}$

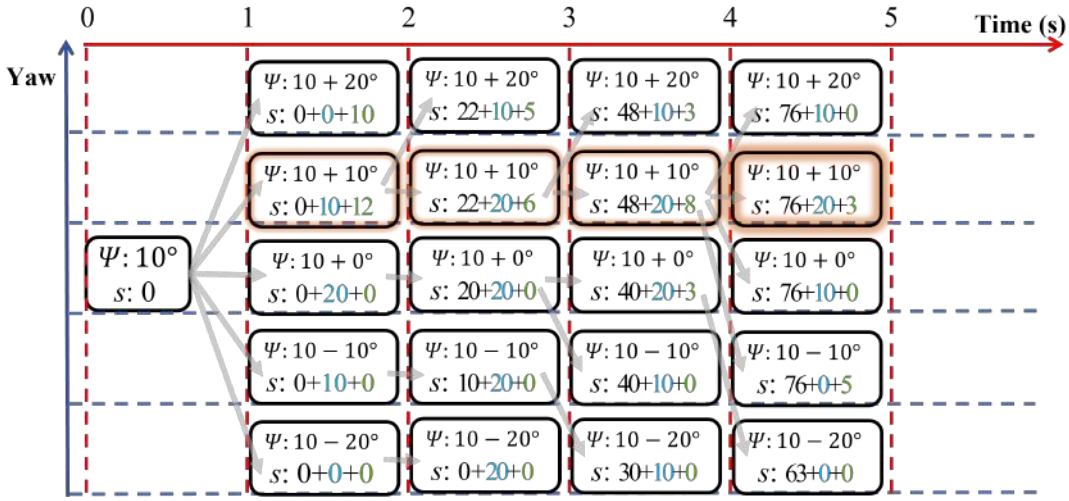


Fig. 6.8.: An example of the planning process. The nodes compose a graph, and at each time sample, they are connected to the former layer (time sample) to maximize the accumulated score in the dynamic programming way. The score inherited from the parent node is marked in black, while the state transition score is in blue and the object visibility score in green. The gray arrow starts from the parent node to its child node. The chosen yaw path is marked by orange glow.

6.4 Experimental Implementation and Results

6.4.1 Vehicle ego-motion compensation and depth map filter

To improve the object state estimation accuracy, the time gap between the latest depth map message from the camera and the vehicle state message from the IMU in the flight controller is addressed [93], which is an important detail. We assume the vehicle's motion in the short time gap is of constant acceleration, and the vehicle's pose is aligned to the timestamp of the depth map by the second-order interpolation

$$\hat{p}_{cam} = p_{cam} + v_{cam} t_{gap} + \frac{1}{2} a_{cam} t_{gap}^2. \tag{6.35}$$

$\hat{p}_{cam}$ is the compensated camera pose, p_{cam}, v_{cam} and $a_{cam} \in \mathbb{R}^{2\times 3}$ are the pose, velocity and acceleration of the camera obtained from raw data (translational and rotational motion). t_{gap} is the time gap that equals the depth map's timestamp minus the vehicle state's timestamp. The ego-motion compensation is the same as in chapter 5.

Also, the point cloud used for the object velocity estimation should be streamlined. The original depth map input is aligned to the RGB image with the camera's intrinsics, and they have the same resolution. However, the depth information is too dense, which is unnecessary and may overburden the computing device. The depth map is down-sampled after it is segmented into different dynamic objects, and the decimation factor $f = d_r/(2d_{min}^i tan(\frac{\alpha_{res}}{2}))$ is different for each object, determined by the minimum depth d_{min}^i of the object i and the angular resolution of α_{res} of the RGB camera. d_r is the pre-assigned average point distance gap of the converted point cloud. The down-sampling operation on the depth map is more efficient than the density filtering with the point cloud. The converted points' size is reduced, thus the time cost of the following operation, including the coordinate conversion and feature tensor building, is shortened. At last, we should consider that the depth image segment within the dynamic object's BB may includes pixels of the background or other objects. Binary large object (BLOB) detection is used to extract the pixels of the interested object inside the BB, which is a simple but robust object detection algorithm against noise. For

each depth map segment, we conduct BLOB detection and only keep the largest group of pixels, under the assumption that the detected object occupies the maximum area in the BB compared to other objects. For each depth map segment, we conduct BLOB detection and only keep the largest group of pixels, under the assumption that the detected object occupies the maximum area in the BB compared to other objects.

6.4.2 Static points memorizing and reusing

In our implementation, we adopt a classical method to store the static points in a list and reuse the memorized static points in the planning. Because the camera FOV is too narrow, the static obstacle will likely escape from the FOV due to the camera's motion during the flight even though the static obstacle is very close to the vehicle. By memorizing the points of the static obstacles that appeared previously, the planner can reuse the static points and avoid the static obstacles when they are not even captured in the current depth frame, otherwise, flight safety is much affected.

After the initial object detection results are published, we convert the depth map except for the regions of the dynamic objects into a pointcloud and push the static points into a list for initialization. The point coordinates are discretized regarding the pre-assigned voxel size (0.1 m). When new stationary points are obtained afterward, we add new discretized points in the list and remove the duplication, and the number of repetitions n_{rep} of each point in the list is also recorded. At last, only the points whose n_{rep} in the corresponding voxel is greater than the threshold N_{sta} are sorted as actual static points, and they are used in the hybrid A* path searching and SFC construction.

In addition, n_{rep} for each voxel will also decrease with time, once a $n_{rep} < N_{sta}$ is found after the decrease, the corresponding point will be removed from the pointcloud used in path planning. When a dynamic obstacle is detected, the part of pointcloud inside its 3D AABB is also deleted. Thus, the influence from dynamic objects in the depth map but not detected in RGB image on the map is much reduced. The repetition threshold can effectively avoid the disturbance of noise, which is a classical practice in most mapping toolkits. Our mapping

method is more flexible in deleting the wrongly classified static obstacles compared to the existing popular mapping toolkits such as Octomap[2] or MLMapping[3].

6.4.3 Experimental Configuration

The whole software system is tested and verified in the Robot Operation System (ROS)/Gazebo simulation environment first and then in the hardware experiment. The drone model used in the simulation is 3DR IRIS, and the underlying flight attitude controller is the PX4 1.12.3 firmware version. We use a cascade PID controller for the outer-loop control to follow the trajectory, with the desired position, velocity, and acceleration input. The depth camera model is an Intel Realsense D435i with a resolution of 640*480 (30 fps). For hardware experiments, we use a self-assembled quadrotor with a Q250 frame and a LattePanda Alpha 864s with an Intel m3-8100y processor, the camera resolution is 848*480, and other configuration keeps unchanged. A visual-inertial odometry toolkit[4] is adopted to obtain the pose of the drone. Table 6.1 shows the parameter settings for the simulation tests.

Table 6.1.: Parameters for the tests

Parameter	Value	Parameter	Value
N_{sta}	2	l_v	0.15 m
l_{iou}	0.3	d_t	0.25 s
l_{dp}	0.5 m	c_{fov}	10^5
T_{node}	0.8 s	n_{sp}	10
N_t	10	N_y	8
ψ_{max}	$\pi/2$	α_{res}	0.1°
d_r	0.1 m [2]	v_{max}	2.0 m/s
a_{max}	16 m/s^2		

[2] https://octomap.github.io/
[3] https://github.com/HKPolyU-UAV/MLMapping
[4] https://github.com/HKPolyU-UAV/FLVIS

6.4.4 Simulation Test

(1) Dynamic perception module test

First, the accuracy and stability of the dynamic object tracking and velocity estimation method are verified in the Gazebo simulation.

In the simulation world depicted in Fig. 6.9(a), there are four moving human models and some static objects such as a table, boxes, and pillars. The moving obstacles reciprocate on different straight trajectories. The camera is fixed on the drone's head, the z axis of the camera coincides with x axis of the drone frame. The cut-off depth is set to 8 m, which is approximate to the effective depth range of the real depth camera. We test the algorithm with a static camera and a moving camera respectively. For the static case, the drone hovers at the point $(-6, 0, 1.2)$ and heading along the positive X direction in the earth coordinate. We choose 50 random goals for the moving case and generate the flight trajectories with the minimum-snap method [98]. The trajectories are adjusted manually by inserting middle waypoints to guarantee collision-free with the static obstacles. We record the data of the vehicle state ground truth and the RGB and depth image for repeat tests. Fig. 6.9(b) depicts the visualized estimation results in Rviz.

The numeral results of the dynamic object perception tests are shown in Table 6.2. The dynamic perception performance is also compared with SOTA works in this table, where the metric Multiple Object Tracking Accuracy (MOTA) is adopted as defined in [88]. We also evaluate the average position estimation error err_p(m) and velocity error err_v(m/s) in the tests. The second line marked with * is for our method without using the image-based object detector and tracker. We convert the depth map into a point cloud, and cluster the points into individual objects. All the objects' velocities are estimated first and classified as static or dynamic with a speed threshold (similar in [76] and [93]). The speed threshold we used for these three methods is $0.3m/s$. We record the images from the camera (at 30 Hz) and UAV state data (at 100 Hz) for about 550 s, and repeat the test on the data 5 times to give the average results.

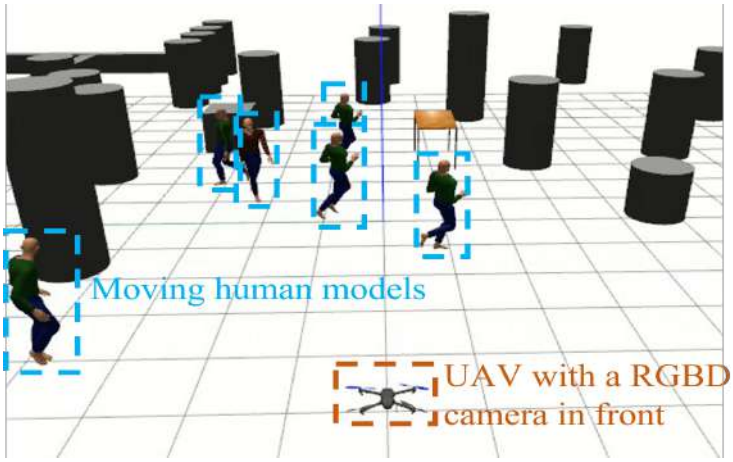
The estimation test results in the Gazebo simulation demonstrate that our estimation algorithm is robust in both static and dynamic cases, which is practical to be applied in AAC tasks. Compared with the existing SOTA works, our method efficiently improves the estimation accuracy and robustness in the clustered environment, especially when the camera is moving. Although the results are close in the static case, comparing our method with and without the front-end object detector, we can see the exciting improvement in MOTA in the dynamic case. The dynamic object detecting accuracy is rarely affected when the camera is moving, and the inaccurate velocity estimation of the actual static obstacles is avoided. For our MOTA, it is composed of a false negatives rate 7.5% (covering non-detected dynamic objects), a false positives rate 0.3% (static objects misclassified as dynamic), and a mismatch rate 1.1%. If the image-based object detector is banned, the false positives rate increases to 7.1%, which make a major difference.

Table 6.2.: Obstacle State Estimation Comparison

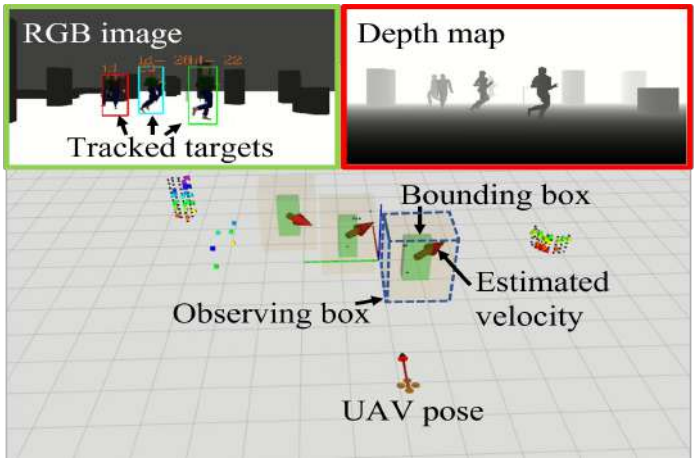
Method	Static case			Dynamic case		
	err_v	err_p	MOTA	err_v	err_p	MOTA
Ours	0.22	**0.12**	**91.1**	**0.35**	**0.14**	**89.8**
Ours*	**0.21**	0.12	87.4	0.45	0.26	77.3
[93]	0.28	0.14	84.1	0.49	0.27	74.5
[76]	0.38	0.26	77.1	0.51	0.30	70.6

(2) System test

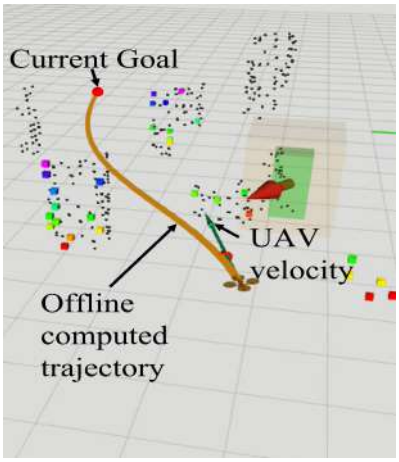
After the dynamic object perception part is validated and compared independently, we test the complete navigation system in simulation. We adopt the same 50 randomly chosen goals mentioned above and program the drone to reach the goals successively. We also replace the planning part with the method proposed in [76] for flight tests to compare the trajectory planning performance. The maximal flight speed is set to 2.5 m/s, the maximal acceleration is set to 10 m/s^2, and the flight height is limited to under 2 m to increase difficulty. For the compared methods, we utilize the total flight time t_{fly}(s), average re-



(a)



(b)



(c)

Fig. 6.9.: (a): The simulation environment for the moving object tracking and velocity estimation test. (b): The visualized estimation results in RVIZ for the static case, corresponding to (a). (c): The visualized results of the dynamic case, the drone is controlled to follow an offline trajectory to traveling in the world. The colorful boxes in the Rviz window represent the stored static points, the black dots are the non-dynamic point cloud of the current depth map.

planning time cost t_{plan}(ms), actual maximal speed $v_{act}(m/s)$, number of collisions as indicators n_{fail}, and the results is shown in Table 6.3. # marks our trajectory planning method without considering the position distribution caused by the acceleration uncertainty, and - represents our method without the yaw planning.

We can conclude from the results that our method is the safest, and our proposed additional cost of object's acceleration uncertainty in the trajectory optimization and the yaw

Table 6.3.: System Performance Comparison

Method	t_{fly}	v_{act}	n_{fail}	t_{plan}
Ours	394.4	2.503	**0**	7.96
Ours#	380.1	2.506	5	7.13
Ours˜	391.2	2.501	2	7.20
[76]	441.9	2.231	7	8.87

planning method can effectively improve flight safety. The re-planning time cost also shows improvement compared to [76]. Because our method is based on SFC and the effective MINCO class, the computation load of calculating the collision cost is lighter than the control point method [76]. In addition, the total flight time of our method shows superiority, which benefits from spatial-temporal optimization. The trajectory duration is also shortened after optimization, while [76] just relaxes the time allocation manually if the maximum trajectory speed exceeds the limitation after the trajectory optimization, and it is also the reason our actual maximal flight speed is higher and closer to the upper boundary.

In Fig. 6.10(a), we intuitively demonstrate the difference between the trajectories with and without the cost of dynamic objects' acceleration uncertainty in the trajectory optimization. The trajectory is more conservative after the acceleration uncertainty is considered when there is adequate free space in the environment. When there are more dynamic obstacles and the free space is insufficient, the resulting trajectory is "pushed" to the middle position between the dynamic obstacles at each future time sample by the gradient of the collision cost. The visualized results of the perception-enhanced yaw planning are displayed in Fig. 6.10(b), and they are compared intuitively with simply keeping the vehicle yaw coincide with the velocity direction (original practice). We can see the camera FOV fails to cover the dynamic obstacles sometimes during the trajectory in the original practice, and our proposed method can track the dynamic obstacles well along the trajectory under the maximal yaw range ψ_{max}. The object tracking robustness and object velocity estimating

precision can both be improved. The related videos for autonomous flight tests in Gazebo simulation are uploaded online[5].

6.4.5 Field test

The field test is based on the same hardware platform introduced in Chapter 5, and here we omit the introduction. In this chapter, the autonomous flight is based on the onboard VIO toolkit rather than an outer localization method, highlighting the practical application value for real-world tasks.

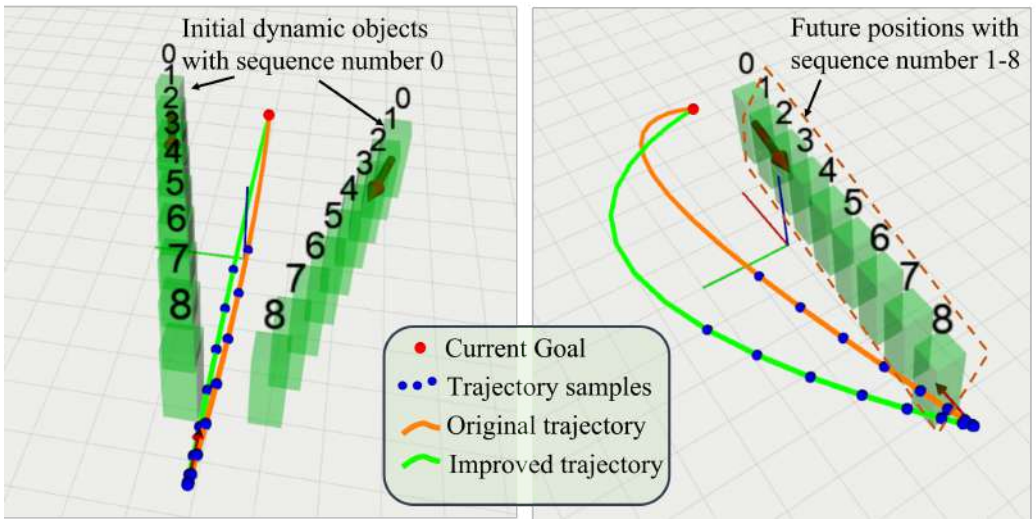
(1) Preparation and implementation details

To deploy all our algorithms with ROS on a tiny onboard computer, an efficient way is using nodelet manager[6]. The most important benefit of nodelet is that the efficiency of ROS message transmission between different ROS nodes is much improved. The traditional way for ROS nodes to transport messages requires frequent compression and decompression. A considerable CPU resource is wasted for messages of large sizes, such as images or pointclouds. By utilizing nodelet method, the nodelets under the same manager can share the ROS messages (data) directly from computer RAM without any additional procession, which can release the tight computing resources. To study the improvement to the coding efficiency after we use the nodelet method, we record the CPU occupancy rate before and after the code is modified. The overall CPU occupancy rate is given by the *htop* tool in Ubuntu 18.04 system, and we conduct the test under different load levels by incrementally launching each submodule. The results are given in Table 6.4, where the VIO receives the stereo images at 15 Hz. The RGB image, depth images, and point cloud are published at 30 Hz.

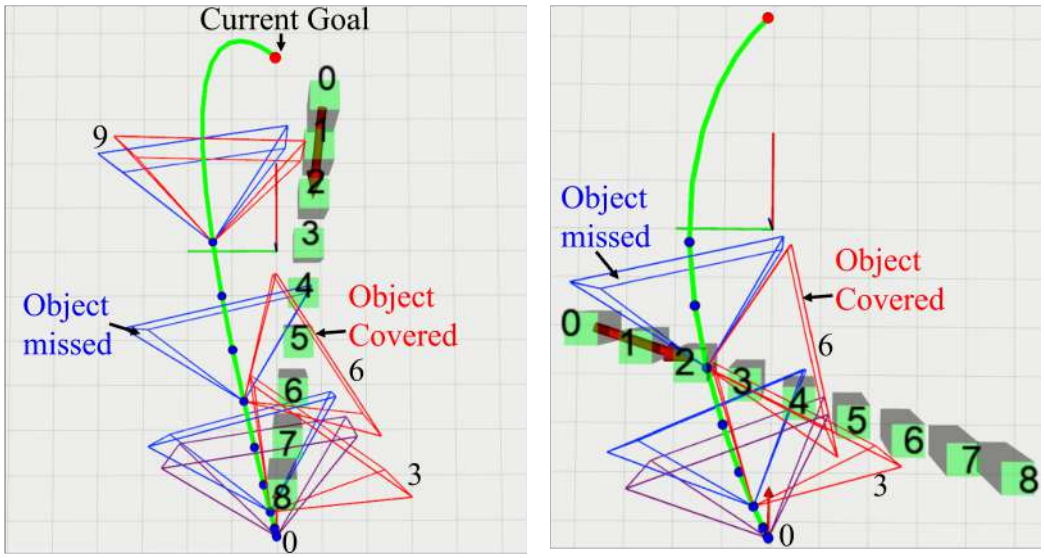
In Table 6.4, '+' in the head represents the submodule is launched while the submodules on the left side are all running. It is pleasing to see that using the nodelet can save 10-15%

[5] `https://youtu.be/pB_6jBgP1TQ`, or, `https://www.bilibili.com/video/BV1eS4y1W7Yx/?vd_source=f6ef8ce98ca35388105d2a5fc1b94035`

[6] `http://wiki.ros.org/nodelet`



(a)



(b)

Fig. 6.10.: (a): Difference between the trajectories with and without the cost of dynamic objects' acceleration uncertainty in the trajectory optimization. The left figure shows a case of multiple dynamic objects, and the right figure shows the single-object case. (b): Visualized results of the perception-enhanced yaw planning. The original FOV pyramids of the camera are represented in blue, and the optimized FOVs are in red. Our method does not change the yaw at the initial state, the FOVs are completely coincident at frame 0 and are shown in purple. The left figure demonstrates an object moving from back to front, and the object in the right figure is moving from left to right. The black numbers are the sequence numbers, and the time gap between samples is 0.5 s.

Table 6.4.: CPU load Comparison

Submodule	Camera&VIO + Detector&tracker		+Filters&Velocimetry	+ Controller & EKF & Planner
CPU load (Node)	42-46%	50-56%	60-68%	72-78%
CPU load (Nodelet)	33-36%	39-44%	49-56%	61-65%

CPU load on the onboard computer, and for the submodules that require image input (VIO and the image-based object detector and tracker) or raw pointcloud (pointcloud filters) the computational load improvement is the most apparent, which fits well with the ”data zero-copy” feature of nodelet. By leaving sufficient free computing resources, we can ensure the loop frequency of each submodule can reach our designed settings even in some complicated scenes, such as dense environments with multiple dynamic objects.

Another critical gap between simulation and hardware is the VIO toolkit[7] has latency and dynamic estimation error towards the vehicle movement, thus it brings trouble to the outer-loop controller. We tune the PID parameter before the autonomous flight with the planner. The outer-loop controller is sensitive to the dynamic delay of VIO output, and the vehicle can only hover stably at small control gains, which leads to bad performance in trajectory tracking. Moreover, when all the submodules are working, the VIO may fail to process the image in time, and sometimes the arrived image has to be discarded. Thus, the state estimation delay from VIO is heavier, the dynamic accuracy is reduced, and the vehicle cannot hover stably even with smaller control gains. To address this problem, we use an extended Kalman filter (EKF) to improve the original odometry output of VIO. As a result, the vehicle can hover at a more aggresive control gain, and the trajectory tracking error is much improved. The control gain of the outer controller and the average trajectory tracking error are measured and listed in Table 6.5.

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

Table 6.5.: Controller tracking error

Configuration	Maximal control gain (kp&kv)	Position error (m)	velocity error (m/s)
VIO + Controller	1.3/1.4	0.34	0.48
Full load	N/A	0.57	0.75
Full load + EKF	2.75/2.6	**0.14**	**0.24**

At last, the latency from the depth camera to generate the depth map should be considered. With the help of the official latency tool, as shown in Fig. 6.11, the latency of the infra image can be measured. Because the timestamp of the depth map and point cloud from the camera is set exactly identical to the infra images, the latency of the depth map can also be determined. The latency t_{lat} is about 103 ms for the $848 * 480$ resolution at 30 Hz, which is a surprisingly huge latency and must be compensated in our algorithm. When we convert the point cloud from the body frame into the earth frame, we need the time-aligned message of the vehicle pose and depth map. So we first measure the latency of the depth image relative to the IMU message, denoted as t_{d2i}. We set up an simple test to measure t_{d2i}. The drone is put on a table while the camera and IMU are working and the data is recording. Then, we hit the vehicle frame and made a sudden displacement. We can check the IMU and image raw data and record the timestamp when the hit happens. The sudden huge change in Gyro and the frame different from the previous one of recorded images are considered to correspond to the same event (hit), and t_{d2i} is the timestamp difference. The hit test is repeated 10 times, and the average t_{d2i} is about 80 ms. The timestamp of the depth map is reduced by t_{d2i} for the time alignment. When the states of the dynamic objects are published, the timestamp is reduced by $t_{lat} - t_{d2i}$ to give the real timestamp for the observed object position and velocity.

[8]Image source: `https://dev.intelrealsense.com/docs/rs-latency-tool`

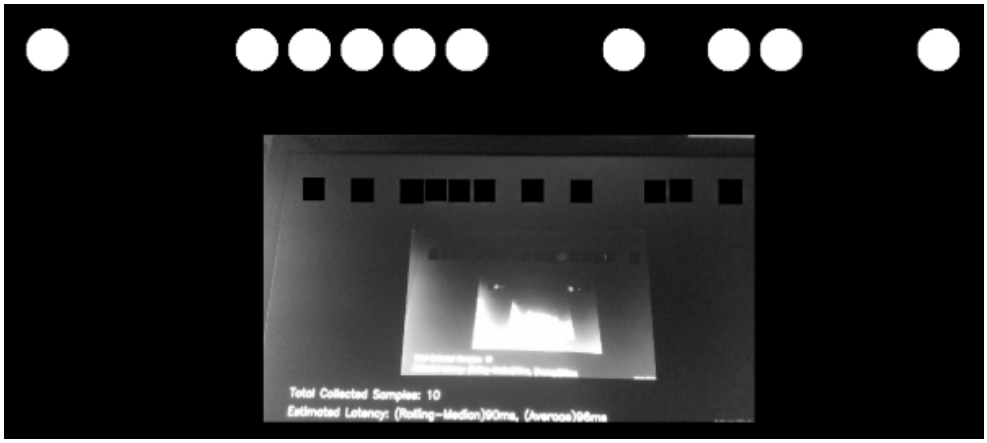


Fig. 6.11.: The latency test window[8]. The digital clock is encoded into binary form, and the program render the bits to screen. It then use Hough Transform to identify sent bits in the latest frame from the camera (marked as black squares), and calculate the latency.

(2) Autonomous flight

First, we arranged an indoor venue with only static obstacles. Due to the limited size of the valid space for UAV flight in the lab, the flight distance between the initial position and the goal is not long, but we make the vehicle travel between the two positions several times to show the robust flight performance. We put several boxes and roll-up banners on the ground as obstacles and let the drone travel between the two ends of the field repeatedly to validate the flight safety in a static environment. When the drone reaches the goal point, it will hover and turns its head to the next goal, and then recall the planner for navigation. The flight environment is shown in Fig. 6.12, and the related video has been uploaded online[9]. In the video, we can see the drone fly agilely among the obstacles, and the body attitude during the flight is changing smoothly, indicating the planner's efficiency in flight time and energy utilization. In this test, the maximum velocity is constrained as $1.8m/s$, and the maximum propulsion acceleration (total thrust divided by vehicle mass) is set to $16m/s^2$.

Then, to test the dynamic obstacle avoidance performance in the real world, we make the drone travel between two points with one pedestrian repeatedly walking straight toward the drone. Because the VIO tool is not robust to aggressive maneuvers and interference

[9] https://www.youtube.com/watch?v=xEJT-yrf6LI, https://www.bilibili.com/video/BV1GB4y1z7Hj/?vd_source=f6ef8ce98ca35388105d2a5fc1b94035

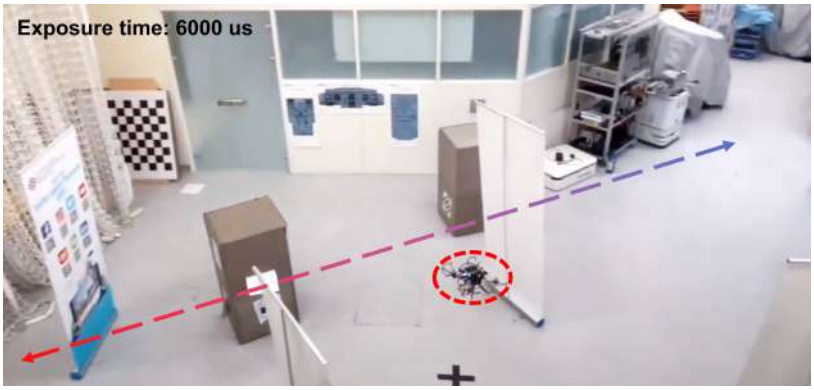
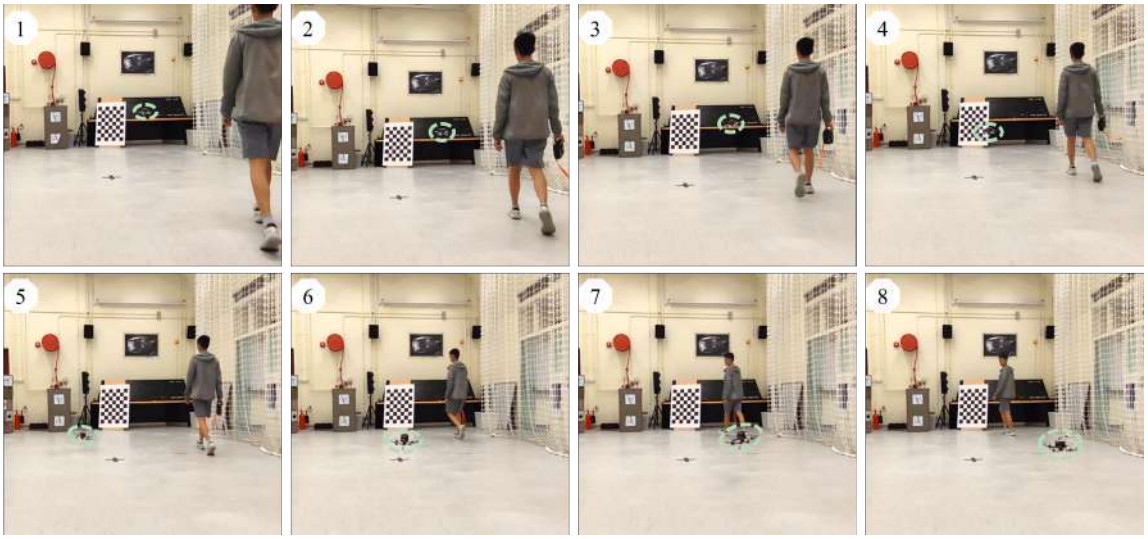


Fig. 6.12.: The obstacle layout for indoor flight test in static environment

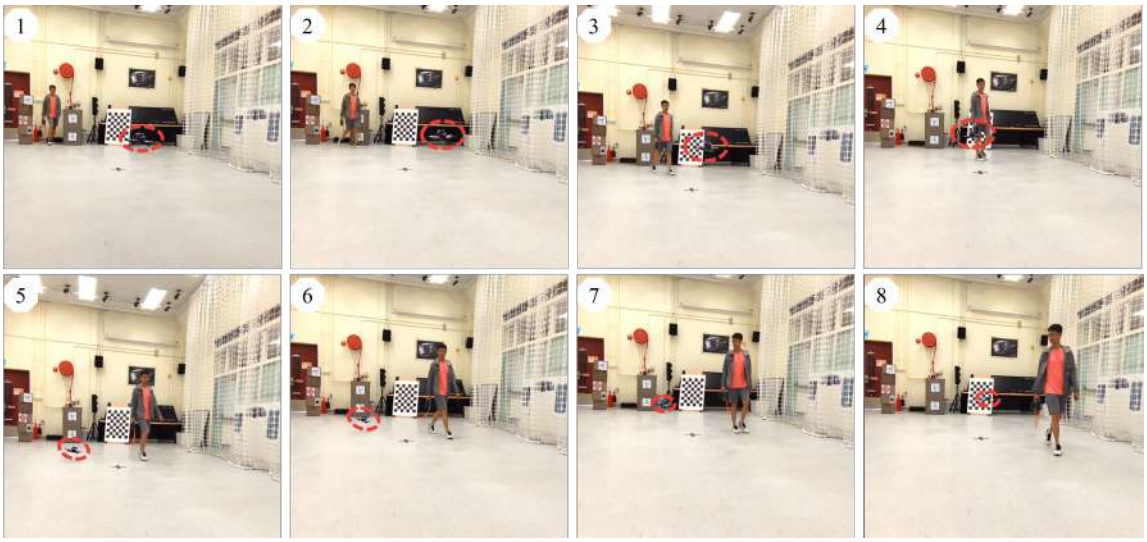
from moving objects, we set the maximum cruising speed to $1.5m/s$, and the speed limit is increased to $2.2m/s$ for avoiding dynamic objects. Because we conduct this test with a real person, safety should be highly valued, and an aggressive test is not capable at the current stage. The propulsion acceleration limit is $16m/s^2$, and the maximum horizontal acceleration is about $12.5m/s^2$. In Fig. 6.13 we put the snapshots of two flight trials. In the first flight, the drone set out from the position far from the recording camera and flew to the goal close to the camera, and it flew back in the second trial. We can see the drone successfully avoid the person, keeping a satisfactory safe distance away, and does not affect the person's original movement. The related video has been uploaded online[10].

At last, we turn to the outdoor environment and test our system in a larger area. We test our autonomous UAV system in two scenarios, as shown in Fig. 6.14, one is in a grove (the same spot introduced in Chapter 4), and another one is located on a hard cement pavement with many roadblocks. Although the lighting conditions vary in the environment while the vehicle is required to fly through both shadow and strong sunlight (the difference in light intensity is about 70 times), and the obstacles in the grove are of complex shape, our system can hold the challenges and achieve safe flight for navigation tasks.

[10] https://www.youtube.com/watch?v=xEJT-yrf6LI, https://www.bilibili.com/video/BV1GB4y1z7Hj/?vd_source=f6ef8ce98ca35388105d2a5fc1b94035



(a)

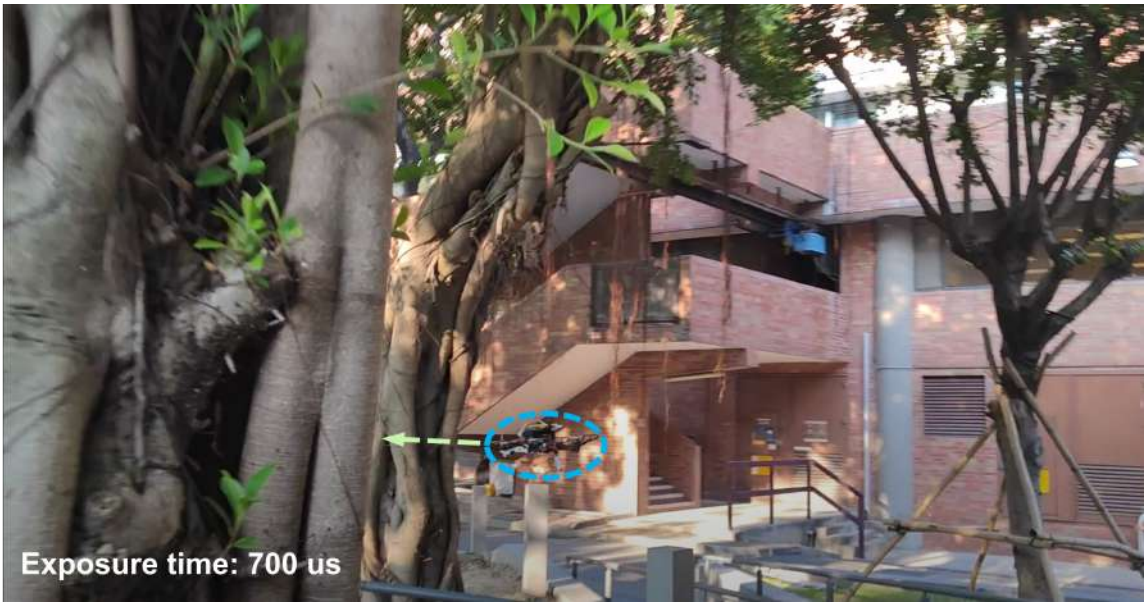


(b)

Fig. 6.13.: Two trials of dynamic object avoidance indoor flight test. The number on the left upper corner of each subfigure mark the frame sequence as time increasing.

6.5 Conclusion

In this chapter, we propose an enhanced autonomous flight system, and the progress is made in several aspects compared to Chapter 5, as well as the SOTA works. The simulation tests show the better robustness of our proposed dynamic object tracking system, and comprehensive flight tests in simulation and real-world demonstrate the feasibility and



(a)

(b)

Fig. 6.14.: (a): The autonomous flight test in the grove. (b): The autonomous flight test among the roadblocks. The arrows point to the direction to destination.

practicality of our proposed system, indicating it can handle the navigation task in complex unknown dynamic environment.

In summary, the contributions of this chapter are from the following two aspects:

For the obstacle perception, we first propose a lightweight image-based object detecting and tracking framework to give continuous object classification and mark the objects that are likely to be dynamic in the environment. We then introduce a novel object velocity estimating technique developed from the original Particle Image Velocimetry (PIV) method. It is extended to the 3D case to estimate the object velocity and can work with the unordered point cloud input by extracting local features of the point cloud. We integrate the techniques together to classify the objects in the environment as dynamic or static and estimate the velocity of the dynamic ones.

For the trajectory planning of the drone, we modify the hybrid A star algorithm to limit the resulted front-end path in the narrow field of view (FOV) of the RGBD camera to improve path safety. In addition, based on the minimum control (MINCO) class for trajectory optimization, we proposed the time-varying safety margin in the objective function for the dynamic obstacles to enhance safety by considering the different maximal accelerations of different object types. At last, we also proposed an efficient yaw planning method to optimize the heading of the camera, making more valuable targets appear in the FOV and better perceive the dynamic objects.

Last but not least, we completely release the codes and hardware plans of our aerial platform for the reference of the community[11].

[11] https://github.com/chenhanpolyu/AutoFly-demo

7. CONCLUSION, DISCUSSION, AND FUTURE WORK

In this thesis, we introduced several novel algorithms on path planning, motion planning, and environment perception, to address the existed problems in UAVs' autonomous flight in unknown and dynamic environments. The proposed algorithms are proved to overperform the state-of-the-art by exhaustive simulation and hardware tests. In addition, we propose complete autonomous flight systems with the introduced novel algorithms and demonstrate real flights on a self-assembled low-cost UAV platform to highlight the practicability of our systems. The tests indicate the promising potential application value in many scenarios, such as UAV logistics, search and rescue, and autonomous aerial photography.

The research starts from the motivation to improve the computing efficiency of the existing autonomous navigation system of UAVs, making all the algorithms capable of being deployed on micro aerial platforms that can only carry tiny onboard computers with a very limited computing resource. Although the motion planning algorithms on the onboard computer turn out fast enough for real-time computing, the narrow FOV of the depth camera becomes the bottleneck of flight safety and efficiency in a complex environment. Later, we developed a path planner working on the map, which is much larger than the camera FOV, to guide the local motion planner and avoid potential detours, confronting a more complex and dense environment. We also noticed that avoiding those continuously moving obstacles requires prediction of their future positions, or the planned trajectory cannot be guaranteed safe in the future. Thus, we propose a velocity planning algorithm based on the relative velocities toward obstacles in the environment. We also design a novel lightweight pointcloud-based obstacle tracking and velocity estimation algorithm method to construct a complete system. At last, we improve the robustness of the dynamic object perception part by introducing the image-based object detector and tracker, and enhance the flight smoothness and speed by the polynomial trajectory optimization approach. In addition, we exploit the active yaw angle control of the vehicle to improve the dynamic object tracking accuracy,

and involve the object state estimation covariance and state transmission covariance in the trajectory optimization to make the planned trajectory adapt to the uncertainty in the obstacle perception.

From the view of the whole system, the most severe factor that prevents the autonomous UAV from widespread real-world applications is the vision-based localization toolkit. While those lidar SLAM algorithms show robust and accurate localization in most scenarios and on high-speed vehicles, the weight and high power consumption make lidars unsuitable for micro aerial platforms. The VIO algorithms suffer from motion blur when the camera is moving or rotating, and it is also not good at working in a dark or feature-lacking environment. Another bottleneck comes from the narrow FOV of the depth camera, if it can only sense the obstacles very close, the vehicle cannot move at high speed, and it is also difficult to avoid the moving obstacles at high speed.

In the future, we plan to investigate the deep-learning-based VIO and depth map generation method to strengthen the flight system's robustness and ability of aggressive motion. Before applying our system in applications such as searching for a target in an unknown environment, or autonomous aerial photography, some problems still remain to be solved in the future. For exploration purposes, the critical issue is to plan an efficient trip to cover all the space in the assigned search range using the onboard sensor, while avoiding obstacles. However, we can only determine the optimal global trip after exploring the entire environment. During the exploration, the intelligent prediction of the unexplored area based on the explored environmental information should be helpful for improving the searching efficiency. In addition, planning a collision-free and optimal trip requires frequent path planning on the latest map to estimate the trip cost. Improving the computing efficiency of solving multi-query path planning problems with a growing map should also be considered.

For autonomous aerial photography, the photography requirement may conflict with obstacle avoidance, including the target visibility and the particular relative position of the flying camera to the target. We should find a clever motion strategy to balance the multiple targets, fulfilling the higher-level tasks while maintaining flight safety. In addition, the higher level of motion strategy planning still remains a massive room for progress. Making

UAVs design a series of keyframes independently according to the characteristics of the scene and the movement of the target so as to complete an aerial video close to the level of professional human photographers is also a precious research direction.

Bibliography

[1] G.-Z. Yang, J. Bellingham, P. E. Dupont, P. Fischer, L. Floridi, R. Full, N. Jacobstein, V. Kumar, M. McNutt, R. Merrifield et al., "The grand challenges of science robotics," Science robotics, vol. 3, no. 14, p. eaar7650, 2018.

[2] D. Dey, K. S. Shankar, S. Zeng, R. Mehta, M. T. Agcayazi, C. Eriksen, S. Daftry, M. Hebert, and J. A. Bagnell, Vision and Learning for Deliberative Monocular Cluttered Flight. Cham: Springer International Publishing, 2016, pp. 391–409.

[3] P. R. Florence, J. Carter, J. Ware, and R. Tedrake, "NanoMap: Fast, Uncertainty-Aware Proximity Queries with Lazy Search Over Local 3D Data," in 2018 IEEE International Conference on Robotics and Automation (ICRA), 2018, pp. 7631–7638.

[4] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," The International Journal of Robotics Research, vol. 33, no. 9, pp. 1251–1270, 2014.

[5] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "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, pp. 1917–1922.

[6] F. Duchoň, A. Babinec, M. Kajan, P. Beňo, M. Florek, T. Fico, and L. Jurišica, "Path planning with modified a star algorithm for a mobile robot," Procedia Engineering, vol. 96, pp. 59–69, 2014.

[7] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," Computer, vol. 22, no. 6, pp. 46–57, 1989.

[8] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: an efficient probabilistic 3D mapping framework based on octrees," Autonomous Robots, vol. 34, no. 3, pp. 189–206, 2013.

[9] S. Choi, J. Park, E. Lim, and W. Yu, "Global path planning on uneven elevation maps," in 2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), 2012, pp. 49–54.

[10] S. A. M. Coenen, J. J. M. Lunenburg, M. J. G. van de Molengraft, and M. Steinbuch, "A representation method based on the probability of collision for safe robot navigation in domestic environments," 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 4177–4183, 2014.

[11] B. T. Lopez and J. P. How, "Aggressive collision avoidance with limited field-of-view sensing," in 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 1358–1365.

[12] C. Yu, Z. Liu, X.-J. Liu, F. Xie, Y. Yang, Q. Wei, and Q. Fei, "DS-SLAM: A Semantic Visual SLAM towards Dynamic Environments," in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 1168–1174.

[13] J. Kim and Y. Do, "Moving obstacle avoidance of a mobile robot using a single camera," Procedia Engineering, vol. 41, pp. 911–916, 2012.

[14] H. Oleynikova, D. Honegger, and M. Pollefeys, "Reactive avoidance using embedded stereo vision for MAV flight," in 2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2015, pp. 50–56.

[15] P. Skulimowski, M. Owczarek, A. Radecki, M. Bujacz, D. Rzeszotarski, and P. Strumillo, "Interactive sonification of U-depth images in a navigation aid for the visually impaired," Journal on Multimodal User Interfaces, vol. 13, no. 3, pp. 219–230, 2019.

[16] T. Nageli, J. Alonso-Mora, A. Domahidi, D. Rus, O. Hilliges, N. Tobias, J. Alonso-Mora, A. Domahidi, D. Rus, and O. Hilliges, "Real-Time Motion Planning for Aerial Videography With Real-Time With Dynamic Obstacle Avoidance and Viewpoint Optimization," in IEEE Robotics and Automation Letters, vol. 2, no. 3, 2017, pp. 1696–1703.

[17] A. Ess, B. Leibe, K. Schindler, and L. van Gool, "Moving obstacle detection in highly dynamic scenes," in 2009 IEEE International Conference on Robotics and Automation, 2009, pp. 4451–4458.

[18] K. Berker Logoglu, H. Lezki, M. Kerim Yucel, A. Ozturk, A. Kucukkomurler, B. Karagoz, E. Erdem, and A. Erdem, "Feature-based efficient moving object detection for low-altitude aerial platforms," in Proceedings of the IEEE International Conference on Computer Vision Workshops, 2017, pp. 2119–2128.

[19] I. A. Bârsan, P. Liu, M. Pollefeys, and A. Geiger, "Robust dense mapping for large-scale dynamic environments," in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 7510–7517.

[20] S. Zhang, R. Benenson, M. Omran, J. Hosang, and B. Schiele, "Towards reaching human performance in pedestrian detection," IEEE transactions on pattern analysis and machine intelligence, vol. 40, no. 4, pp. 973–986, 2017.

[21] T. Eppenberger, G. Cesari, M. Dymczyk, R. Siegwart, and R. Dubé, "Leveraging stereo-camera data for real-time dynamic obstacle detection and tracking," in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2020, pp. 10 528–10 535.

[22] S. Kraemer, C. Stiller, and M. E. Bouzouraa, "LiDAR-based object tracking and shape estimation using polylines and free-space information," in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018, pp. 4515–4522.

[23] J. Miller, A. Hasfura, S.-Y. Liu, and J. P. How, "Dynamic arrival rate estimation for campus mobility on demand network graphs," in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2016, pp. 2285–2292.

[24] A. Cherubini, F. Spindler, and F. Chaumette, "Autonomous Visual Navigation and Laser-Based Moving Obstacle Avoidance," IEEE Transactions on Intelligent Transportation Systems, vol. 15, no. 5, pp. 2101–2110, 2014.

[25] D. Falanga, K. Kleber, and D. Scaramuzza, "Dynamic obstacle avoidance for quadrotors with event cameras," Science Robotics, vol. 5, no. 40, p. eaaz9712, 2020.

[26] B. He, H. Li, S. Wu, D. Wang, Z. Zhang, Q. Dong, C. Xu, and F. Gao, "Fast-dynamic-vision: Detection and tracking dynamic objects with event and depth sensing," in 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2021, pp. 3071–3078.

[27] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," The International Journal of Robotics Research, vol. 5, no. 1, pp. 90–98, 1986.

[28] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," international conference on robotics and automation, vol. 7, no. 3, pp. 278–288, 1991.

[29] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," IEEE Robotics Automation Magazine, vol. 4, no. 1, pp. 23–33, 1997.

[30] E. W. Dijkstra, "A note on two problems in connexion with graphs," Numerische Mathematik, vol. 1, pp. 269–271, 1959.

[31] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, pp. 100–107, 1968.

[32] M. Likhachev, G. J. Gordon, and S. Thrun, "ARA*: Anytime A* with provable bounds on sub-optimality," Advances in neural information processing systems, vol. 16, 2003.

[33] D. Harabor and A. Grastien, "Online Graph Pruning for Pathfinding On Grid Maps," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 25, no. 1, pp. 1114–1119, 2011.

[34] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments," The International Journal of Robotics Research, vol. 29, no. 5, pp. 485–501, 2010.

[35] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," The International Journal of Robotics Research, vol. 20, no. 5, pp. 378–400, 2001.

[36] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," IEEE Transactions on Robotics and Automation, vol. 12, no. 4, pp. 566–580, 1996.

[37] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," The International Journal of Robotics Research, vol. 30, no. 7, pp. 846–894, 2011.

[38] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in 2013 IEEE International Conference on Robotics and Automation, 2013, pp. 5054–5061.

[39] A. Bry and N. Roy, "Rapidly-exploring Random Belief Trees for motion planning under uncertainty," in 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 723–730.

[40] S. Liu, M. Watterson, S. Tang, and V. Kumar, "High speed navigation for quadrotors with limited onboard sensing," in 2016 IEEE international conference on robotics and automation (ICRA). IEEE, 2016, pp. 1484–1491.

[41] H. Chen, P. Lu, and C. Xiao, "Dynamic Obstacle Avoidance for UAVs Using a Fast Trajectory Planning Approach," in 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO), 2019, pp. 1459–1464.

[42] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 2520–2525.

[43] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in Algorithmic foundations of robotics XI. Springer, 2015, pp. 109–124.

[44] ——, "Efficient mixed-integer planning for uavs in cluttered environments," in 2015 IEEE international conference on robotics and automation (ICRA). IEEE, 2015, pp. 42–49.

[45] B. Landry, R. Deits, P. R. Florence, and R. Tedrake, "Aggressive quadrotor flight through cluttered environments using mixed integer programming," in 2016 IEEE international conference on robotics and automation (ICRA). IEEE, 2016, pp. 1469–1475.

[46] J. A. Preiss, K. Hausman, G. S. Sukhatme, and S. Weiss, "Trajectory optimization for self-calibration and navigation." in Robotics: Science and Systems, vol. 13, 2017.

[47] F. Gao, W. Wu, Y. Lin, and S. Shen, "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, pp. 344–351.

[48] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," IEEE Robotics and Automation Letters, vol. 4, no. 4, pp. 3529–3536, 2019.

[49] M. Watterson and V. Kumar, "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, pp. 3235–3240.

[50] J. van den Berg, D. Wilkie, S. J. Guy, M. Niethammer, and D. Manocha, "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, pp. 346–353.

[51] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto, "Safe Local Exploration for Replanning in Cluttered Unknown Environments for Microaerial Vehicles," in IEEE Robotics and Automation Letters, vol. 3, no. 3, 2018, pp. 1474–1481.

[52] M. W. Mueller, M. Hehn, and R. D. Andrea, "for Quadrocopter Trajectory Generation," IEEE Transactions on Robotics, vol. 31, no. 6, pp. 1294–1310, 2015.

[53] T. M. Howard, C. J. Green, A. Kelly, and D. Ferguson, "State space sampling of feasible motions for high-performance mobile robot navigation in complex environments," Journal of Field Robotics, vol. 25, no. 6-7, pp. 325–345, 2008.

[54] M. Weinmann, B. Jutzi, S. Hinz, and C. Mallet, "ISPRS Journal of Photogrammetry and Remote Sensing Semantic point cloud interpretation based on optimal neighborhoods , relevant features and efficient classifiers," ISPRS Journal of Photogrammetry and Remote Sensing, vol. 105, pp. 286–304, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.isprsjprs.2015.01.016

[55] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2014, pp. 2997–3004.

[56] ——, "Batch informed trees (bit): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in 2015 IEEE international conference on robotics and automation (ICRA). IEEE, 2015, pp. 3067–3074.

[57] M. P. Strub and J. D. Gammell, "Adaptively informed trees (ait*): Fast asymptotically optimal path planning through adaptive heuristics," in 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020, pp. 3191–3198.

[58] ——, "Ait* & eit*: Asymmetric bidirectional sampling-based path planning," The International Journal of Robotics Research (IJRR), 2021.

[59] J. Chen, T. Liu, and S. Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," in 2016 IEEE International Conference on Robotics and Automation (ICRA), vol. 2016-June. IEEE, 2016, pp. 1476–1483.

[60] M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-UAV motion replanning for exploring unknown environments," in 2013 IEEE International Conference on Robotics and Automation, 2013, pp. 2452–2458.

[61] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online UAV replanning," in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016, pp. 5332–5339.

[62] J. Tordesillas, B. T. Lopez, J. Carter, J. Ware, and J. P. How, "Real-Time Planning with Multi-Fidelity Models for Agile Flights in Unknown Environments," in 2019 International Conference on Robotics and Automation (ICRA), 2019, pp. 725–731.

[63] M. Otte and E. Frazzoli, "RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning," International Journal of Robotics Research, vol. 35, no. 7, pp. 797–822, jun 2016.

[64] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, "EGO-Planner: An ESDF-Free Gradient-Based Local Planner for Quadrotors," IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 478–485, aug 2021. [Online]. Available: http://arxiv.org/abs/2008.08835

[65] K. Mohta, M. Watterson, Y. Mulgaonkar, S. Liu, C. Qu, A. Makineni, K. Saulnier, K. Sun, A. Zhu, J. Delmerico, and Others, "Fast, autonomous flight in GPS-denied and cluttered environments," Journal of Field Robotics, vol. 35, no. 1, pp. 101–120, 2018.

[66] P. Fiorini and Z. Shiller, "Motion planning in dynamic environments using velocity obstacles," The international journal of robotics research, vol. 17, no. 7, pp. 760–772, 1998.

[67] J. Van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in 2008 IEEE international conference on robotics and automation. Ieee, 2008, pp. 1928–1935.

[68] B. Damas and J. Santos-Victor, "Avoiding moving obstacles: the forbidden velocity map," in 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009, pp. 4393–4398.

[69] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in Distributed autonomous robotic systems. Springer, 2013, pp. 203–216.

[70] N. Malone, H.-T. Chiang, K. Lesser, M. Oishi, and L. Tapia, "Hybrid Dynamic Moving Obstacle Avoidance Using a Stochastic Reachable Set-Based Potential Field," IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1124–1138, 2017.

[71] H. Febbo, J. Liu, P. Jayakumar, J. L. Stein, and T. Ersal, "Moving obstacle avoidance for large, high-speed autonomous ground vehicles," in 2017 American Control Conference (ACC), 2017, pp. 5568–5573.

[72] J. Lin, H. Zhu, and J. Alonso-Mora, "Robust vision-based obstacle avoidance for micro aerial vehicles in dynamic environments," in 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020, pp. 2682–2688.

[73] W. Luo, W. Sun, and A. Kapoor, "Multi-Robot Collision Avoidance under Uncertainty with Probabilistic Safety Barrier Certificates," in Advances in Neural Information Processing Systems, vol. 33, 2020.

[74] C. Cao, P. Trautman, and S. Iba, "Dynamic channel: A planning framework for crowd navigation," in 2019 International Conference on Robotics and Automation (ICRA). IEEE, 2019, pp. 5551–5557.

[75] D. Zhu, T. Zhou, J. Lin, Y. Fang, and M. Q.-H. Meng, "Online state-time trajectory planning using timed-esdf in highly dynamic environments," arXiv preprint arXiv:2010.15364, 2020.

[76] Y. Wang, J. Ji, Q. Wang, C. Xu, and F. Gao, "Autonomous flights in dynamic environments with onboard vision," arXiv preprint arXiv:2103.05870, 2021.

[77] B. Zhou, Y. Zhang, X. Chen, and S. Shen, "FUEL: Fast UAV Exploration Using Incremental Frontier Structure and Hierarchical Planning," IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 779–786, oct 2021. [Online]. Available: http://arxiv.org/abs/2010.11561

[78] T. Liu, Q. Wang, X. Zhong, Z. Wang, C. Xu, F. Gao, F. Zhang, and F. Gao, "Star-Convex Constrained Optimization for Visibility Planning with Application to Aerial Inspection," in arXiv preprint arXiv:2204.04393, 2022.

[79] G. Chen, W. Dong, X. Sheng, X. Zhu, and H. Ding, "An active sense and avoid system for flying robots in dynamic environments," IEEE/ASME Transactions on Mechatronics, vol. 26, no. 2, pp. 668–678, 2021.

[80] J. Bialkowski, M. Otte, S. Karaman, and E. Frazzoli, "Efficient collision checking in sampling-based motion planning via safety certificates," International Journal of Robotics Research, vol. 35, no. 7, pp. 767–796, jun 2016.

[81] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in Robotics research. Springer, 2016, pp. 649–666.

[82] H. Chen and P. Lu, "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). IEEE, 2020, pp. 5693–5699.

[83] J. Tordesillas, B. T. Lopez, and J. P. How, "FASTER: Fast and Safe Trajectory Planner for Flights in Unknown Environments," in 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019, pp. 1934–1940.

[84] M. Burri, H. Oleynikova, M. W. Achtelik, and R. Siegwart, "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, pp. 1872–1878.

[85] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments," IEEE Robotics and Automation Letters, vol. 2, no. 3, pp. 1688–1695, jun 2017.

[86] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding Horizon "Next-Best-View" Planner for 3D Exploration," in 2016 IEEE International Conference on Robotics and Automation (ICRA), 2016, pp. 1462–1468.

[87] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in Proc. 1996 Int. Conf. Knowledg Discovery and Data Mining (KDD '96), 1996, pp. 226–231.

[88] K. Bernardin, A. Elbs, and R. Stiefelhagen, "Multiple object tracking performance metrics and evaluation in a smart room environment," in Sixth IEEE International Workshop on Visual Surveillance, in conjunction with ECCV, vol. 90, no. 91. Citeseer, 2006.

[89] H. Zhu and J. Alonso-Mora, "Chance-constrained collision avoidance for mavs in dynamic environments," IEEE Robotics and Automation Letters, vol. 4, no. 2, pp. 776–783, 2019.

[90] W. Zhang, S. Wei, Y. Teng, J. Zhang, X. Wang, and Z. Yan, "Dynamic obstacle avoidance for unmanned underwater vehicles based on an improved velocity obstacle method." Sensors, vol. 17, no. 12, p. 2742, 2017.

[91] M. Danelljan, G. Häger, F. S. Khan, and M. Felsberg, "Discriminative Scale Space Tracking," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 8, pp. 1561–1575, 2017.

[92] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-Speed Tracking with Kernelized Correlation Filters," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 3, pp. 583–596, 2015. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2014.2345390

[93] H. Chen and P. Lu, "Real-time identification and avoidance of simultaneous static and dynamic obstacles on point cloud for uavs navigation," Robotics and Autonomous Systems, vol. 154, p. 104124, 2022.

[94] D. Satkunskiene, V. Grigas, V. Eidukynas, and A. Domeika, "487. acceleration based evaluation of the human walking and running parameters." Journal of Vibroengineering, vol. 11, no. 3, 2009.

[95] J. L. Blanco and P. K. Rai, "nanoflann: a C++ header-only fork of FLANN, a library for Nearest Neighbor (NN) with KD-trees," https://github.com/jlblancoc/nanoflann, 2014.

[96] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," IEEE Robotics and Automation Letters, vol. 2, no. 3, pp. 1688–1695, 2017.

[97] Z. Wang, X. Zhou, C. Xu, and F. Gao, "Geometrically constrained trajectory optimization for multicopters," arXiv preprint arXiv:2103.00190, 2021.

[98] Z. Wang, H. Ye, C. Xu, and F. Gao, "Generating large-scale trajectories efficiently using double descriptions of polynomials," in IEEE International Conference on Robotics and Automation. Xi'an, China: IEEE, 2021, pp. 7436–7442.

VITA

VITA

Born

Jan, 1994 in Songyuan, Jilin, China.

Education

- Ph. D. in Aeronautical and Aviation Engineering *2019-Present*
  The Hong Kong Polytechnic University, Hong Kong SAR, China
- M. Sc. in Armament Science and Technology *2016-2019*
  Beijing Institute of Technology, Beijing, China
- B. Eng. in Weapon system and engineering *2012-2016*
  Beijing Institute of Technology, Beijing, China

Research Experience

- Visiting Student *2021.9-2022.2*
  FAST Lab (supervised by Dr. Gao Fei), Zhejiang University, China.