Motion Planning for Autonomous Flights: Algorithms, Systems, and Applications

by

Fei Gao

A Thesis Submitted to
The Hong Kong University of Science and Technology
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy
in the Department of Electronic and Computer Engineering

August 2019, Hong Kong

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Fei Gao

Fei Gao

August 2019

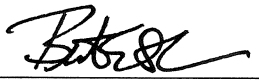
Motion Planning for Autonomous Flights: Algorithms, Systems, and Applications

by

Fei Gao

This is to certify that I have examined the above PhD thesis
and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by
the thesis examination committee have been made.

Prof. Shaojie Shen, Thesis Supervisor



Prof. Bertram Shi, Head of Department

Thesis Examination Committee

1. Prof. Shaojie Shen (Supervisor)	Department of Electronic and Computer Engineering
2. Prof. Lilong Cai	Department of Mechanical and Aerospace Engineering
3. Prof. Jungwon Seo	Department of Electronic and Computer Engineering
4. Prof. Ming Liu	Department of Electronic and Computer Engineering
5. Prof. Rong Xiong (External Examiner)	College of Control Science and Engineering Zhejiang University

Department of Electronic and Computer Engineering
August 2019

Acknowledgments

This thesis would never be completed without help from many people. First of all, I would like to thank my supervisor, Professor Shaojie Shen, for his constructive guidance and inspiring encouragement throughout my Ph.D. study. He taught me what a problem is, how to analyze the problem, how to solve the problem, and how to present my research. I would keep his invaluable teaching in my mind and try to be a good supervisor like him for the rest of my life.

I would like to thank Prof. Rong Xiong, from Zhejiang University, to serve on my Ph.D. defense external committee, and Prof. Wenjing Ye, for being the chairperson of my thesis committee. Also, I would thank all thesis committee members Prof. Lilong Cai, Prof. Ming Liu, and Prof. Junwo Seo, for giving me advice and taking time to join my defense committee. I want to thank my colleagues in Aerial Robotics Group, as well as others in Robotics Institute, for technical discussions in academic and happiness in our daily life. Especially, I thank Boyu and Luqi, for the hardships we went through, the difficulties we overcame together, and the late nights we stayed for debugging. I thank William Wu, for all the sun and wind we endured during field experiments. I thank Tianbo Liu, Zhenfei Yang, and Kunyue Su, for telling me how to be a qualified robotics engineer.

Last but not least, I thank my parents for their selfless and infinite love and support, throughout all the time of my growing-up. I thank the everlasting encouragement and high expectation from my grandpa in heaven. In my memorize, he was always a respectful physics teacher and an enthusiastic educator. And I thank my girlfriend, Xueyi, who has been with me, bravely step into every darkness without any doubts.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Motion Planning for Autonomous Flights: Algorithms, Systems, and Applications

by Fei Gao

Department of Electronic and Computer Engineering

The Hong Kong University of Science and Technology

Abstract

As the development of autonomy in aerial robots, Micro Aerial Vehicle (MAV) has been more and more involved 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, the capability of navigating with full autonomy while avoiding unexpected obstacles is the most crucial requirement. In this thesis, we present methodologies, system designs, and cutting-edge applications, with a focus on motion planning, that enables a quadrotor autonomously navigate unknown complex indoor/outdoor environments using only onboard resources. We start by introducing algorithms for utilizing gradient information in the map to generate a safe and smooth local trajectory. Then we investigate the problem of hard-constrained trajectory planning and propose an approach to generate MAV trajectory, which is guaranteed to be safe and kinodynamic feasible. Based on the above research, we further propose a planning framework which functions directly on point clouds, the most underlying data from different sensor type, without any post-processed maps. After that, we turn to study motion planning in the time domain and propose a method to generate the optimal time parametrization for spatial trajectories. Finally, we investigate and answer the problem of what is the best way to incorporate a human's intention in autonomous and aggressive flight, and what is a flexible, robust and complete aerial system for the human involved applications. We present a system that integrates our past research on motion planning, and state-of-the-art perception and estimation, for aggressively flying

in complex environments and capturing human's intentions. Extensive experimental and benchmarked results, and detailed system set-up are presented throughout the thesis. We conclude this thesis by proposing future research opportunities.

CHAPTER 1

INTRODUCTION

With the development of mechanism and autonomy, mobile robots have been widely used in field applications, such as disaster response, aerial videography, field surveillance, and search-and-rescue, to improve the working efficiency and replace the labor of humans. In these missions, the narrowness of confined environments sets strict constraints on the size, weight, and power of the robots. And many scenarios that are inaccessible or dangerous to human and ground robots permit only the aerial entrance or inspection from flying robots. Therefore, the Micro Aerial Vehicle (MAV), especially quadrotors, are drawing increasing attention in both academia and industry.

For accomplishing such missions in challenging environments, the capability of navigating with full autonomy while avoiding unexpected obstacles is the most fundamental requirement. Among all components in an autonomous aerial system, motion planning works as the brain which coordinates other modules and commands movements of the drone, and is, therefore, the most crucial functionality. The topic of quadrotor motion planning has been studied extensively over the past few decades. However, due to massive computational costs and over-simplified environmental assumptions, most existing methods are incapable of applying to real-world MAV systems in challenging scenarios. Therefore, this leaves the full autonomous navigation of MAV still an open question.

In this thesis, the primary consideration is to develop fast, complete, and safety guaranteed motion planning methods that suit onboard usage. To this end, several studies on related topics motivate this thesis. In this thesis, we review and introduce our past works in the scope of quadrotor motion planning, from the perspective of not only the methodology but also the system design and experimental validation. In the following, we firstly discuss related research problems by categories.

1.1 Research Problems

1.1.1 Path Finding in Complex Environments

A typical motion planning framework for MAV can be roughly divided as a front-end path finding module that finds an initial feasible motion plan in low dimensional space, and a back-end trajectory optimization module in which the smoothness and meets dynamical requirements of the robot. The basic role of path finding is to search for an initial solution in a relatively low dimensional space, such as the $R(3)$ for 3D environments, instead of the full state space of a robot. The problem of path finding is extensively studied over the past decades. However, finding a path with minimum computing effort in complex natural environments is still challenging. Also, most existing path finding methods rely on a post-processed discrete map, such as the grid map or the Octo-map to conduct the searching. These maps provide fast query operations used in path finding. However, they always lose their fidelity when they are not built with a fine resolution in real-time onboard usage. Developing a path finding method independents with map representations and adapts to different sensor types is therefore desirable,

1.1.2 Energy-efficient Trajectory Generation

Trajectory optimization is essential in generating safe and executable trajectories given an initial feasible path. The minimum-snap trajectory optimization is proposed by Mellinger [55], where piecewise polynomials are used to represent the quadrotor trajectory and are optimized by quadratic programs (QP). A method for solving a closed-form solution of the minimum snap is proposed in [71]. In this work, a safe geometric path is firstly found to guide the generation of the trajectory. By adding intermediate waypoints to the path iteratively, a safe trajectory is finally generated after solving the minimum-snap problem several times. Our previous works [32] [29] [31] carve a flight corridor consisting of simple convex shapes (sphere, cube) in a complex environment. The flight corridor constructed by a series of axis-aligned cubes or spheres can be extracted very fast on occupancy map or Kd-tree. Then we use the flight corridor and physical limits to constrain a piecewise Bézier curve, to generate a guaranteed safe and kinodynamic feasible trajectory, Other works are proposed to find general convex polyhedrons for constraining the trajectory. In [53], a piecewise linear path is used to guide and initialize the polyhedron generation. In [14], by assuming all obstacles are convex, SDP and QP are iteratively solved to find the maximum polyhedron seeded

at a random coordinate in 3-D space. Gradient information in maps is also valuable for local trajectory optimization. In CHOMP [70], the trajectory optimization problem is formulated as a nonlinear optimization over the penalty of safety and smoothness. In [58], [51] and [59], gradient-based methods are combined with piecewise polynomials for local planning of quadrotors.

1.1.3 Temporal Trajectory Optimization

Besides generating a safe and smooth trajectory, motion planning also includes planning in the time domain. Time optimization, or so-called time parametrization, is used to optimize the time profile of a trajectory under the physical limits and is critical in generating aggressive motions for a MAV. Finding the optimal time profile is called time-optimal path parametrization (TOPP) problem in the robotics community. TOPP can be achieved by directly considering time duration while searching for the path or optimizing the trajectory. However, the former methods [53] must include a linear dynamic model of the robot and therefore, significantly scale up the complexity of the problem. The later ones [82] tightly couple the time optimization into the inner loop of the energy optimization program, and use nonlinear optimization to solve the nesting problem. Both these two categorized approaches suffer from a heavy computational burden. TOPP can also be solved by decoupling the spatial and temporal optimization for trajectory generation. In this way, the geometric shape of a trajectory is firstly fixed. The trajectory can be parametrized to an arbitrary parameter, not time, but only a virtual variable. Then the optimal mapping function which maps the virtual variable to the shortest time duration under kinodynamic limits is obtained. Two typical categories of methods, Convex optimization [85] and numerical integration [66], are used to optimize the mapping function, Numerical integration [66] [65] has superior performance in computing efficiency. And convex optimization [85] has the advantage of adding regularization terms other than total time into its objective function. This specialty suits well for our application where the user defines the expected aggressiveness of the drone, and sometimes the drone may not be expected to fly as fast as possible.

1.1.4 System Integration

The control, localization, and perception functionalities are necessary for an autonomous quadrotor and are also worthy of attention while integrating a complete aerial system. In recent years, vision-

based localization is widely applied to many systems, among which, looly-coupled [8], or tightly-coupled [68] vision-inertial localization is the most popular solution. In situations where a relatively large payload is available, LiDAR can also be used to localize the MAV and build its surrounding environments at the same time [93] [17]. For perception, RGBD [89] or stereo [81] cameras are most used sensors in the mapping function of a MAV. In our previous research, we also show that a monocular camera is sufficient to equip a drone with all necessary autonomy to navigate in complex envioronment [51].

However, in this thesis, we put emphasis on motion planning and integrate existing localization and perception methods into our system. Given onboard state estimates, we can use a variety of existing control strategies to stabilize the MAV. For MAVs operating at low speeds with small changes in acceleration, a linear controller that assumes near hover state can be used [57]. For more aggressive motions that involve significant attitude changes, a controller with a nonlinear error metric is a more appropriate choice [46].

1.2 Thesis Overview

Motion planning for MAVs has several critical requirements; they are:

1. Safety. Safety enjoys the highest priority while developing an online motion planning system since it protects the robots and operators from having unexpected collisions. The planning methods presented in this thesis guarantee the safety of the drone in their first place.

2. Feasibility. Like other mobile robots, MAVs have model-dependent kinematics and limited control bandwidth. Therefore, the kinodynamic feasibility of the generated motion plans must be satisfied to ensure the trajectories can be executed by the drone.

3. Smoothness. Energy efficiency is another critical issue in motion planning. A smooth trajectory not only respects the continuous nature of an aerial robot but also saves energy consumed by the flight.

These requirements are fulfilled in all the algorithms introduced in this thesis and will be illustrated in detail in the following chapters.

We start the thesis by introducing two typical trajectory generation methods, which are the soft-constrained method and the hard-constrained method. The soft-constrained method (Chap. 3) is good at finding trajectories with multiple objectives and may utilize environmental information to provide a solution with high path clearance. However, it needs a good initial guess to start the optimization warmly and has no guarantee on the global optimality of the final solution. On the contrast, the hard-constrained method (Chap. 4) often have convex formalism and is guaranteed to find a global optimal and strictly feasible solution in its solution space. The drawback of the hard-constrained method is it needs slightly higher computational effort, and often generate a trajectory that is near to obstacles in a complex environmental set-up.

With the pros and cons for each method, the soft-constrained and hard constrained methods are both useful and can be chosen according to the practical requirement on whether high smoothness or high path clearance. However, methods proposed in Chap. 3 and 4 both require a discretized map. To better adapt to different sensors and omit the computational expensive map processing, we propose a planning framework (Chap. 5) to generate trajectories that satisfy the above-mentioned tree requirements directly on point clouds. Up to Chap. 5, these methods majorly focus on optimizing (wrapping) the trajectory in $R(3)$ space to avoid obstacles, while at the same time, adding extra conditions to ensure the kinodynamic feasibility. In these methods, temporal optimality is not the first concern, and time allocation is obtained by heuristic at the most time. In Chap. 6, instead, we look into the time parametrization of the MAV planning and seek for optimizing the time profile of an aerial trajectory. Finally, in Chap. 7, we integrate our previous research and propose a new framework to capture user's intention in general aerial robotic applications. Global spatial trajectory optimization, optimal time allocation, free space extraction, local trajectory re-planning are integrated and improved in this chapter. The thesis is concluded, and future research is discussed in Chap. 8.

1.3 Thesis Contributions

This thesis contributes to several aspects of aerial robotics motion planning. Novel planning algorithms, as well as detailed system designs and extensive real-world validations, are presented throughout this thesis. Herein, we summarize the concrete contributions of each chapter in this thesis.

In Chap. 3, we propose a local trajectory optimization method utilizing gradient information in environments to wrap trajectory out of dangerous regions and keep smoothness. The planning problem is formulated as a typical nonlinear optimization program and is then solved efficiently. The method is validated by a fully autonomous drone operating in complex and natural environments.

In Chap. 4, a novel planning framework that can guarantee the safety and dynamical feasibility of the generated trajectories is proposed. In this chapter, a collision-free flight corridor which consists of a series of axis-aligned cubes is firstly carved on an occupancy grid map. Then the derivation of a hard-constrained planning formulation is given as a Quadratic Program (QP).

Then in Chap. 5, we discuss the topic of online planning without a discrete post-processed map as in Chaps. 3 and 4. A method, safe-region RRT*, along with a trajectory optimization method, is proposed in this chapter, to enable a quadrotor autonomously navigates directly on point clouds. We present autonomous flights with a LiDAR-based quadrotor, where point clouds are used for both localization, mapping, and planning.

Chap. 6 focuses on temporal planning, which finds the optimal time allocation given a fixed spatial curve of the quadrotor, instead of generating safe trajectories among cluttered obstacles. We follow classical methods in robotics time-optimal path parametrization and design an efficient tool to obtain the best time allocation given an arbitrary quadrotor trajectory.

A complete and robust autonomous flight system, *Teach-Repeat-Replan* is introduced in Chap. 7. In this system, the motion planning of quadrotor is divided as global planning, which is done by spatial and temporal trajectory optimization iteratively on an offboard computer; and local planning, which is done onboard the drone to online re-plan safe motions against unmapped obstacles. *Teach-Repeat-Replan* also includes a novel mechanism to capture humans' intentions and build the topological equivalent solution space, which serves as the foundation to generate topo-equivalent motions as the human demonstrations. Autonomous drone racing, as well as other real-world challenging flight experiments, are presented to validate the robustness and efficiency of the proposed system.

CHAPTER 2

SCIENTIFIC BACKGROUND AND LITERATURE REVIEW

2.1 Autonomous Aerial Systems

Enabling the quadrotor to navigate autonomously from a starting position to a targeting position is the most fundamental requirement for a UAV motion planning module. For this kind of mission, the simplest set-up is by assuming all information about the environment is known, and the map representation is already built before the quadrotor flight. In the state-of-the-art optimization-based quadrotor planning framework [71], the dense map of the environment is pre-built by OctoMap [91] before the planning. Then RRT* is used to find a collision-free path in the map. Based on the path, a piecewise polynomial trajectory is generated by iteratively adding collision-free intermediate waypoints. And the time duration of the trajectory is also iteratively optimized by the numeric time optimization proposed by the autors [71]. The final generated trajectory after several seconds calculation is executed by the quadrotor with onboard state estimation and control functions.

The map of surrounding obstacles of the MAV can also be built during the flight. In [4], the fast autonomous flight of a fixed-wing UAV is presented. In this work, control, estimation, and planning are all running onboard a mini embedded computer. The authors propose the pushbroom stereo method to calculate the disparity of a pair of stereo cameras efficiently. This algorithm only estimates the depth results at a fixed distance, thus significantly saves computational cost with the sacrifice of mapping accuracy. Motion planning is done by selecting a local trajectory from an offline built trajectory library. Similarly, to enable the agile and fast flight of fix-wing vehicles in cluttered environments, a two-layer motion primitive method is proposed in [1]. This method consists of two families of primitives. The first is a family of time-delay dependent 3D circular path connecting two points in space, and the second family is an aggressive turn-around (ATA) maneuvers which are used by the aerial vehicle to retreat from dead-ends.

Due to the limited sensing range of a UAV, conducting only local planning in the perception range is reasonable, although with the sacrifice of the global completeness. In [60], the authors

present a local planning pipeline to select a promising local target and optimize a piecewise polynomial trajectory towards that local target. At each replanning epoch, a bunch of random coordinates is drawn in the unoccupied space of the TSDF [34] map. Then the best local target which minimizes a cost function of weighting information gain and distance penalty is selected. Note that since there is no global planning, this works naturally lacks the completeness of global navigation. Which means that one can only hope that the quadrotor may arrive at the global target by conducting numerous local planning. In [20], the authors present an online motion-planning framework for active local collision avoidance. Local trajectories are generated by using CHOMP [70] to optimize the best local path selected from 27 pre-computed paths, which are designed offline for a corridor-like environment where obstacles only lie on two sides. In this way, the planning method is only suitable for particular environments, such as shipboard environments and cannot be applied to general search-and-rescue missions in unstructured, complex environments.

Most system-level autonomous quadrotor works use the combination of local and global motion planning. Representative works are presented in [18], [51] and [56]. In [18], a quadrotor system is proposed with multimodal sensors consisting of a 3-D laser scanner, two stereo camera pairs, ultrasonic distance sensors, and GPS. The motion planning framework in the paper is hierarchical. For global planning, an initial static environment model is needed as prior; for local planning, a reactive collision-avoidance method is used to generate a repelling force on the vehicle to push it away from collisions. However, the reactive collision avoidance method has no safety and dynamical feasibility guarantee and often outputs low-quality non-smooth paths. Moreover, the requirement of a prior map makes the system inapplicable for most search-and-rescue applications in unknown environments. In [51], a complete vision-based quadrotor system, is proposed. In this article [51], the quadrotor uses visual-inertial fusion to do localization and dense 3d reconstruction. Based on that, a local planning method is adopted from [28]. In this method, the local trajectory is generated by conducting gradient-based nonlinear optimization on the local path, and a sampling-based global path is also used to guide the search of the local path. When there is no path exists, or the quadrotor planning is in local minima, the global planner is used to find a new path; otherwise, local planner generates local trajectories towards the target position. Similarly, in [56], a quadrotor equipped with Lidar, stereo camera and IMU is built. The authors [56] also combine a corridor-based local planner adopted from their previous work [53] and the A* global planner in the motion planning framework.

2.2 Path Finding in 3D Environments

Roughly speaking, the motion planning problem can be divided into front-end discrete path finding and back-end continuous trajectory optimization. For the front-end path finding, methods ranging from sampling-based [43] to searching-based [50] have been proposed and applied. In the representative sampling-based method, rapidly-exploring random tree (RRT) [43], samples are drawn randomly from the configuration space to guide a tree to grow towards the planning target. RRT is good at efficiently finding a feasible path. However, it has been proven that RRT has no asymptotical optimality and will converge to the homotopy of the first feasible solution [39]. Asymptotically optimal sampling-based methods, including PRM*, RRG, and RRT*, in which the solution converges to the global optimality as the samples increase, have also been proposed in [39]. RRG is an extension of the RRT algorithm, as it connects new samples not only to the nearest node but also to all other nodes within a range, and a path is searched after the construction of the graph. Also, in PRM*, connections are attempted between roadmap vertices, which are within a range. RRT*, meanwhile, has a rewire mechanism to locally reconnect nodes in the tree and keep the shortest path from the root node to each leaf node. Under the same category, the approach in [88] combines a fixed final state and free final time-optimal controller with the RRT* method to ensure asymptotic optimality and kinodynamic feasibility of the path. In this method, the optimal state transition trajectory connecting two states is calculated by solving a linear quadratic regulator (LQR) problem. Another method that combines RRG and the belief roadmap was proposed by Bry and Roy et al. [9]. In this method, a partial ordering is introduced to trade-off belief and distance while expanding the graph in the belief space.

Searching-based methods discretize the configuration space and convert the path finding problem to graph searching. The graph can be defined in a 2-D, 3-D, or higher-order state space. Typical methods include Anytime Repairing A* (ARA*) [50], Jump Point Search (JPS) [35], and hybrid A* [16]. ARA* accepts a suboptimal solution given a time limit, then reuses search efforts from previous executions to improve the optimality of the path. JPS is only applied on a uniform-cost grid lattice to prune the neighbors of a node being searched. In some cases, JPS can potentially reduce its running time by order of magnitude compared to the traditional A* without sacrificing the optimality. Hybrid A* considers the dynamic model of a robot to steer the extension of a state for generating the graph and searching a dynamically feasible path. Methods using the state-lattice [48] construct an action space consisting of motion primitives as well as a heuristic

look-up table offline. The path is then online searched by a graph search method like AD* [49]. Similarly, in [52], motion primitives are online computed using a time-optimal LQR control policy, and a search graph is online expanded. Usually, the path found by the front-end cannot be directly executed by vehicles since it may be discontinuous or contain unnatural swerves. Therefore, the path is further optimized to generate a safe, continuous, and dynamically feasible trajectory, which is considered as executable.

2.3 Hard Constrained Trajectory Optimization

The minimum-snap trajectory generation algorithm proposed by Mellinger and Kumar [55] is a pioneering work. As presented by the authors, it is possible to reduce the full state space of a quadrotor system to the 3-D position, yaw angle and their derivatives, since the system enjoys the property of differential flatness. As a result, the trajectory can be represented by smooth piecewise polynomials as in (2.1) with bounded derivatives.

$$f_\mu(t) = \begin{cases} \Sigma_{j=0}^{N} p_{1j}(t-T_0)^j & T_0 \le t \le T_1 \\ \Sigma_{j=0}^{N} p_{2j}(t-T_1)^j & T_1 \le t \le T_2 \\ \quad\vdots & \quad\vdots \\ \Sigma_{j=0}^{N} p_{Mj}(t-T_{M-1})^j & T_{M-1} \le t \le T_M \end{cases} \tag{2.1}$$

The optimal trajectory with respect to snap can be generated by solving a quadratic programming (QP) problem that minimizes the integral of squared snap (2.2).

$$J = \sum_{\mu\in\{x,y,z\}} \int_0^T \left(\frac{d^{k_\phi} f_\mu(t)}{dt^{k_\phi}} \right)^2 dt = \quad \eta^T \mathbf{Q}\, \eta \tag{2.2}$$

where η is a vector of polynomial coefficients and $\mathbf{Q}$ is the Hessian matrix of the objective. Richer, Bry, and Roy [71] showed that the minimum snap trajectory could be obtained in closed form. Instead of looking for the optimal polynomial coefficients directly, the coefficients are mapped to the derivatives at each segment points of the piecewise polynomial by a mapping matrix. These derivatives are then separated into fixed and free derivatives by a selection matrix, and the optimal free derivatives can be found in close-form:

In their proposed framework, the dense map is built before the quadrotor flight. Then a path is found by calling RRT* in the complicated map. Based on the path, the closed-form piecewise min-

imum snap trajectory generation algorithm is applied to convert the path to an optimized quadrotor trajectory, which passes all vertex of the RRT* path. This method is also often called a waypoint-based method since its shape is determined by the waypoints of the front-end path. Although the path is surely collision-free, the overshoot in the generated minimum-snap trajectory may have collisions with obstacles. Therefore, the trajectory is then checked, whether it is collision-free. If the trajectory has collisions in one of the pieces, then an intermediate waypoint is added at the middle of this piece, and the trajectory is re-generated. The safety of the generated trajectory is achieved by iteratively adding intermediate waypoints and checking the collision. In the paper [], the authors also proposed a method to do the time allocation of the piecewise polynomial trajectory.

Deits and Russ [15] used IRIS (iterative regional inflation by semidefinite programming) to compute the convex region of safe space. Then they performed a mixed-integer optimization to assign the polynomial trajectories to the convex regions. To ensure that the entire trajectory is within the safe space, they introduced a sums-of-squares (SOS) programming technique. Inspired by IRIS, Liu [53] proposed a method to generate trajectory using the concept of a safe flight corridor, which is a sequence of overlapping convex polyhedra. Jump Point Search (JPS) is first used in uniform-cost grid maps to get a piecewise initial path and reduce the running of the A* algorithm by order of magnitude. Then the set of convex polyhedra are found by an iteratively ellipsoid shrinking and dilating procedure. These polyhedra provide linear inequality constraints in the following quadratic programming, which generate a smooth piecewise polynomial trajectory. Compared with IRIS, the resulted trajectories are very similar, but the time consumed in finding convex regions is much faster. Chen et. al [11] proposed to generate free-space flight corridors consisting of 3-D cubes by using efficient operations in the octree-based environment representation. The standard A* algorithm is used to generate an initial sequence of connected 3-D grids. These initial grids are then inflated to maximal volume using efficient operations in the octree data structure. Finally, an efficient quadratic programming approach is applied to generate a smooth trajectory that fits entirely within the flight corridor and satisfies higher-order dynamical constraints.

In our previous research, we proposed a method to generate trajectory directly on point clouds [31]. A point cloud map of the environment is built incrementally using a 3-D laser range finder. Based on the map, a sampling-based path finding method is adapted to generate a safe space flight corridor consisting of a sequence of spheres. The path finding algorithm utilizes KD-tree for fast nearest neighbor search and is efficient. Then a smooth trajectory that fits within the spherical corridor and

is kinodynamically feasible is generated in a similar manner to J. Chen [11]. The same problem that exists for both of these methods is that it sometimes takes many iterations to obtain a feasible solution. Their algorithms iterate while the polynomial violates feasibility constraints and more constraints are added to the quadratic programming. Although it was proven in [11] that a feasible trajectory could be generated within a finite number of iteration, the approach is inefficient sometimes. In [10], trajectories with piecewise constant accelerations are generated. The authors derived the velocity bound as well as the maximum deviation between the trajectory and the path, respecting the maximum acceleration and minimum path clearance. Since the problem is formulated as a convex optimization problem that is guaranteed to be feasible, no iterative procedure is needed as previous methods. However, the bound is very conservative, making the generated trajectory's speed always deficient.

One of the key factors that significantly influence the quality of the piecewise trajectory is time allocation. However, the initial path found by the above-listed methods contains no hints for proper time allocation. Therefore, in the following trajectory optimization, the time progress is often estimated by some naive heuristic. In [32], a fast marching-based method in a velocity field is applied to search a time-indexed path. Since the velocity field is derived from a Euclidean signed distance field, the resulted time allocation along the path adapts better to obstacle densities in environments. Besides, the Bernstein basis is used to represent the trajectory, which results in piecewise Bezier curves. The convex hull property of the Bezier curve is utilized so that the safety and high-order dynamical feasibility can be bounded within feasible space efficiently.

2.4 Soft Constrained Trajectory Optimization

The methods mentioned in the previous sections set hard constraints for the optimization variables that are required to be satisfied. For many other methods, however, constraints are penalized in the objective function. CHOMP [97] is a trajectory optimization method that minimizes smoothness and collision costs locally on a discrete-time trajectory. With discrete waypoints as optimization variables, the planner performs gradient descent in each step. The algorithm can find smooth and collision-free trajectories from straight-line initialization, which might not be collision-free. However, in cluttered environments, the success rate of the method is low. STOMP [38] is also based on optimization using a two-part objective function. In contrast, the optimization is solved

by gradient-free candidate sampling and combining the best-scoring candidates linearly.

Inspired by CHOMP, [74] presented an approach based on sequential convex optimization. The workspace is broken up into convex free regions, and sequential convex optimization that penalizes collisions with a hinge loss is performed. The original non-convex optimization problem is solved by constructing and solving approximate convex subproblems repeatedly, each of which generates a step that makes progress on the original problem. Since convex optimization algorithms minimize each subproblem efficiently, the proposed algorithm converges faster than the previous two methods. The main drawback of the method is that convex regions are difficult to compute online and thus require a pre-built map with convex regions.

In [58], the authors propose an online continuous-time trajectory optimization method for local replanning. Unlike previous methods, continuous-time trajectory representation is used because dynamic constraints can be more accurately expressed, and it avoids numerical differentiation errors. The collision cost of the continuous-time trajectory is formulated as the line integral of the distance penalty over the arc length along the trajectory. The integral is discretized on different time stamps for numerical calculation:

$$\begin{aligned} f_o &= \int_{T_0}^{T_M} c(p(t))ds \\ &= \int_{T_0}^{T_M} c(p(t))\|v(t)\|dt = \sum_{k=0}^{\tau/\delta t} c(p(\mathcal{T}_k))\|v(t)\|\delta t \end{aligned} \tag{2.3}$$

where $\mathcal{T}_k = T_0 + k\delta t$, and $v(t)$ is the velocity at position $p(t)$. To improve the efficiency of the optimization, the coefficients of a polynomial are mapped to end-derivatives of each segment, as proposed in [71]. This mapping transforms a constrained optimization problem into an unconstrained programming problem, which is much faster to solve. The gradient of the discretized distance cost is needed for solving this non-linear optimization problem, which can be computed efficiently. The main drawback of this method is that it suffers from converging to local minima, and this can only be slightly relieved by several random restarts. [28] adopted a similar formulation with the difference of initializing the trajectory by piecewise straight line generated by an informed sampling-based path generation algorithm. Having a voxel grid map as environment representation, a random-exploring graph is generated, in which the safe volume of a given point is evaluated by using a Kd-tree to do a fast nearest neighbor search. A minimum distance path can be found by

the standard A* algorithm. An informed sampling scheme in which the best path and the heuristic sampling domain are updated iteratively is also utilized to improve the efficiency of the sampling and the quality of the path. Thanks to the better quality of the initial path, this method enjoys a significantly higher success rate compared with CHOMP [97] and CT[58].

Ewok [83] proposed an algorithm to generate a globally smooth and collision-free trajectory and do local replanning that can handle dynamic obstacles while keeping the deviation from the global trajectory small. Instead of using polynomial as trajectory representation, they use uniform b-splines. Since the b-spline trajectory is smooth, given an arbitrary set of control points and has the property of locality, it results in fewer constraints and optimization variables. These properties make it very efficient to do local replanning.

2.5 Time Optimal Path Parametrization

The time parameterization or so-called time allocation dominates the progress of the trajectory in the time domain and is regarded as one of the main reasons that lead to sub-optimality [71]. One way to achieve reasonable time allocation is by iteratively refining allocated times using gradient descent [71]. However, iteratively regenerating the trajectory prevents this method to be used in real-time, especially when the trajectory generation is costly. In our previous work [29], we allocate times for each piece of the trajectory according to the Euclidean distance between the segmentation points and rely on the overlapping regions of the flight corridor to adjust the time allocation. Another standard way is using some heuristic [53] [78] to allocate the time. All of the above methods have no guarantee on the optimality of the time.

Minimum time trajectory or so-called time-optimal trajectory generation aims at fully utilizing the ability of actuators of mobile robots to travel as fast as possible without violating kinodynamic limits. Methods can be divided into direct methods [52], which directly generate spatial-temporal optimal trajectory and indirect methods[72], which generates the trajectory independent of time firstly and then find the relationship between the trajectory and time. The proposed approach in this paper is an indirect method. For direct methods, a recent result is presented in [52], where the trajectory generation problem is formulated as a minimum-time optimal control based graph search. The linear model of the quadrotor is established, and a graph consists of minimum time-controlled motion primitives is formed. This method considers the time optimality when searching

a safe and dynamical feasible trajectory towards the goal. However, the complexity of the search grows exponentially as the graph size increases. Also, linearizing the quadrotor model at the hover point can lead to a high error when flying at high acceleration. In [72], the geometric shape of the trajectory is fixed at a virtual domain firstly, which is the same as our method. Then the mapping function from the virtual parameter to time is optimized by nonlinear optimization. Compared to our approach, this method needs a kinodynamic feasible initial solution, and the convergence rate is slow. Also, it cannot guarantee global optimality. The approach proposed in [37] also fixes the geometric trajectory and finds the mapping function by iteratively adds mapping points along the spatial trajectory to squeeze the kinodynamic feasibility. However, this method also needs numerous numerical iterations, making it unsuitable for onboard usage. Convex optimization [85] and numerical integration [66] are two typical methods of robotics time optimal path parametrization (TOPP) problem. Although numerical integration [66] [65] has shown superior performance in computing efficiency, convex optimization [85] has the advantage of adding regularization terms other than total time into its objective function. This specialty suits well for our application where the user defines the expected aggressiveness of the drone, and sometimes the drone may not be expected to fly as fast as possible. As for the efficiency, since we do temporal optimization off-line before the repeating, computing time is not critical.

2.6 Human Robot Interaction

Humans' intentions are also nonnegligible in many applications. For example, in aerial videography, industrial inspection, and human-robot interactions, human's preferences should dominate the rough route of the MAV. How to better incorporate human intentions and convert it into a flight plan for an autonomous MAV is another hot topic. The most common way to this end is teach-and-repeat, where the operators manually teach the robot his/her expect motions and the robot then repeat these motions autonomously.

Many robotics teach-and-repeat works, especially for mobile robots, have been published in recent years. Most of them focus on improving the accuracy or robustness in repeating/following the path by operators, which is fundamentally different from our motivation. A lidar-based teach-and-repeat system is proposed in [80], where laser scans are used to localize the ground vehicle against its taught path driven by the user. Furgale et al. [25] [42] also develop a lidar-based ground robot,

which is specially designed for repeating long-term motions in highly dynamic environments. This system equips a local motion planner which samples local trajectory to avoid dynamic elements during route following. A map maintenance module is used to identify moving objects and estimate their velocities. An iterative learning controller is proposed in [61], to reduce the tracking error during the repeating of the robot. This controller can compensate disturbances such as unmodelled terrains and environmental changes by learning a feedforward control policy. Vision-based teaching-and-repeat systems are also proposed in several works, such as the visual localization used by the rover in [26]. In this work, the authors build a manifold map during the teaching and then use it for localization in the repeating, In [63], a multi-experience localization algorithm is proposed to address the issue of environmental changes. The ground robot is localized robustly against several past experiences. In [62] and [6], to further improve the accuracy and robustness in localization, illumination and terrain appearances are considered in their proposed visual navigation system used for teach-and-repeat, Compared to ground teach-and-repeat works, research on aerial teach-and-repeat is few. In [21], a vision-based drone is used to inspect infrastructure repetitively. In the teaching phase, the desired trajectory is demonstrated by the operator, and some keyframes in the visual SLAM are recorded as checkpoints. While repeating, local trajectories are generated to connect those checkpoints by using the minimum-snap polynomials [55]. To function properly, in this work, the teaching trajectory itself must be smooth, and the environments must have no changes during the whole repeating.

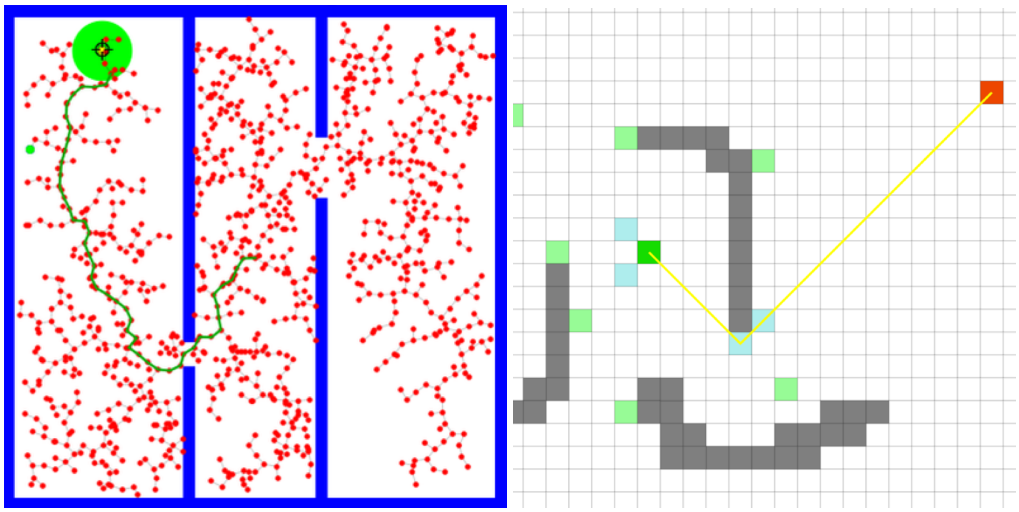
CHAPTER 3

GRADIENT-BASED LOCAL TRAJECTORY OPTIMIZATION

3.1 Research Background and Motivation

In this chapter, we introduce an online collision-free trajectory generation method utilizing gradient information in the MAV's configuration space. Given measurements from onboard sensors, a description of the environment can be established, and a configuration space for the high-level motion planning task is built. In this configuration space, a lot of methods are proposed to generate smooth and collision-free trajectories by combining traditional path finding method such as RRT* [43] and trajectory optimization method such as minimum-snap [55]. Since in the objective, only the energy is considered, trajectories generated in this way may suffer from being close to obstacles, making the quadrotor flying with hazard when there is uncertainty in control and perception. Gradient information, which is naturally encoded in an obstacle map, can be used to improve the clearance of the trajectory.

In this chapter, we introduce our method to improve the trajectory's clearance by utilizing the gradient information in the environments. The focus of this chapter is the back-end trajectory optimization, which fully utilizes the gradient information. And the front-end path finding is of course not the point herein in this chapter. In what follows, we assume a piecewise collision-free path is already found by classical path finding methods such as A* [50], Jump Point Search (JPS) [35], or RRT* [43], or by our other works [32, 31] detailed in later Chaps 4, 5. The proposed method in this chapter can guarantee the safety of the MAV, while at the same time considering the smoothness and dynamical feasibility of the trajectory. We integrate the proposed planning module into a fully autonomous vision-based quadrotor system and present online experiments in unknown complex environments.



(a) A path found by RRT. (b) A path found by JPS.

Figure 3.1. An initial safe path found by a general front-end path finding module.

3.2 Piecewise Polynomial Trajectory Optimization

3.2.1 Problem Formulation

Having a piecewise collision-free path as shown in Fig. 3.1, we parametrize a piecewise polynomials with each segments of it corresponding to one piecewise of the path.

The trajectory is parametrized to the time variable t in each dimension μ out of x, y, z. The N^{th} order M-segment trajectory of one dimension can be written as follows:

$$p_\mu(t) = \begin{cases} \sum_{j=0}^{N} \eta_{1j}(t-T_0)^j & T_0 \le t \le T_1 \\ \sum_{j=0}^{N} \eta_{2j}(t-T_1)^j & T_1 \le t \le T_2 \\ \quad\vdots & \quad\vdots \\ \sum_{j=0}^{N} \eta_{Mj}(t-T_{M-1})^j & T_{M-1} \le t \le T_M, \end{cases} \tag{3.1}$$

where η_{ij} is the j^{th} order polynomial coefficient of the i^{th} segment of the trajectory. $T_1, T_2, ..., T_M$ are the end times of each segment, with a total time $\tau = T_M - T_0$.

We follow [58] to optimize directly on the coefficients of the piecewise polynomial trajectory. The formulation of the complete objective function is written as

$$\min \quad \lambda_1 f_s + \lambda_2 f_o + \lambda_3 (f_v + f_a), \tag{3.2}$$

where f_s is the regularized smoothness term to keep the trajectory smooth, f_o is the cost of the clearance of the trajectory, f_v and f_a are the penalties of the velocity and acceleration exceeding the dynamical feasibility. λ_1, λ_2 and λ_3 are weight parameters to trade off the smoothness, trajectory clearance, and dynamical feasibility. The smoothness term is the integral of the square of the ϕ^{th} derivative. In this paper, we minimize the snap (4^{th} derivative of the position) of the quadrotor to obtain a smooth trajectory, which is good for our vision-based localization and mapping system, so ϕ is 4, and the order N of the polynomial we use is 8.

3.2.2 Gradient Evaluation

We adopt the method proposed in [71] and introduce a mapping matrix $\mathbf{M}$ and a selection matrix $\mathbf{C}$ to map the original variables, the polynomial coefficients, to the derivatives at each segment points of the piecewise polynomial:

$$\boldsymbol{\eta} = \mathbf{M}^{-1}\mathbf{C}\begin{bmatrix}\mathbf{d}_F\\ \mathbf{d}_P\end{bmatrix}. \tag{3.3}$$

Here the matrix $\mathbf{M}$ maps the polynomial coefficients $\boldsymbol{\eta}$ to $\mathbf{d}$, and matrix $\mathbf{C}$ separates the derivatives $\mathbf{d}$ into fixed derivatives $\mathbf{d}_F$ (which are pre-defined before the trajectory generation) and free derivatives $\mathbf{d}_P$ (which are the optimized variables). In this paper, all derivatives at middle waypoints, including the position, velocity and acceleration, are free variables, and derivatives at the start and target points are fixed variables. The cost of the smoothness term can be written as

$$f_s = \begin{bmatrix}\mathbf{d}_F\\ \mathbf{d}_P\end{bmatrix}^T \mathbf{C}^T\mathbf{M}^{-T}\mathbf{Q}\mathbf{M}^{-1}\mathbf{C}\begin{bmatrix}\mathbf{d}_F\\ \mathbf{d}_P\end{bmatrix}. \tag{3.4}$$

Denote $\mathbf{C}^T\mathbf{M}^{-T}\mathbf{Q}\mathbf{M}^{-1}\mathbf{C}$ as matrix $\mathbf{R}$, then the cost function can be written in a partitioned form as

$$f_s = \begin{bmatrix}\mathbf{d}_F\\ \mathbf{d}_P\end{bmatrix}^T \begin{bmatrix}\mathbf{R}_{FF} & \mathbf{R}_{FP}\\ \mathbf{R}_{PF} & \mathbf{R}_{PP}\end{bmatrix}\begin{bmatrix}\mathbf{d}_F\\ \mathbf{d}_P\end{bmatrix}. \tag{3.5}$$

The Jacobian and Hessian of f_s with respect to $\mathbf{d}_p$ can be written as follows, in where $\mu \in$

(x, y, z):

$$\mathbf{J}_s = \left[\frac{\partial f_s}{\partial \mathbf{d}_{P_x}}, \frac{\partial f_s}{\partial \mathbf{d}_{P_y}}, \frac{\partial f_s}{\partial \mathbf{d}_{P_z}}\right], \mathbf{H}_s = \left[\frac{\partial^2 f_s}{\partial \mathbf{d}_{P_x}^2}, \frac{\partial^2 f_s}{\partial \mathbf{d}_{P_y}^2}, \frac{\partial^2 f_s}{\partial \mathbf{d}_{P_z}^2}\right],$$
$$\frac{\partial f_s}{\partial \mathbf{d}_{P_\mu}} = 2\mathbf{d}_F^T \mathbf{R}_{FP} + 2\mathbf{d}_P^T \mathbf{R}_{PP}, \quad \frac{\partial^2 f_s}{\partial \mathbf{d}_{P\mu}^2} = 2\mathbf{R}_{PP}^T, \tag{3.6}$$

For the cost of collision on the trajectory, we need a differentiable cost function to penalize the distance value, and we want the cost to rapidly grow up to infinity at where near the obstacles and to be flat at where away from the obstacles. If we provide a proper collision-free initial trajectory, the collision cost at where near obstacles will dominate the objective function and prevent the trajectory from colliding with obstacles. The cost function we select is an exponential function. At a position in the map with distance value d, the cost $c(d)$ is written as

$$c(d) = \alpha \cdot \exp(-(d - d_0)/r), \tag{3.7}$$

where α is the magnitude of the cost function, d_0 is the threshold where the cost starts to rapidly rise, and r controls the rate of the function's rise. The cost is negatively correlated to the distance value. We follow[58] to formulate the collision cost as the line integral of the distance value over the arc length along the trajectory. For numerical calculation, we can discretize the integral and formulate it as a summation of costs on different time stamps:

$$\begin{aligned} f_o &= \int_{T_0}^{T_M} c(p(t))ds \\ &= \int_{T_0}^{T_M} c(p(t))\|v(t)\|dt = \sum_{k=0}^{\tau/\delta t} c(p(\mathcal{T}_k))\|v(t)\|\delta t, \end{aligned} \tag{3.8}$$

where $\mathcal{T}_k = T_0 + k\delta t$, and $v(t)$ is the velocity at $p(t)$.

The Jacobian of the collision term in a discrete form is

$$\mathbf{J}_o = \left[\frac{\partial f_o}{\partial \mathbf{d}_{P_x}}, \frac{\partial f_o}{\partial \mathbf{d}_{P_y}}, \frac{\partial f_o}{\partial \mathbf{d}_{P_z}}\right]$$
$$\frac{\partial f_o}{\partial \mathbf{d}_{P_\mu}} = \sum_{k=0}^{\tau/\delta t} \left\{\nabla_\mu c(p(\mathcal{T}_k))\|v\|\mathbf{F} + c(p(\mathcal{T}_k))\frac{v_\mu}{\|v\|}\mathbf{G}\right\}\delta t, \tag{3.9}$$

where $\mu \in (x, y, z)$, $\mathbf{L}_{dp}$ is the right block of matrix $\mathbf{M}^{-1}\mathbf{C}$ which corresponds to the free derivatives

on the μ axis $\mathbf{d}_{p_\mu}$. $\mathbf{F} = \mathbf{T}\mathbf{L}_{dp}$, $\mathbf{G} = \mathbf{T}\mathbf{V}_m\mathbf{L}_{dp}$. $\nabla_\mu c(\cdot)$ is the gradient in the μ axis of the collision cost. $\mathbf{V}_m$ is the mapping matrix which maps the polynomial coefficients of the position to the polynomial coefficients of velocity, and $\mathbf{T} = [\mathcal{T}_k^0, \mathcal{T}_k^1, ..., \mathcal{T}_k^n]$ is the vector of a given time t_0. Furthermore, we get the Hessian matrix as follows:

$$\begin{aligned}
\mathbf{H}_o &= \left[\frac{\partial^2 f_o}{\partial \mathbf{d}_{P_x}^2}, \frac{\partial^2 f_o}{\partial \mathbf{d}_{P_y}^2}, \frac{\partial^2 f_o}{\partial \mathbf{d}_{P_z}^2}\right], \\
\frac{\partial^2 f_o}{\partial \mathbf{d}_{P\mu}^2} &= \sum_{k=0}^{\tau/\delta t} \Big\{ \mathbf{F}^T \nabla_\mu c(p(\mathcal{T}_k) \frac{v_\mu}{\|v\|} \mathbf{G} + \mathbf{F}^T \nabla_\mu^2 c(p(\mathcal{T}_k)) \|v\| \mathbf{F} \\
&+ \mathbf{G}^T \nabla_\mu c(p(\mathcal{T}_k)) \frac{v_\mu}{\|v\|} \mathbf{F} + \mathbf{G}^T c(p(\mathcal{T}_k)) \frac{v_\mu^2}{\|v\|^3} \mathbf{G} \Big\} \delta t,
\end{aligned} \tag{3.10}$$

where $\nabla_\mu^2 c(\cdot)$ is the 2^{nd} derivative of the collision cost in μ.

For formulating the cost on the dynamical feasibility, we generate an artificial cost field on velocity between the maximum velocity and minus maximum velocity in the x, y and z axis. And we write f_v as the sum of the line integral of the velocity in x, y and z axis. The cost function of the high-order constraints is also an exponential function, since it is good at penalizing when close to or beyond the limit of bounds and staying flat when away from the bounds, $c_v(v)$ is the cost function applied on the velocity, which has the same form as in Eq.(3.7). The cost of velocity feasibility is:

$$\begin{aligned}
f_v &= \sum_{\mu \in \{x,y,z\}} \int_{T_0}^{T_M} c_v(v_\mu(t)) ds \\
&= \sum_{\mu \in \{x,y,z\}} \int_{T_0}^{T_M} c_v(v_\mu(t)) \|a_\mu(t)\| dt.
\end{aligned} \tag{3.11}$$

The Jacobian and Hessian of $\mathbf{J}_v$ has similar formulation to Eq.(3.9) and Eq.(3.10). We omit the formulation of the acceleration feasibility cost f_a for brevity.

Having the total Jacobian $\mathbf{J} = \lambda_1 \mathbf{J}_s + \lambda_2 \mathbf{J}_o + \lambda_3(\mathbf{J}_v + \mathbf{J}_a)$ and Hessian $\mathbf{H} = \lambda_1 \mathbf{H}_s + \lambda_2 \mathbf{H}_o + \lambda_3(\mathbf{H}_v + \mathbf{H}_a)$, we use the Newton trust region method [79] to optimize the objective. The update equation is:

$$(\mathbf{H} + \lambda \mathbf{I}) \Delta \mathbf{d}_p = -\mathbf{J}^T, \;\; \mathbf{d}_p \leftarrow \mathbf{d}_p + \Delta \mathbf{d}_p, \tag{3.12}$$

where λ is the factor determined by a heuristic and is initialized with a large value to ensure convergence.

The fundamental difference between our method and previous gradient-based methods [58, 70] is twofold:

1. We always require an initial safe and dynamical feasible path to initialize the optimization.
2. We use the exponential function to penalize the collision and dynamical infeasibility costs.

Based on the fact that during the optimization procedure, the objective value should always be descent, which is indeed held in our implementation of the nonlinear optimization. If we provide a proper collision-free and dynamically feasible initial trajectory, then the collision cost, which blooms up near obstacles or near infeasible states will dominate the objective function and prevent the trajectory being colliding or infeasible. That's the key factor that results in a superior high success rate than others [58, 70]. We demonstrate the success rate later in Sect. 3.4.1.

3.2.3 Locally Voxel Caching for Collision Cost Evaluation

In order to evaluate the cost of the collision penalty and get the gradient information along the trajectory, the most commonly used method is to maintain a distance map. But even for a local map with a short-range, the maintenance and updating of the complete distance field are costly. Furthermore, maintaining a distance field map and getting the gradient by taking the difference on the map in x, y and z directions only provides an approximated gradient and can be easily undifferentiable at places where the distance value is indeed not continuous. Thus we propose a new light-weight method to help us evaluate the collision cost.

Having the initial trajectory generated in Sect. 3.2.1, only a small part of the space will be evaluated during the optimization. Thanks to the sparsity of the distribution of the evaluated voxel in the map, we can accelerate the procedure by voxel hashing. We evaluate the cost and gradient of a point on the trajectory and simply hash the voxel it belongs to according to a pre-defined resolution (voxel size). For each point, we do the nearest neighbor search using the K-d tree. Since the tree has been built in path finding, there is no extra cost on building the tree. The nearest neighbor query only takes $O(logN)$ computation time, where N is the number of voxels. In this way, the Euclidean distance in 3-D space of a point to its nearest neighbor is obtained, and at the

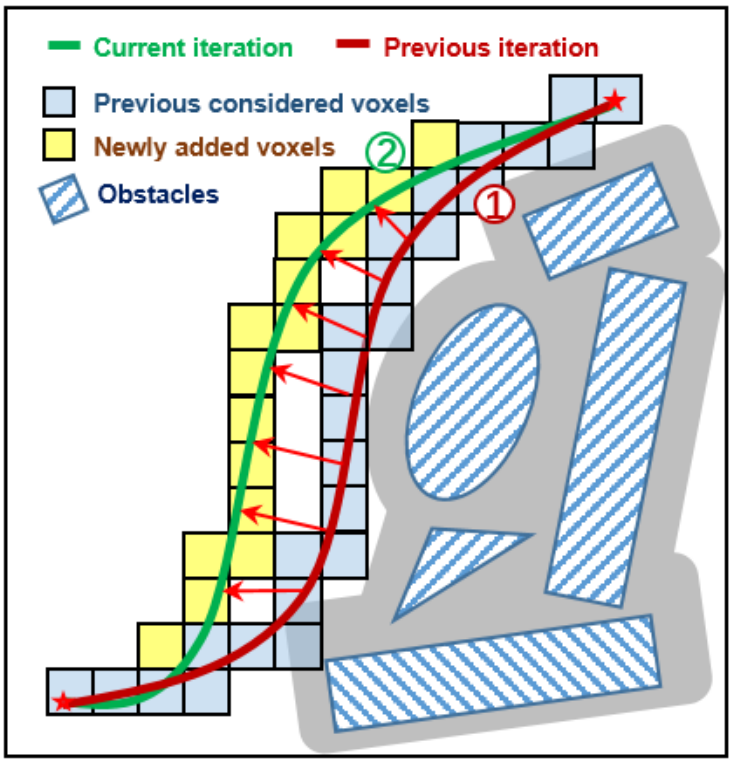
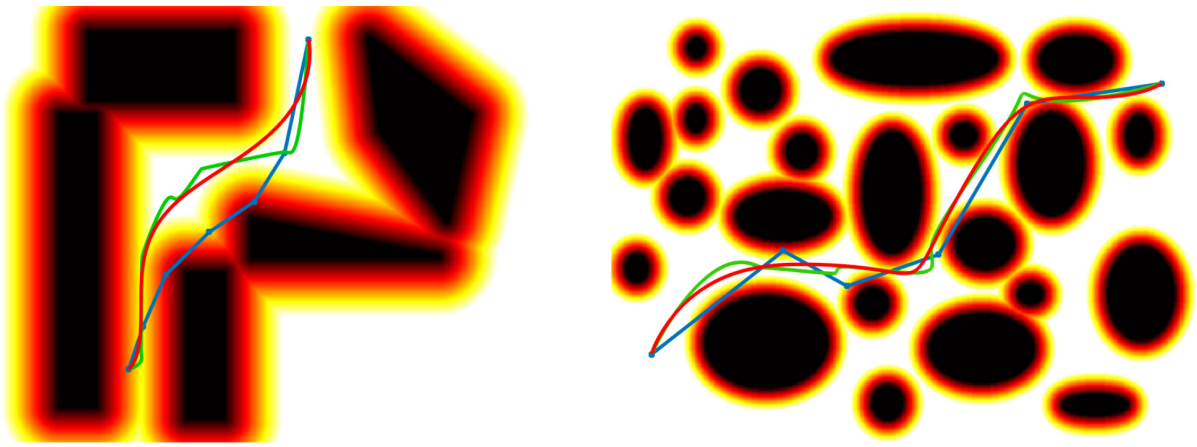


Figure 3.2. Illustration of the voxel hashing. Shapes with shading are obstacles. The red curve and green curve are trajectories before and after one iteration, respectively. Blue voxels have already been recorded, while yellow voxels are newly added. Gray areas are collision margins near to obstacles but still considered as safe. Red arrows roughly show the gradient direction.

same time, we get the position of the nearest neighbor, which directly indicates the direction of the gradient. During the optimization, we cached all voxels which have been queried with the cost and gradient information for further iterations.

3.2.4 Two-Steps Optimization Framework

Nonlinear optimization is applied to smooth the collision-free initial trajectory, and at the same time, ensure the safety and dynamical feasibility. The initial trajectory which geometrically follows the collision-free path (see Sect. 3.2.1) is used as the initial value. We apply a two-steps optimization strategy which can be summarized as follows: 1) Optimize the collision cost only. All the free variables, including positions of middle waypoints on the initial trajectory, will be pushed to the basin of the distance field from the neighborhood of obstacles. 2) Re-allocate the time and re-parametrize the trajectory to time according to current waypoints' positions. Then optimize the



(a) Trajectory generated in a sparse environments (b) Trajectory generated in a dense environments

Figure 3.3. trajectory generated in 2-D sparse and dense map. The color code shows the value of the distance to the nearest obstacles, with darker colors corresponding to a higher cost of a collision. The initial trajectory shown with the blue line is a collision-free but hard-to-follow straight-line trajectory. After the first step in the optimization, the trajectory and middle waypoints are pushed away from the near obstacles, as the green line. The final trajectory is shown in red.

objective with a smoothness term, and dynamical penalty term added. The optimizer smooths the safe trajectories and squeezes the dynamic infeasibility, and at the same time prevents the trajectories from moving near to obstacles again.

Since path finder may place waypoints close to obstacles, optimizing only on the collision cost in the first step can lead the trajectory quickly converge to a much more safe trajectory. After that, the re-allocation of time in each segment of the trajectory according to current waypoints' positions makes the time allocation more appropriate for the following optimization. Note, before the second stage of the optimization, we initialize each piece of the path to a rest-to-rest trajectory with a conservative time allocation to guarantee the dynamical feasibility. The entire optimization procedure finishes when the termination condition is met or the pre-defined time limit for optimization is reached. Some results of the two-steps optimization are shown in Fig. 3.3.

3.3 Implementation Details

3.3.1 System Settings

The planning method proposed in this paper[1] is implemented in C++11 using a general nonlinear optimization solver NLopt[2]. The flight experiments are done on a self-developed quadrotor platform (Fig. 3.5). Motion planning, state estimation, and control modules are running on a dual-core 3.00 GHz Intel i7-5500U processor, which has 8 GB RAM and 256 GB SSD. And the monocular-fisheye mapping module is running at 10 Hz on an Nvidia TX1 which has 256 cores. The software and hardware system architecture of the quadrotor system is shown in Figs. 3.4 and 3.5. To provide a large field of view for the mapping module, we use a 235° fish-eye lens and crop images on it to cover the horizontal FOV. The output dense map is fed into the front-end path searching module of the planning layer. Based on it, the path between the current quadrotor state to the target position is searched. Then the path is optimized to generate a time-parametrized smooth, safe, and dynamically feasible piecewise trajectory. The maximum number of iterations in trajectory optimization is set as 50, and the weights of the smoothness, collision, and dynamical cost are 2.5, 1.0, and 1.0. The range of the local map is set as $5\ m$ to ensure the mapping module works online at a high frequency (10 Hz). A trajectory server is then used to receive the trajectory and convert it to control commands. We use a geometric controller [47] to track the generated trajectory, and the attitude control is left to the DJI A3 autopilot on the quadrotor.

[1]Source code: `https://github.com/HKUST-Aerial-Robotics/grad_traj_optimization`

[2]`https://https://nlopt.readthedocs.io/en/latest/`

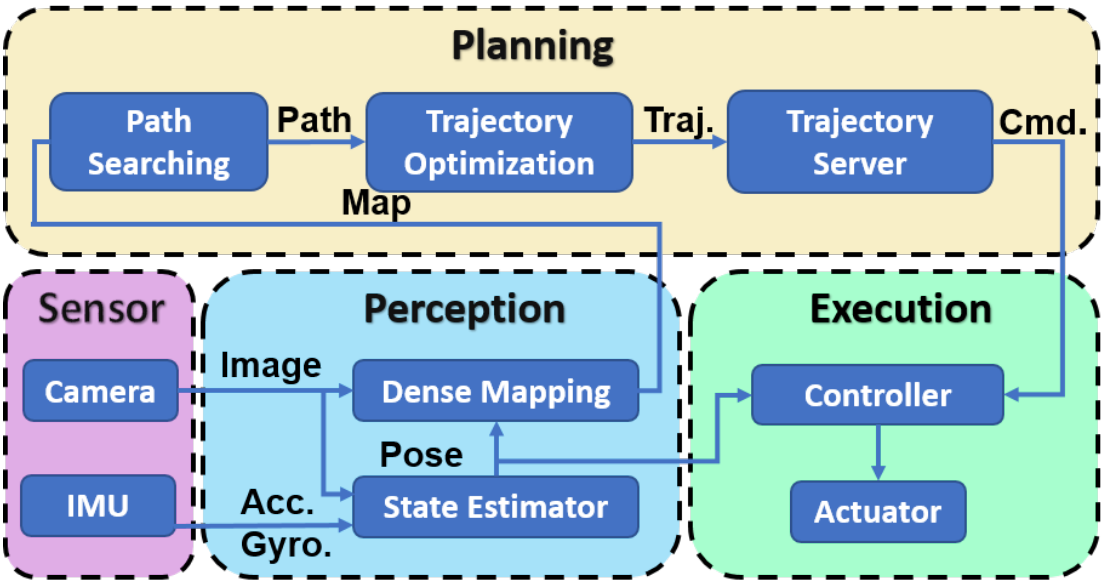


Figure 3.4. The system architecture of the quadrotor platform. The quadrotor uses a monocular fish-eye camera and an IMU for localizing and mapping. The mapping module is running on the Nvidia TX1 GPU cores, while state estimation, control, and motion planning are running on the Intel i7 CPU. All processings are done onboard. The quadrotor is stabilized by a DJI A3 autopilot.

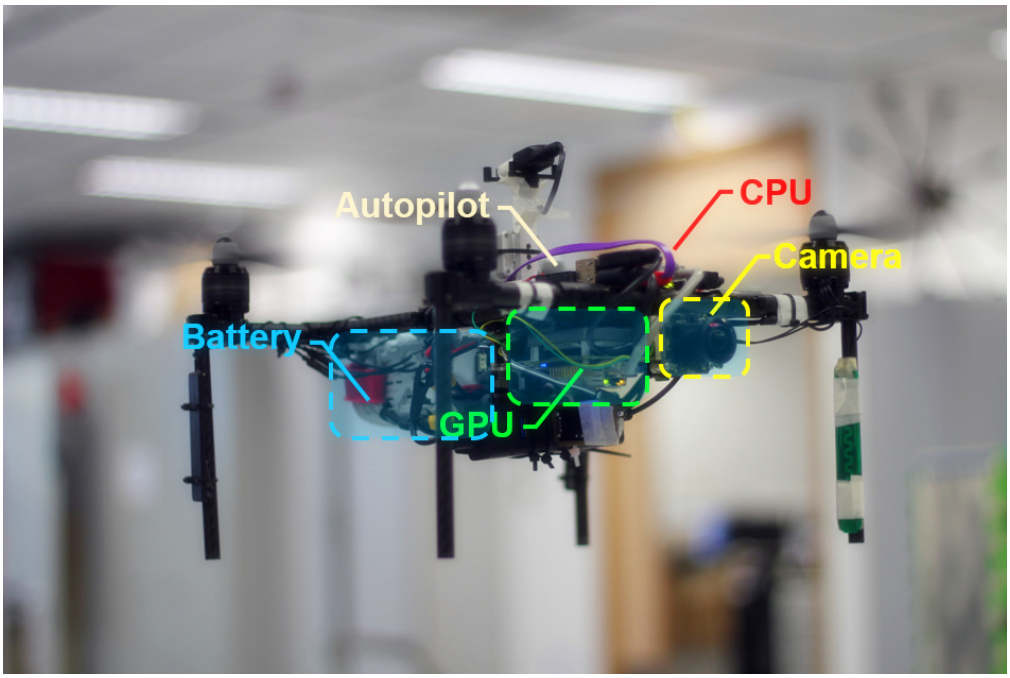


Figure 3.5. The hardware setting of the qyadrotor platform, which consists of a monocular camera, a CPU, a GPU, and an autopilot.

3.3.2 State Estimation and Dense Mapping

Referring to our previous work [92], the 6-degrees of freedom state estimation results, which are provided by a tightly-coupled visual-inertial fusion framework, are fed into the feedback control as well as the motion stereo mapping. The mapping module fuses each keyframe's depth map into a global map by means of truncated signed distance fusion (TSDF). A frontward-looking fish-eye camera with a large field of view of 235° is used to ensure a wide surrounding environment perception. The state estimation module runs at 20 Hz, and the dense mapping module outputs a local map with a range of $5\ m$ at 10 Hz.

3.3.3 Incremental Planning Scheme

Due to the limited sensing range of our mapping module, we use a receding horizon incremental planning strategy. Before the flight starts, a target point is set, and a global path finder is utilized to find a global path to reach the target. During the navigation, a local planner is used to find the path and optimize the trajectory within the planning horizon Ψ_p. The trajectory will only be executed in an execution horizon Ψ_e, after which the local planner will be called again, as is shown in Fig. 3.6. And if the newly updated map has a collision with the trajectory which is going to be executed, the local planner will be called immediately. The planning target for the local planner is selected firstly on the global path, and if the target is occupied (e.g., has obstacles), the planner will automatically adjust the target and search for a free point in a nearby region. In the motion planning module, both the front-end path finding and back-end trajectory optimization part have pre-defined time limits, which control the termination conditions of the whole pipeline. Note that all unknown space (outside the perception range) is treated as free in the global path finder and is treated as occupied in the local planner.

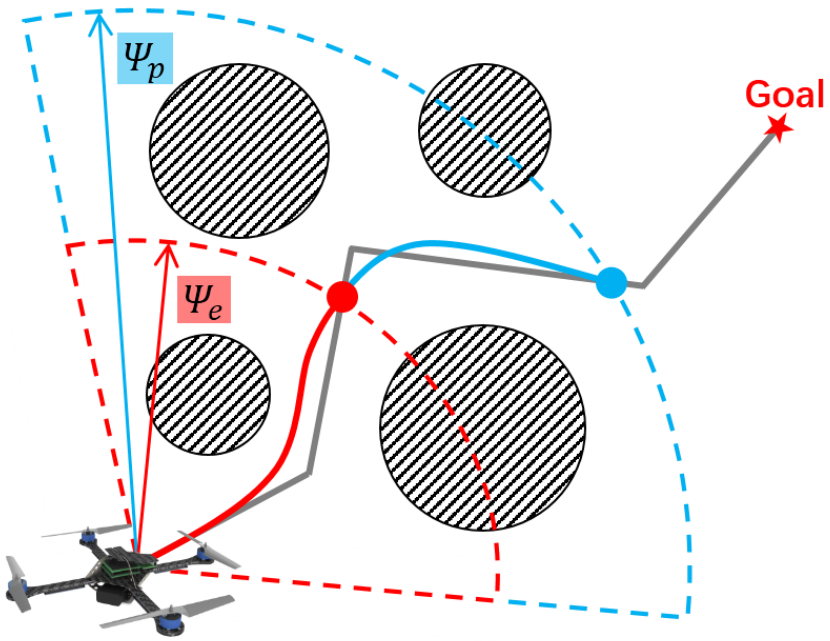
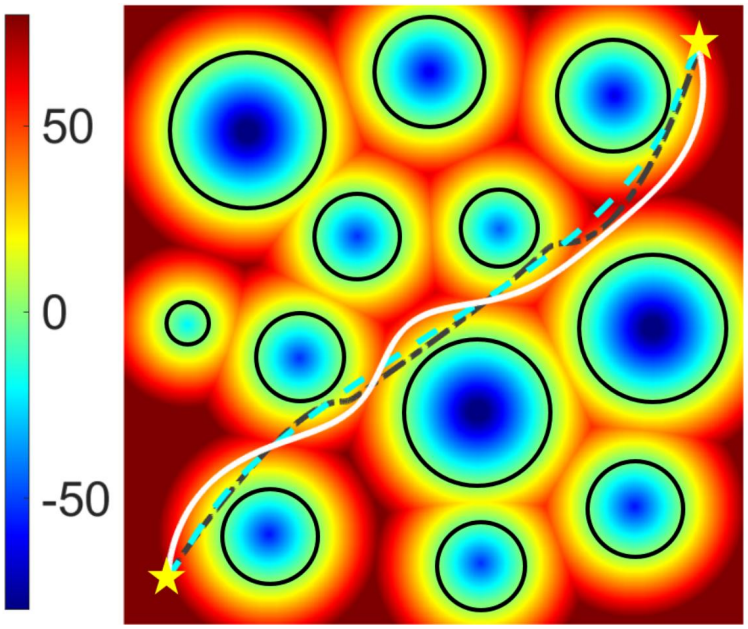


Figure 3.6. Illustration of the receding horizon incremental planning strategy. The global guiding path is shown with the gray line. The red curve and blue curve are the planning trajectory and the execution trajectory, respectively. See Sect. 3.3.3 for details.

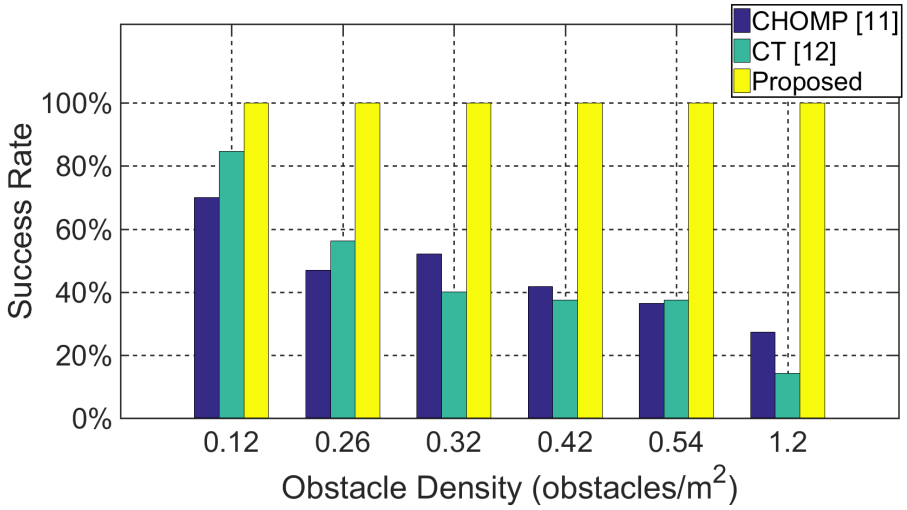
3.4 Results

3.4.1 Benchmark Results

We compare our proposed method with continuous-time (CT) trajectory generation [58] and CHOMP [70] using obstacles randomly deployed 2-D maps. For CHOMP, we set the parameters for balancing the computing time and success rate and set the number of points as 100 and the number of iterations as 200. For CT, we use the best parameters claimed in [58]. We set the number of segments of the trajectory as 3 and minimize the jerk of the trajectory. The results are given in Fig. 3.7. As the obstacle density increases, the success rate of CHOMP and CT both decrease rapidly, while at the same time, our method still generates feasible and safe trajectories.



(a) Trajectories generated by our method, CT and CHOMP.

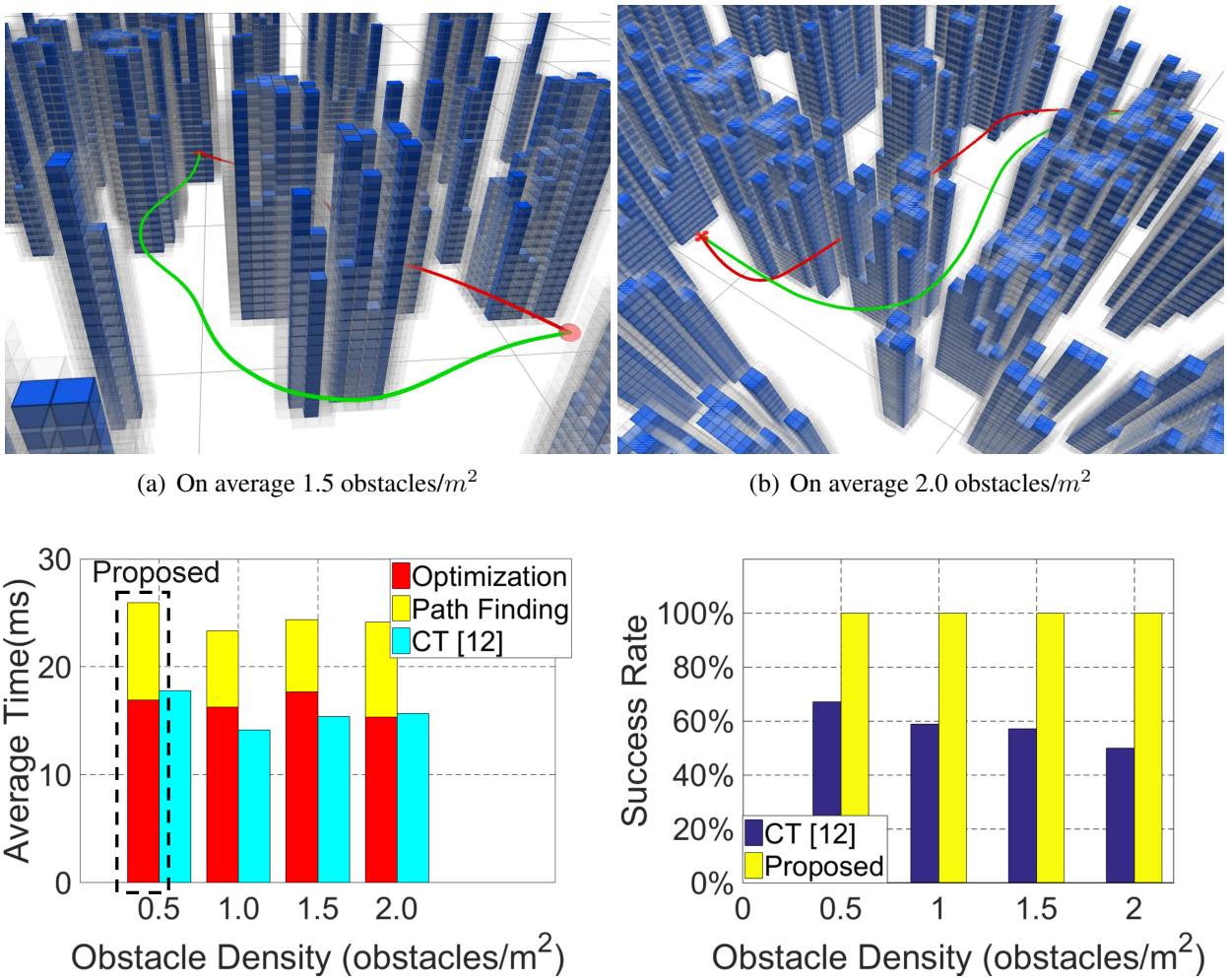


(b) Comparison of success rate. As the obstacle density increases, the success rate of CHOMP and CT both decrease rapidly.

Figure 3.7. Benchmark results. In Fig. 3.7(a), the white curve, brown dashed curve and blue dashed curve are trajectories generated by our method, CT and CHOMP, respectively. The color code indicates the distance value in the field. Black circles are obstacles randomly deployed.

For 3-D environments with randomly deployed obstacles, the map is $20\ m \times 20\ m \times 10\ m$ and the planning target is selected randomly. The results are shown in Fig. 3.8. The computing time for the optimization of our method is roughly equal to that of CT, while we obtain a 100% success rate with the cost of finding the initial safe path. Note that since we use our own implementation of

CHOMP and CT, the comparison may not be completely fair.



(a) On average 1.5 obstacles/m^2

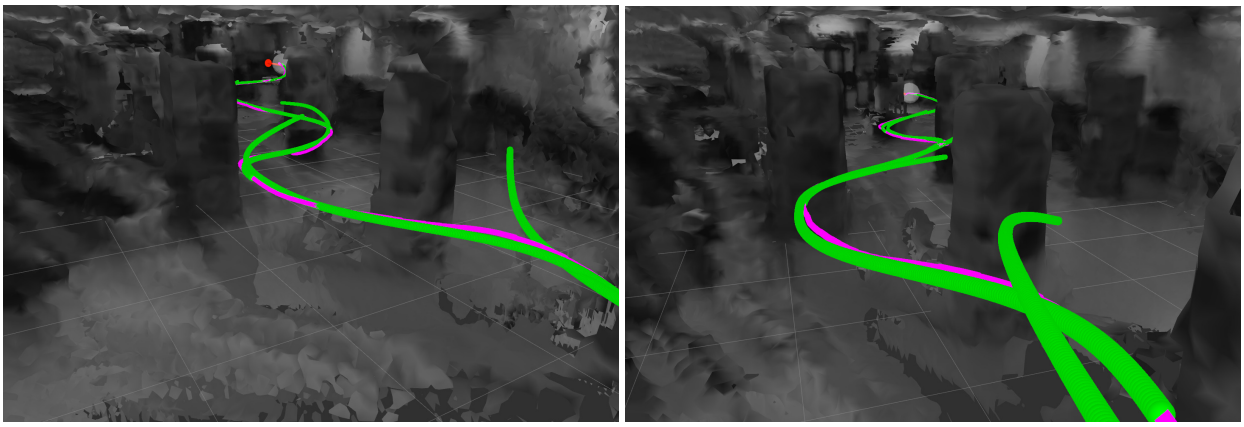
(b) On average 2.0 obstacles/m^2

(c) Comparison of the computing time and success rate

Figure 3.8. Comparison in random 3-D maps. The green curve indicates the trajectory generated by our method, and the red curve is the trajectory generated by continuous-time trajectory generation [58]. Comparing our method with CT, the time consumption of the optimization is roughly equal, but there is extra time consumed for path finding.

3.4.2 Indoor Autonomous Flight

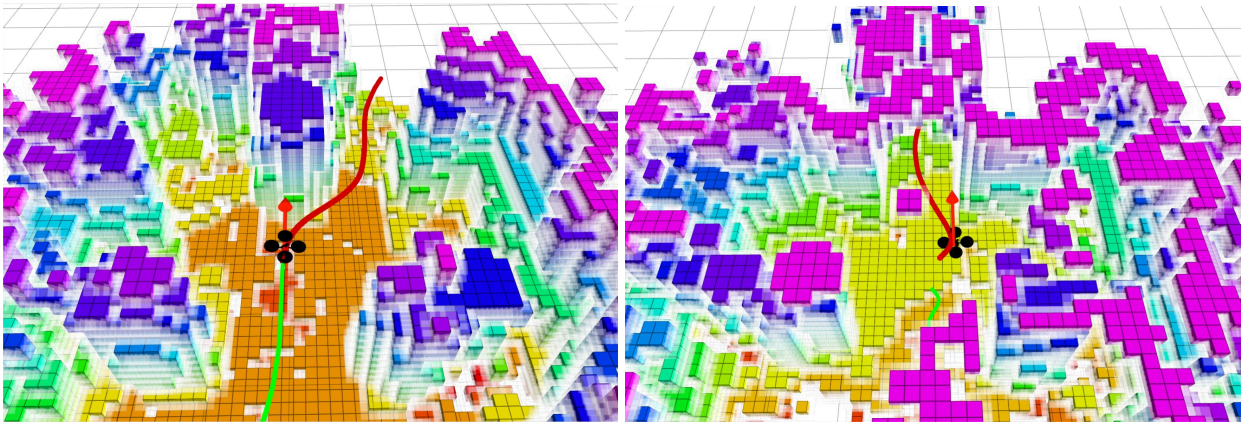
We present several autonomous flights in our laboratory with randomly placed obstacles to validate our proposed method. In the experiment, the quadrotor is commanded to fly to the targets. State estimation, mapping, and trajectory generation are all performed online without any prior information about the environments. The local planning horizon is set as $5\ m$, and the execution



(a) (b)

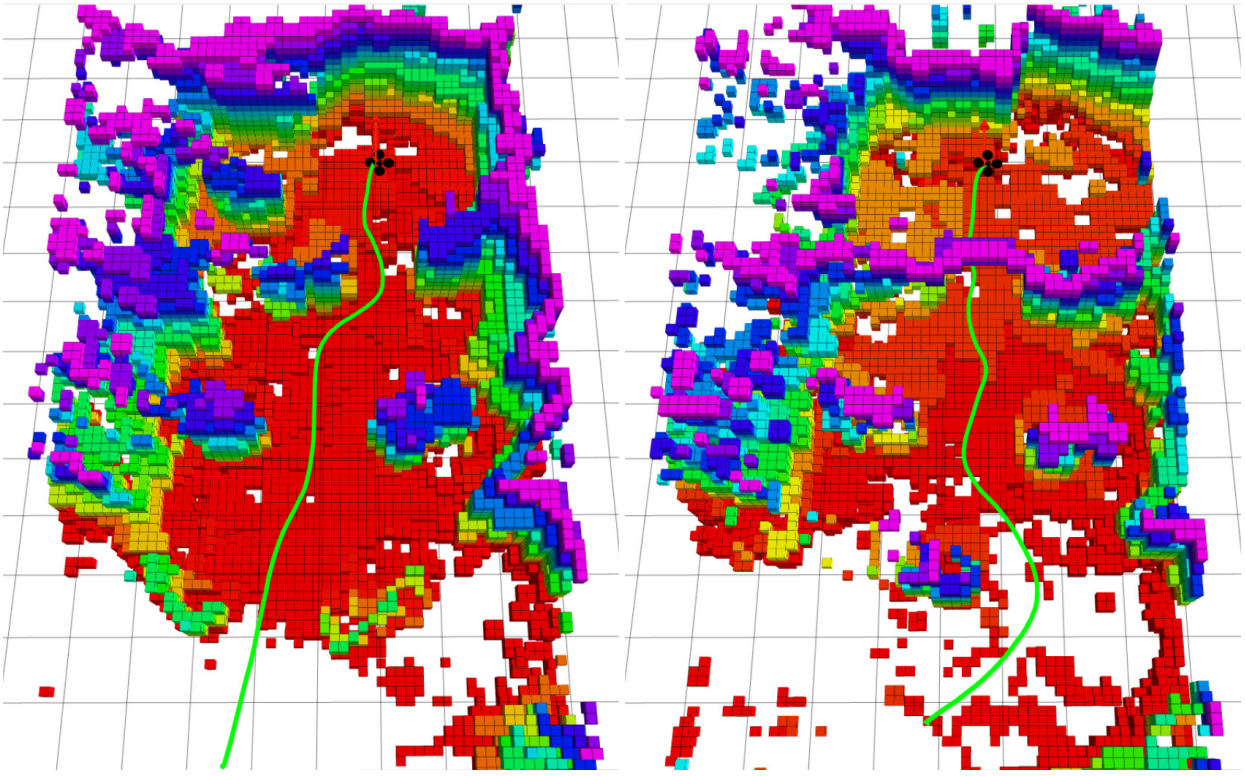
Figure 3.9. Three different demonstrations of the same dense indoor navigation experiment.

horizon is $2\,m$. Computing time limits for path finding and trajectory optimization are set as $15\,ms$ and $25\ ms$, respectively. In the mapping module, the size of the local map is $5\ m \times 5\ m \times 5\ m$, and the resolution of TSDF in mapping and local free voxel hashing in planning are both set as $0.15\ m^3$. Representative figures which record the indoor flights are shown in Figs. 3.9 and 3.10. The quadrotor generated safe and smooth trajectories and traveled through complex environments autonomously. In a real implementation, when observations are not sufficient, the unknown area may block the way for finding a path by the local planner, especially when there is no significant base-line for motion stereo. Under this situation, the quadrotor will generate safe but short trajectories ($< 1\ m$) in the free space, and explore the surrounding environments until the map is fully observed and a feasible path is found.



(a) Test 1: Passing through obstacles

(b) Test 2: Passing below an arch



(c) Complete map and path in test 1

(d) Complete map and path in test 2

Figure 3.10. Autonomous indoor flight in unknown environments. Two tests in different environments are given in Figs. 3.10(a) and 3.10(b). The colour code indicates the height of the obstacles. The white transparent area is the safe margin in the trajectory planning module. The red curve is the latest generated trajectory and the green curve is the trajectory that has been executed. The delay of the executed path in Fig. 3.10(b) is caused by an I/O jam onboard the quadrotor. The complete map and trajectory are shown in Figs. 3.10(c) and 3.10(d).

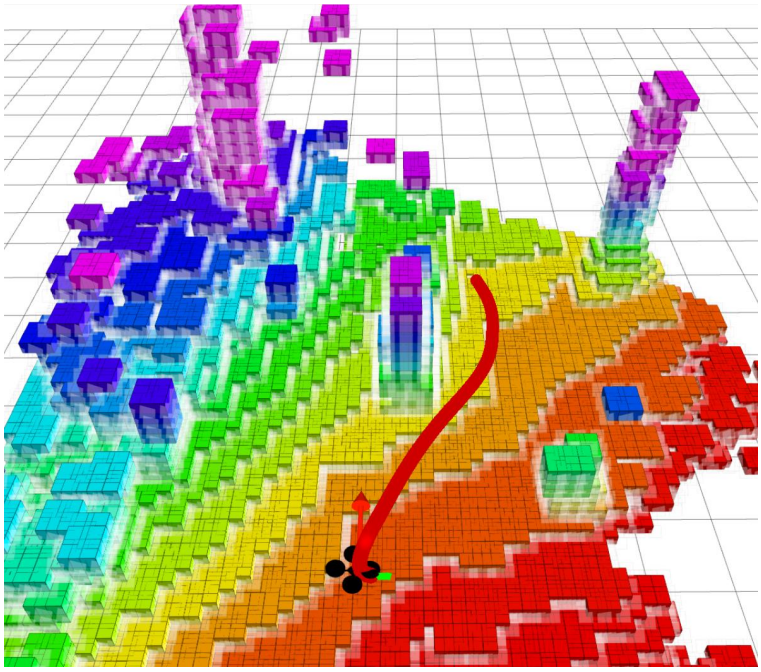
3.4.3 Outdoor Autonomous Flight

The outdoor experiments are done in two unknown complex environments; both contain a slope and obstacles in various shapes. The figure records the outdoor flight is given in Fig. 3.11. A complete mesh map recording the flight is given in Fig. 3.12. The parameters of the planning module are set as the same as in the indoor flights, and our quadrotor autonomously navigates in the cluttered environments. Note that although our planning module can run at a high frequency (25 Hz) since we adopt the planning and execution horizon, the planer would not be called every-time the map updated. This property saves computing resources significantly. We also employ the exploration strategy (Sect. 3.4.2) in the outdoor flights.

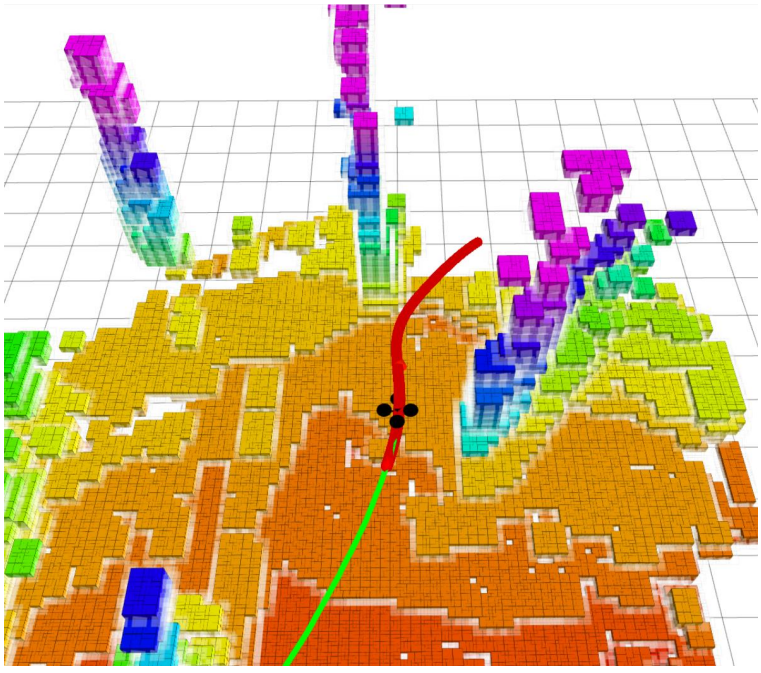
3.5 Conclusion and Future Work

In this chapter, we introduce a gradient-based motion planning framework for quadrotor autonomous navigation. The proposed method uses an optimization-based method to refine the trajectory and improve the dynamical feasibility. We integrate the planning, state estimation, and dense mapping module onboard our self-developed vision-based quadrotor platform and present fully autonomous flights in both indoor and outdoor unknown environments to validate our method.

For quadrotor motion planning, path finding and trajectory optimization work as front-end solution searching and back-end solution refinement, respectively. The method proposed in this chapter follows the soft-constrained principle. While some hard-constrained methods formulate the problem as convex programs with hard constrains, as stated in later chapters. Compared to hard-constrained methods, using unconstrained nonlinear optimization has the natural advantages of modeling costs for several different considerations and the ability to be terminated at any-time to meet the high-speed requirement and limited computing resources. We further extend the method proposed to significantly its convergence rate and robustness in [95], which is beyong the scope of this thesis and is therefore omitted here.

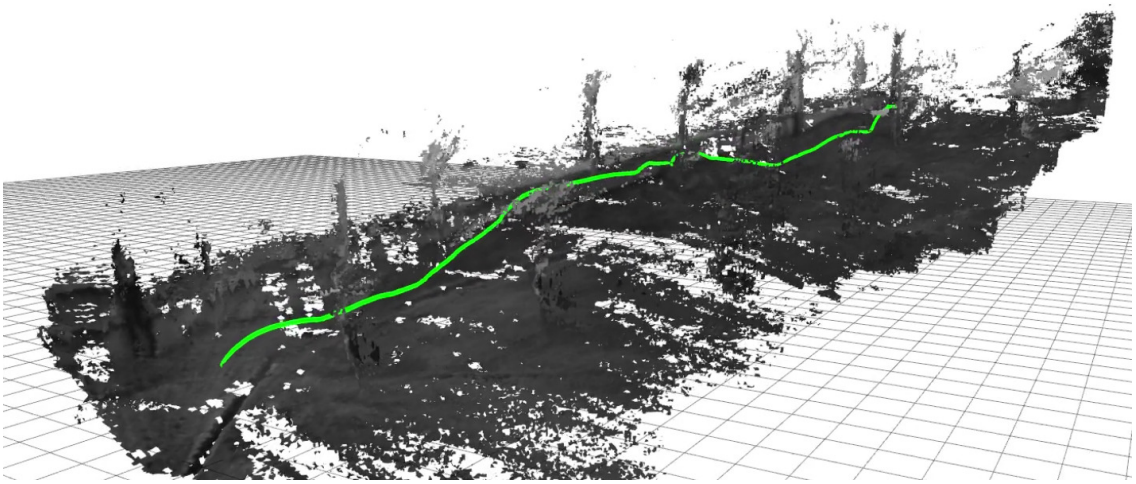


(a) Test 1: Pass through obstacles

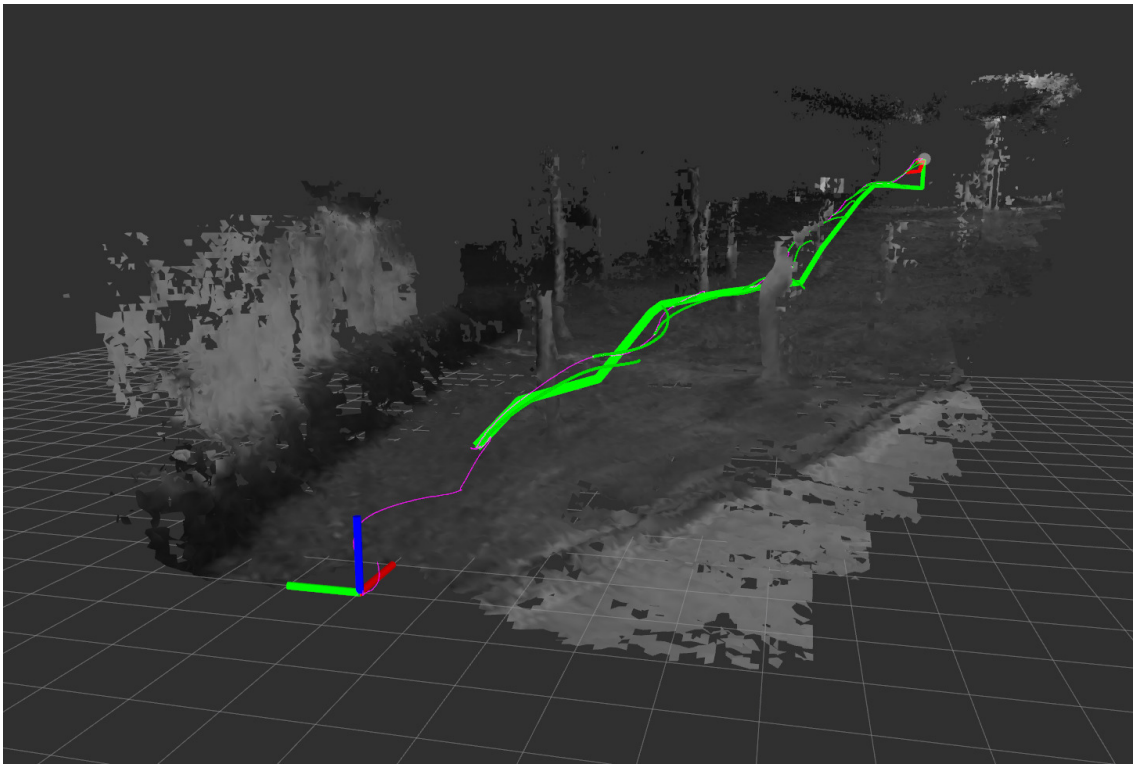


(b) Test 2: Pass below the arch

Figure 3.11. Autonomous outdoor flight in unknown environments. Two tests in different environments are shown in Figs. 3.11(a) and 3.11(b). Markers can be interpreted in the same way as Fig. 3.10. More outdoor trials of our proposed method are presented in the video.



(a) The overall flight trace of the autonomous drone



(b) Initial flight path and all local re-plans.

Figure 3.12. Complete map built and trajectory executed in outdoor flight. Note that the mesh map is generated offline for visualization since only the occupancy voxels are essential in assisting autonomous flight. The green curve is the complete trajectory which has been executed.

CHAPTER 4

HARD CONSTRAINED ONLINE SAFE TRAJECTORY GENERATION

4.1 Research Background and Motivation

In the last chapter, we introduce the gradient-based trajectory optimization techniques for generating smooth and collision-free trajectories, which have relatively high path clearance. The gradient-based method is also called the soft-constrained method since no constraints are mathematically enforced during the trajectory generation and are therefore not guaranteed to have a feasible solution. In the last chapter, although we use the initial safe path to improve the success rate in generating a collision-free final trajectory. It sacrifices the optimality in the final solution. That's because the generated trajectory is dominated by the initial path, and smoothness is not the only consideration in the objective function. In this chapter, we introduce the hard-constrained motion planning method for MAV and take our previous research [32] as its representative.

Under the category of hard-constrained methods, our previous work [29] and other works [11, 53] have been proposed to extract the free space in the configuration space and represent the free space by using a series of convex shapes. Then the trajectory generation problem is formulated as confining the trajectory within the convex shapes by convex optimization. However, two issues exist in these methods. The first issue is the time allocation for piecewise trajectories, a poorly chosen time allocation would easily produce a low-quality trajectory. And the second issue is how to efficiently bound the entire trajectory within the free space and its derivatives within the feasible space with hard constraints. In this paper, we use the fast marching method [75] in a velocity field based on Euclidean signed distance field (ESDF) to search a time-indexed path. The resulted time allocation is adaptive to obstacle densities in environments and achieves better trajectory quality compared to benchmark methods. To resolve the second issue, instead of using the traditional monomial polynomial basis, we use Bernstein basis to represent the trajectory as piecewise Bézier curves. In this way, the trajectory's safety and high-order dynamical feasibility are guaranteed through the optimization. We summarize our contributions in this chapter as:

1. A fast marching-based method applied on a Euclidean signed distance field (ESDF)-based velocity field, for searching a time-indexed path which provides reasonable time allocation for trajectory optimization.

2. A trajectory optimization framework, for generating trajectories that are guaranteed to be smooth, safe, and dynamically feasible by using Bernstein basis.

3. Real-time implementation of the proposed motion planning method and system integration in a complete quadrotor platform. Autonomous navigation flights in unknown indoor and outdoor complex environments are presented, and the source code will be released.

4.2 Fast Marching-Based Path Searching

4.2.1 Fast Marching Method

The Fast Marching (FM) method [75], as a special case of Level Set method [75], is a numerical method for simulating the propagation of a wavefront and is widely used in image processing, curve evolution, fluid simulation, and path planning. The fast marching method calculates the time when a wave first arrives at a point from the source by assuming the wavefront propagates in its normal direction with speed f. Suppose the propagation speed f>0, i.e., the wavefront only evolves outwards, and f is time-invariant and depends only on the position in the space, then the evolution of the wavefront is described by the Eikonal differential equation:

$$|\nabla T(x)| = \frac{1}{f(x)}, \tag{4.1}$$

where T is the function of the arrival time and x is the position, $f(x)$ depicts the velocity at different position x. An approximation solution to the Eq.4.1 in discrete space was introduced in [75], and we leave the details about the solution for brevity in this paper and refer readers to [75].

For path searching, we can define a velocity map for the robot to navigate. After simulating a wave expanded from the start point, the arrival time of each point in the map is obtained. By backtracking the path along the gradient descent direction of the arrival time from the target point to the start point, we obtain a path with minimal arrival time. That is the main idea of applying the fast marching method in path searching. Unlike other potential field-based methods, the fast marching

method has no local minimum and is complete and consistent as is proved in [44]. Theoretically, it has the same time complexity of $O(Nlog(N))$ as A*.

For traditional graph search methods used in discretized space, such as Dijkstra or A*, a discrete path is searched and then parametrized to time t. This may lead to sub-optimality of the back-end trajectory optimization since it's hard to parameterize the path to time reasonable given only the path length. However, the path searched by the fast marching method is naturally parametrized to t. Since the path is optimal on the metric of arrival time in the velocity field, not the distance, and each node on the path is associated with the first arrival time of a simulated wave propagated from the source point.

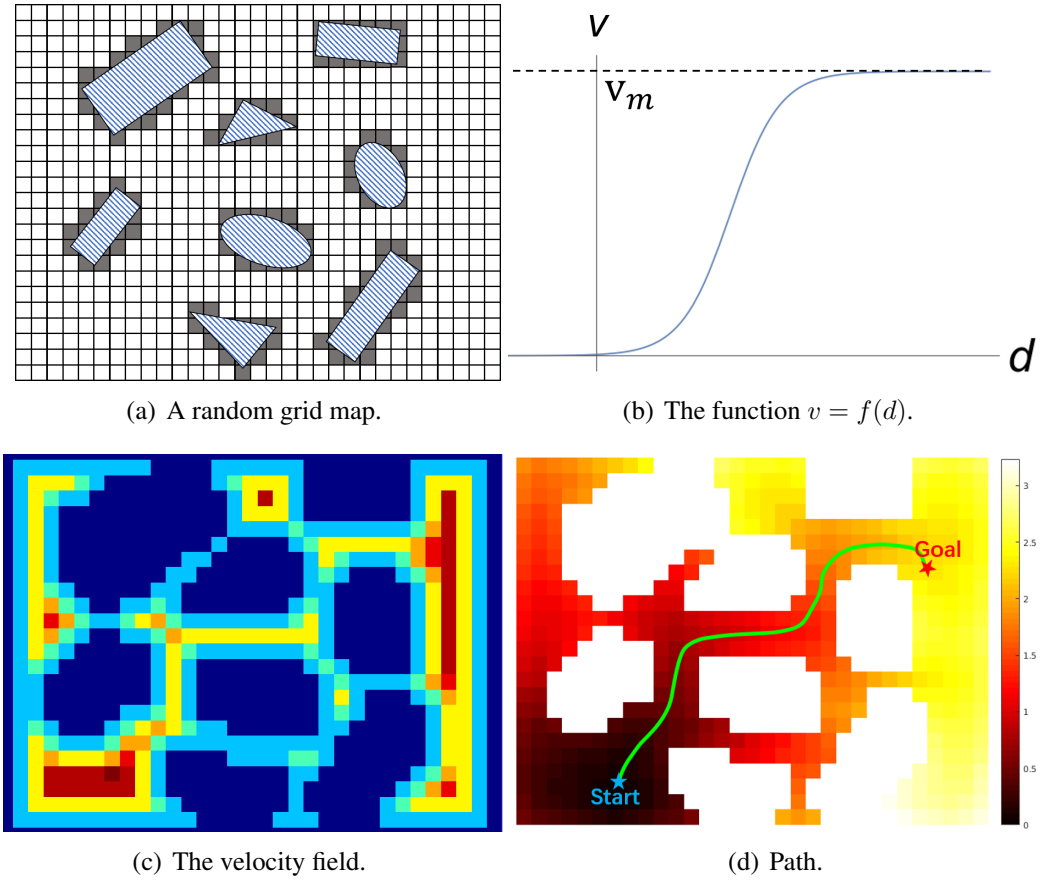
4.2.2 Fast Marching in Distance Field

For the purpose of finding a path with properly allocated times, we need to build a reasonable velocity field. We argue that it is natural for the quadrotor to reduce its speed in environments with dense obstacles since the quadrotor has to change its attitude more frequently to avoid collisions. Similarly, the quadrotor should be allowed to move fast in sparse environments. In other words, it's reasonable to assign the allowable velocity of the quadrotor according to the distance to obstacles. Therefore, Euclidean signed distance field (ESDF) is an ideal reference for designing the velocity map. In this paper, we use a shifted hyperbolic tangent function $tanh(x)$ as the velocity function:

$$f(d) = \begin{cases} v_m \cdot (tanh(d-e)+1)/2, & 0 \leq d \\ 0, & d < 0 \end{cases} \tag{4.2}$$

where d is the distance value at position x in the ESDF, e is Euler's number, v_m is the maximum velocity. The plot of the velocity function f is shown in Fig.4.1(b). This shifted hyperbolic tangent function grows slowly when the distance value is small, i.e. at where near to obstacles, and asymptotically approaches the pre-defined maximum velocity v_m. In occupied areas, the velocity is 0, means wavefront can't expand into these areas. An illustration of the velocity field and the path found by fast marching method is shown in Fig. 4.1.

In order to speed up the propagation of the wavefront, the fast marching method can be driven by heuristic function [64] to reduce the number of expanded nodes. In our implementation, since fast marching is applied on a velocity field, the admissible heuristic $h(x)$ can be estimated by



(a) A random grid map.

(b) The function $v = f(d)$.

(c) The velocity field.

(d) Path.

Figure 4.1. Path founded by fast marching method in a sampled 2-D map. In Fig. 4.1(a), grids in grey indicate being occupied by obstacels, while white grids are free. The velodity field of the map is obtained based on the euclidean distance field and Eq. 4.2, and is shown in 4.1(c). Dark blue indicates 0 velocity and dark red indicates maximum velocity. In Fig. 4.1(d), the minimum arrival time path is shown in green. Color shows the arrival time in each grid, where white indicates not expanded by the wavefront.

calculating the minimum time to arrival the target $h(x) = d^*(x)/v_m$, where $d^*(x)$ is the Euclidean distance from x to the target point.

4.2.3 Flight Corridor Generation

After obtaining the time-indexed minimal arrival path, we extract the free space in environments to form a flight corridor for the back-end optimization (Sec. 4.3). We would like to fully utilize the free space, since finding the solution space and obtaining the optimal solution are equally essential for trajectory generation. Therefore our approach is to initialize a flight corridor and enlarge it. For each node in the path, the distance from the node's position to the nearest obstacle is obtained easily by query the ESDF. In this way, at each node, we obtain a sphere of safe space against the

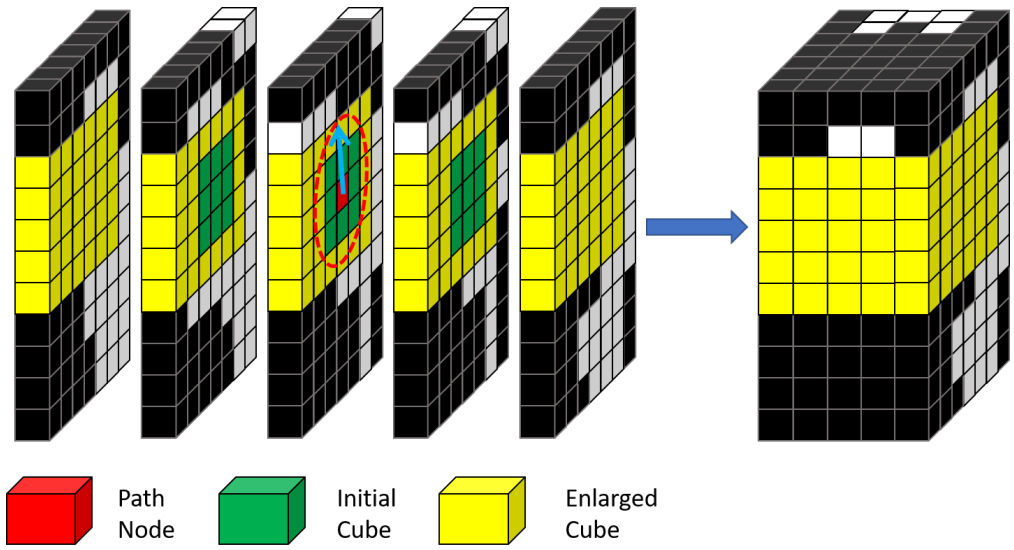
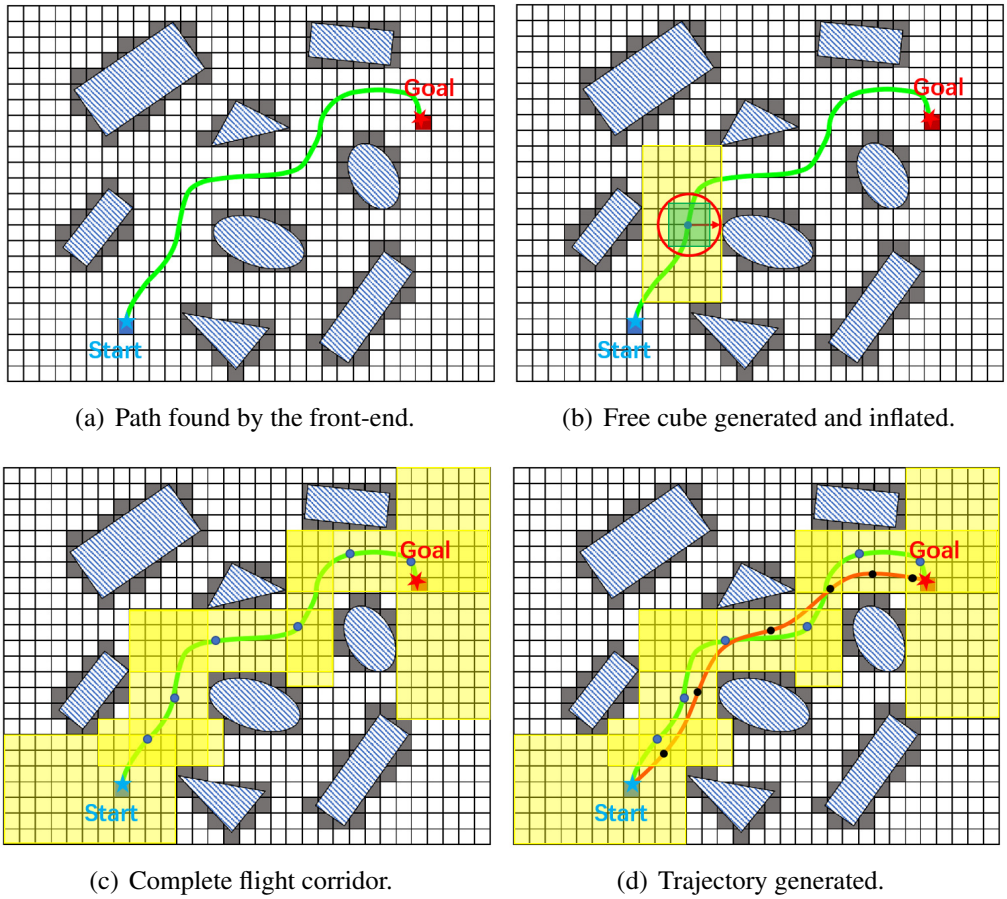


Figure 4.2. Illustration of the flight corridor generation and inflation in the 3-D occupancy grid map. The red voxel is the node on the path found by the fast marching method. The red dashed-line sphere and blue arrow indicate the nearest obstacle to this node. The initial free cube, which is shown in green, is generated within the sphere and is enlarged to its maximum volume against obstacles in axis-aligned directions. The inflated cube is shown in yellow.

nearest obstacle, and we initialize the flight corridor as the inscribed cube of the sphere. Then we enlarge each cube by querying neighbor grids on the axis-aligned direction x, y, z till the largest free volume it may have. The procedure is shown in Fig. 4.2. Since we initialize the flight corridor as a series of dense nodes, some nodes share the same enlarged cubes, which is unnecessary but increases the complexity of the back-end optimization. Therefore we prune all repeated cubes in the corridor. The pipeline of the path searching, initial corridor generation, and corridor inflation are illustrated in Fig. 4.3.

In this paper, the flight corridor inflation has a similar purpose as in [11]. However, we do not use the octo-map data structure which introduces extra maintenance cost, and we do it on a more general map structure voxel grids. Besides, compared to the method proposed in [53], we only inflate the flight corridor in axis-aligned directions. Thus we do not need to do the range query iteratively and save computational costs.



(a) Path found by the front-end.

(b) Free cube generated and inflated.

(c) Complete flight corridor.

(d) Trajectory generated.

Figure 4.3. Illustration of the trajectory generated in the corridor. The path found by the fast marching method is shown as the green curve. As in 4.3(b), for one node in the path, the free cube is initialized (in green) in the initial safe region (red circle) against the nearest obstacle and is inflated to its maximum axis-aligned volume (in yellow). After pruning redundant nodes, the flight corridor is formed and is marked in yellow in Fig. 4.3(c). Finally, the trajectory is optimized to be confined entirely within the flight corridor and satisfy dynamical feasibilities, as the red curve in Fig. 4.3(d).

4.3 Safe and Dynamically Feasible Trajectory Generation

The trajectory consisting of piecewise polynomials is parametrized to the time variable t in each dimension μ out of x, y, z. In this paper, instead of using traditional monomial polynomial basis, we use Bernstein polynomial basis similar to [23, 67] and represent the trajectory as piecewise Bézier curve, which is one special case of B-spline curve.

4.3.1 Bernstein Basis Piecewise Trajectory Formulation

The Bernstein polynomial basis is defined as:

$$b_n^i(t) = \binom{n}{i} \cdot t^i \cdot (1-t)^{n-i}, \tag{4.3}$$

where n is the degree of the basis and $\binom{n}{i}$ is the binomial coefficient. The polynomial consists of Bernstein basis is called a Bézier curve, and is written as:

$$B_j(t) = c_j^0 b_n^0(t) + c_j^1 b_n^1(t) + \ldots + c_j^n b_n^n(t) = \sum_{i=0}^{n} c_j^i b_n^i(t), \tag{4.4}$$

where $[c_j^0, c_j^1, ..., c_j^n]$ denoted as $\mathbf{c_j}$ is the set of control points of the j^{th} piece of the Bézier curve. Compared to monomial basis polynomial, Bézier curve has several special properties:

1. Endpoint interpolation property. The Bézier curve always start at the first control point, end at the last control point, and never pass any other control points.
2. Fixed interval property. The Bézier curve parametrized to variable t is defined on $t \in [0, 1]$.
3. Convex hull property. The Bézier curve $B(t)$ consists of a set of control points c_i are entirely confined within the convex hull defined by all these control points.
4. Hodograph property. The derivative curve $B^{(1)}(t)$ of a Bézier curve $B(t)$ is called as hodograph, and it is still Bézier curve with the controls points defined by $c_i^{(1)} = n \cdot (c_{i+1} - c_i)$, where n is the degree.

Actually, for a Bézier curve, the control points can be viewed as the weights of the basis and also the basis can be viewed as the weights of control points. Since Bézier curve is defined on a fixed interval $[0, 1]$, for each piece of the trajectory, we need a scale factor s to scale the parameter time t to an arbitrary allocated time for this segment. Accordingly, the m-segment piecewise Bernstein basis trajectory in one dimension μ out of *x, y, z* can be written as follows:

$$f_\mu(t) = \begin{cases} s_1 \cdot \sum_{i=0}^{n} c_{\mu 1}^i b_n^i(\frac{t-T_0}{s_1}), & t \in [T_0, T_1] \\ s_2 \cdot \sum_{i=0}^{n} c_{\mu 2}^i b_n^i(\frac{t-T_1}{s_2}), & t \in [T_1, T_2] \\ \vdots & \vdots \\ s_m \cdot \sum_{i=0}^{n} c_{\mu m}^i b_n^i(\frac{t-T_{m-1}}{s_m}), & t \in [T_{m-1}, T_m], \end{cases} \tag{4.5}$$

where c_{ji} is the i^{th} control point of the j^{th} segment of the trajectory. $T_1, T_2, ..., T_m$ are the end times of each segment. The total time is $T = T_m - T_0$. $s_1, s_2, ..., s_m$ are the scale factors applied on each piece of the Bézier curve, to scale up the time interval from $[0, 1]$ to the time $[T_{i-1}, T_i]$ allocated in one segment. Also, in practice, multipling a scale factor in position of each piece of the curve achieves better numerical stability for the optimization program.

The cost function is the integral of the square of the k^{th} derivative. In this paper we minimize the jerk along the trajectory, so k is 3. The objective is written as:

$$J = \sum_{\mu \in \{x,y,z\}} \int_0^T \left(\frac{d^k f_\mu(t)}{dt^k} \right)^2 dt, \tag{4.6}$$

which is indeed in a quadratic formulation $\mathbf{p}^T \mathbf{Q}_o \mathbf{p}$. $\mathbf{p}$ is a vector containing all control points c_{ij} in all three dimensions of *x, y, z*, and $\mathbf{Q}_o$ is the Hessian matrix of the objective function. Suppose in μ dimension the j^{th} segment of the trajectory is denoted as $f_{\mu j}(t)$, the non-scaled Bézier curve in interval $[0, 1]$ is $g_{\mu j}(t)$, as in Equ. 4.4. Denote $(t - T_j)/s_j = \tau$. Then the minimum jerk objective function in the μ dimension of the j^{th} segment is derived and written as follow:

$$\begin{aligned} J_{\mu j} &= \int_0^{s_j} \left(\frac{d^k f_{\mu j}(t)}{dt^k} \right)^2 dt = \int_0^1 s_j \left(\frac{s_j \cdot d^k (g_{\mu j}(\tau)}{d(s_j \cdot \tau)^k} \right)^2 d\tau \\ &= \frac{1}{s_j^{2k-3}} \cdot \int_0^1 \left(\frac{d^k g_{\mu j}(t)}{d\tau^k} \right)^2 d\tau, \end{aligned} \tag{4.7}$$

which only depends on degree n and scale s_j and can be written following the classical minimum snap formulation [55]. We leave the details of the $\mathbf{Q}_o$ for brevity.

4.3.2 Enforcing Constraints

For piecewise trajectory generation, we must enforce a number of constraints to ensure the smoothness, safety as well as the dynamical feasibility of the trajectory. For each piece of the Bézier curves, higher order derivatives of it can be represented by linear combinations of corresponding lower-order control points, which is written as

$$a_{\mu j}^{0,i} = c_{\mu j}^i, a_{\mu j}^{l,i} = \frac{n!}{(n-l)!} \cdot (a_{\mu j}^{l-1,i+1} - a_{\mu j}^{l-1,i}), \quad l \geq 1, \tag{4.8}$$

where l is the order of the derivative and n is the degree of the Bernstein basis.

Waypoints Constraints

Waypoints that the quadrotor needs to pass through, such as the start and target positons as well as the derivatives up to n^{th} order. Waypoints constraints are directly enforced by setting equality constraints on corresponding control points. For a fixed l^{th} ($l \leq n$) order derivative $d_{\mu j}^{(l)}$ of μ dimension exists at the beginning of the j^{th} piece of the trajectory (waypoints), we have:

$$a_{\mu j}^{l,0} \cdot s_j^{(1-l)} = d_{\mu j}^{(l)}, \tag{4.9}$$

Continuity Constraints

The trajectory must be continuous at all the ϕ^{th} derivatives at the connecting point between two pieces ($0 \leq \phi \leq k-1$). The continuity constraints are enforced by setting equality constraints between corresponding control points in two consecutive curves. For the j^{th} and $(j+1)^{th}$ pieces curves, we have:

$$a_{\mu j}^{\phi,n} \cdot s_j^{(1-\phi)} = a_{\mu,j+1}^{\phi,0} \cdot s_{j+1}^{(1-\phi)}, \quad a_{\mu j}^{0,i} = c_{\mu j}^{i}. \tag{4.10}$$

Safety Constraints

Thanks to the convex hull property of the Bézier curve as is illustrated in the property. 3, the entire trajectory is confined within the convex hull formed by all its control points. Thus we can add safety constraints to enforce all control points of one segment to be inside the corresponding cube generated in 4.2.3. For one segment of the trajectory, since the enlarged cube is a convex polygon, and all control points are inside it, then the convex hull of these control points is surely inside the cube, which means the entire segment of the trajectory is confined within the cube. The safety constraints are applied by adding boundary limit on control points, for control points of the j^{th} segment:

$$\beta_{\mu j}^{-} \leq c_{\mu j}^{i} \leq \beta_{\mu j}^{+}, \;\; \mu \in \{x, y, z\}, \;\; i = 0, 1, 2, ..., n, \tag{4.11}$$

where $\beta_{\mu j}^{+}$ and $\beta_{\mu j}^{-}$ are the upper and lower boundary of the corresponding j^{th} cube.

Dynamical Feasibility Constraints

Similarily to the safety constrians, utilizing the properties 3 and 4, we can enforce hard constraints on high order derivatives of the entire trajectory. Constraints are added to bound the derivatives of the curve in each dimenson μ. In this paper, the velocity and acceleration of the quadrotor are constrained in $[v_m^-, v_m^+]$ and $[a_m^-, a_m^+]$ to make the generated trajectory dynamically feasible. For control points of the j^{th} segment we have:

$$\begin{aligned} & v_m^- \leq n \cdot (c_{\mu j}^i - c_{\mu j}^{i-1}) \leq v_m^+, \\ & a_m^- \leq n \cdot (n-1) \cdot (c_{\mu j}^i - 2c_{\mu j}^{i-1} + c_{\mu j}^{i-2})/s_j \leq a_m^+. \end{aligned} \tag{4.12}$$

The waypoints constraints and the safety constraints are directly formulated as the feasible domain of the optimization variables in each segment ($\mathbf{c}_j \in \Omega_j$). The continuity constraints and high order dynamical constrains are reformulated as linear equality constraints ($\mathbf{A}_{eq}\mathbf{c} = \mathbf{b}_{eq}$) and linear in-equality constraints ($\mathbf{A}_{ie}\mathbf{c} \leq \mathbf{b}_{ie}$), here $\mathbf{c} = [\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_m]$. Then the trajectory generation problem is reformulated as:

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{Q}_o \mathbf{c} \\ \text{s.t.} \quad & \mathbf{A}_{eq}\mathbf{c} = \mathbf{b}_{eq}, \\ & \mathbf{A}_{ie}\mathbf{c} \leq \mathbf{b}_{ie}, \\ & \mathbf{c}_j \in \Omega_j, \quad j = 1, 2, ..., m, \end{aligned} \tag{4.13}$$

which is a convex quadratic program (QP), and can be solved in polynomial time by general off-the-shelf convex solvers.

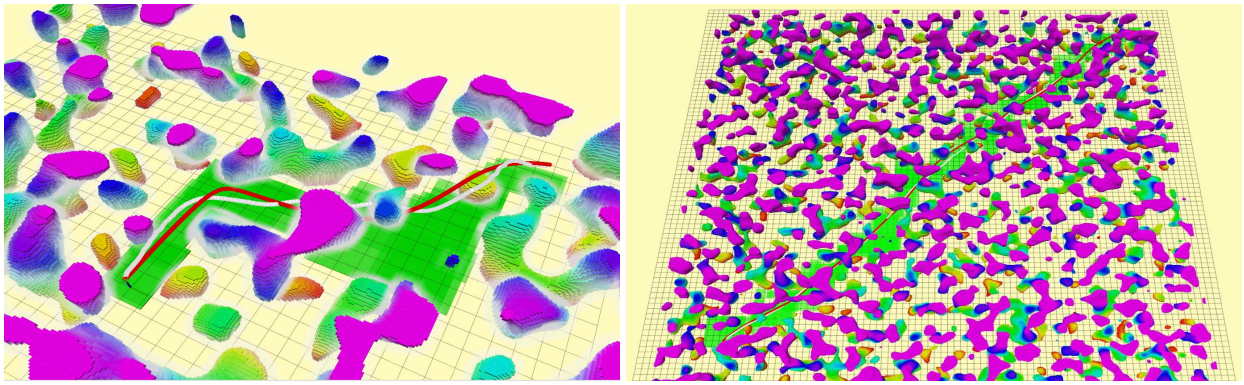
4.4 Results

4.4.1 System Settings

The planning method proposed in this paper[1] is implemented in C++11 using a general convex solver Mosek[2]. The quadrotor platform used in Chap. 3 is also adopted in this chapter for autonomous flight tests.

[1] Source code: `https://github.com/HKUST-Aerial-Robotics/Btraj`

[2] `https://www.mosek.com`



(a) Trajectory in a sparse environment. (b) Trajectory in a dense environment.

Figure 4.4. The time-indexed path searching and trajectory generation. The initial path searched by fast marching method is shown in white curve and the generated trajectory are in red. The flight corridor is in green and is projected into $x - y$ plane for visualization. The map is generated randomly by using Perlin noise.

4.4.2 Planning Strategy

For onboard usage, the building of the Euclidean distance field is usually the bottleneck of the entire motion planning pipeline. Therefore, in practice, we adopt a local buffer strategy to build the Euclidean signed distance field only in a local range, which equals the range of the mapping results, plus a buffer length. Another practical issue that has been revealed in our previous work [28] is when observations are not sufficient, the monocular fish-eye mapping module may output outliers that block the way for finding a feasible path. This issue is especially obvious at where far away from the camera and when there is no significant baseline for motion stereo. One way to resolve this issue is by using a double horizon planning strategy, as is shown in Fig. 4.5. We define an execution horizon and only check the collision status of the current trajectory within this horizon. This strategy makes sense since the camera always has more informative observations at where near it, and the mapping module is more confident at a shorter range. During the navigation, if any collision happens within this execution horizon, the re-plan mechanism will be triggered, and a new trajectory to the target will be generated.

4.4.3 Comparisons and Analysis

There exist other methods utilizing the free-space to formulate a convex flight corridor and generate trajectory within it. Method in [15] uses IRIS (iterative convex regional inflation by semidefinite programming) to obtain the maximum corridor and use combinatorial optimization to generate a

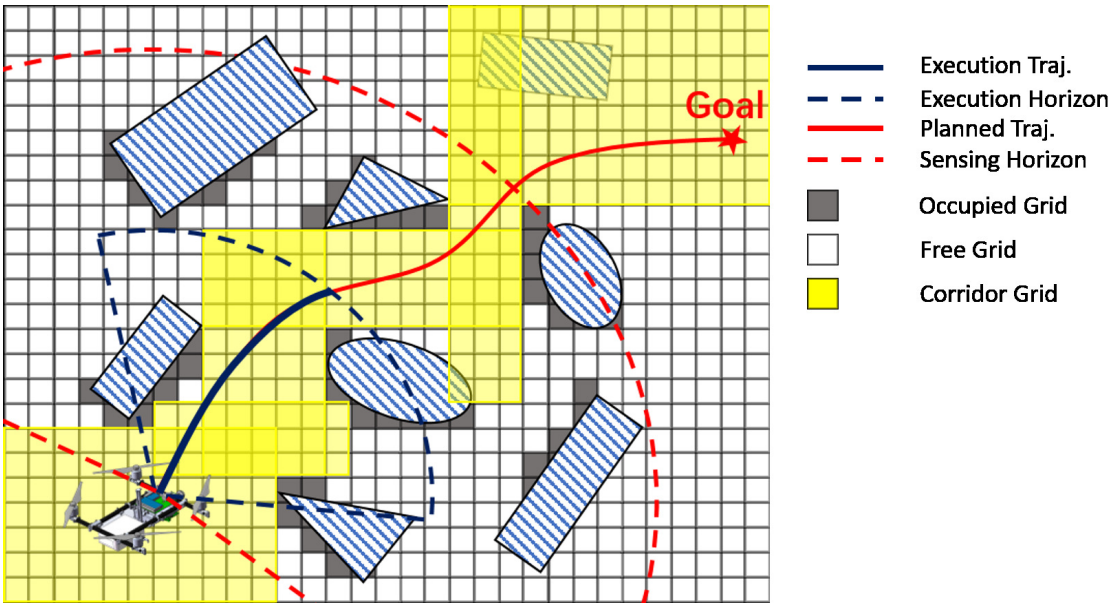


Figure 4.5. The planning strategy in 4.4.2. Grey grids are occupied grids detected by the onboard sensing, while white grids are free. The red dashed line, and black dashed line indicates the sensing range of our monocular fish-eye mapping module and the execution horizon, respectively. The planned trajectory is shown in a red curve, and the trajectory being executed is shown in black.

trajectory within several convex regions. However, it takes several seconds to generate a trajectory and is not for real-time onboard usage. Therefore we do not compare the proposed method with this method.

Chen's method [11], Liu's method [53] and the proposed method in this paper are all system-level methods for quadrotor motion planning. Compared to these two works, our method utilizes a fast marching method in a velocity field to provide a naturally time-indexed path, instead of searching a path and allocating time-based on some heuristic. Also, we utilize Bernstein polynomial basis to obtain safety and dynamical feasibility directly, which avoids the iterative optimization in Chen's procedure [11] and the collision risk in Liu's sample-based procedure [53]. Here we present a comparison between the proposed method with Chen's method in the aspect of trajectory quality (objective value), trajectory length, trajectory smoothness, and running time. One hundred trajectories are generated by randomly selecting a start and target positions at least $60m$ away in 10 random $40m \times 40m$ forests with obstacles number larger than 100 (as is shown in Fig. 4.6). Since the velocity of the trajectory has a great influence on the objective value, we set the average velocity equal for two methods. The maximum velocity is set as $2m/s$, and the acceleration limit is not set. As the results are shown in Table. 4.4.3, with roughly similar average velocity, the proposed

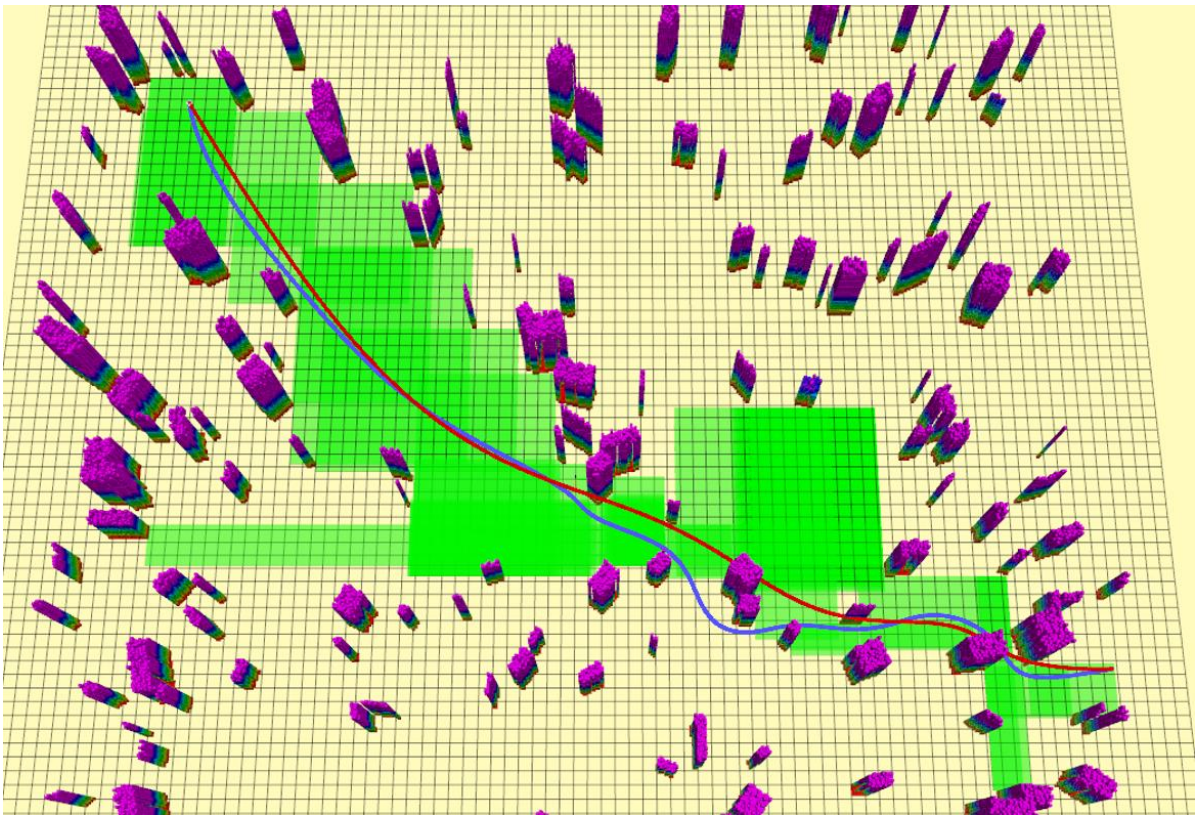


Figure 4.6. An instance of the comparison between the proposed method and Chen's method [11]. The map is $80m \times 80m$ and is generated with randomly deployed obstacles. The trajectory generated by the proposed method is shown in red, and Chen's method is shown in blue. The corridor in our method is projected into the $x - y$ plane for visualization.

method achieves lower objective cost, as well as the average and maximum jerk. Fast marching method tends to find a longer path then A* since the metric is velocity in the field, not the path length. This results in the proposed method generate longer trajectories than Chen's method, as is shown in the table. Besides, as the average velocity increases, the computational cost in Chen's method increases. This is because as the average velocity gets closer to the limit ($2m/s$), it is harder to confine it by iteratively adding constraints, more iterations are taken in this way.

4.4.4 Indoor Autonomous Flight

Indoor autonomous flights are done in previously unknown environments with randomly deployed obstacles to validate our proposed method. During the flights, the environment perception and trajectory generation are all performed online without any prior information about the environment. In the experiments, as shown in Fig. 4.7, the dense mapping module outputs a $5m \times 5m$ local dense

Table 4.1. Comparison of Trajectory Generation

Velocity (m/s)	Method	Objective Cost	Traj. Lenth (m)	Comp. Time (ms)	Average Vel.
0.65	Proposed	**0.0088**	82.8400	**100.1**	0.6427
	Chen's [11]	0.3058	**80.9628**	128.4	**0.6538**
1.0	Proposed	**0.0671**	86.3208	**106.5**	**1.0321**
	Chen's [11]	0.4508	**86.1697**	143.4	0.9764
1.5	Proposed	**0.8961**	85.3192	**101.4**	**1.4745**
	Chen's [11]	6.1166	**84.9211**	198.8	1.4028

perception of the environment, and the buffer length is set as $1.5m$. The resolution of our map is set at $0.2m$. The degree n of the Bézier curve is set as 10^{th}. We treat all unknown areas as collision-free areas. In each flight, a target position is sent to the quadrotor, and an initial trajectory to the target is generated. Re-plan mechanism is triggered based on the strategy illustrated in 4.4.2. Representative demonstrations of the indoor flights are shown in Fig. 4.7.

4.4.5 Outdoor Autonomous Flight

Figure 4.9. Autonomous flight in an unknown outdoor environment.

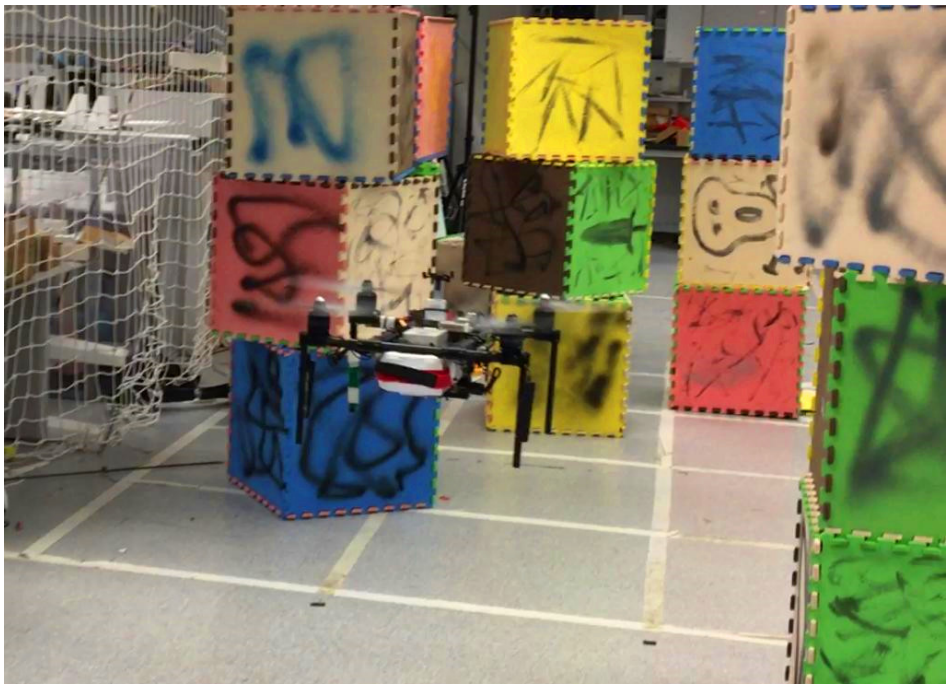
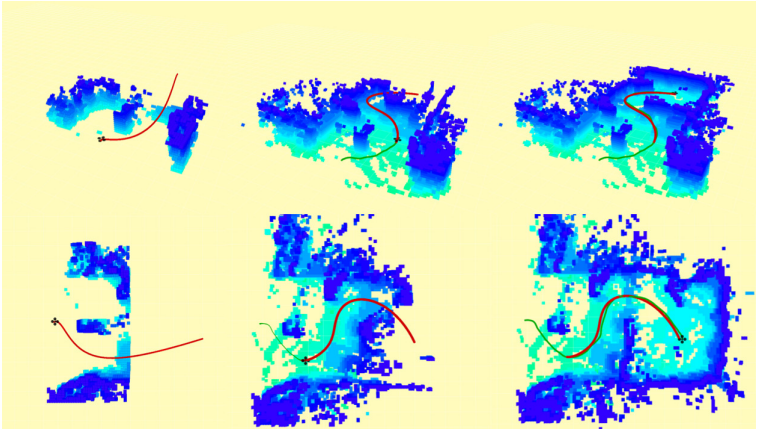


Figure 4.7. Autonomous flight in an unknown indoor environment.

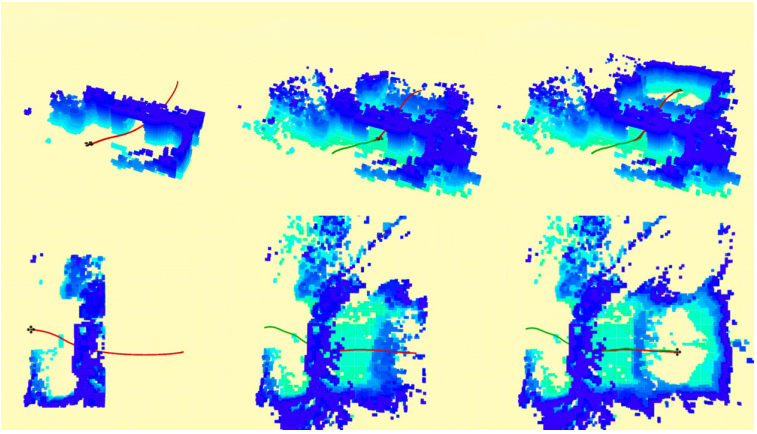
The outdoor experiments are done in unknown complex environments. Figures record several outdoor flights are given in Figs. 4.9 and 4.10. Complete map and trajectory of a flight are given in Fig. 4.10(c). The system configuration, parameter settings, and planning strategy are the same as in the indoor experiments 4.4.4. Average computing time for path searching, corridor generation and trajectory generation are 2.04 ms, 37.17 ms and 26.85 ms, respectively. Note that time cost in trajectory generation is dominated by the degree of the Bézier curves since the number of constraints scale linearly with the number of unknown variables. We refer readers to the video for more trials and details about the outdoor flights.

4.5 Dicussion

In this work, we propose a novel online motion planning framework for quadrotor autonomous navigation. The proposed method adopts a fast marching-based path searching method to find a time-indexed path in a velocity field adaptive to environments. A flight corridor is generated and inflated based on the path to utilize the free space in environments fully. Finally, we use an optimization-based method to generate safe and dynamically feasible trajectories with hard constraints by uti-



(a) Indoor test 1: passing through obstacles.

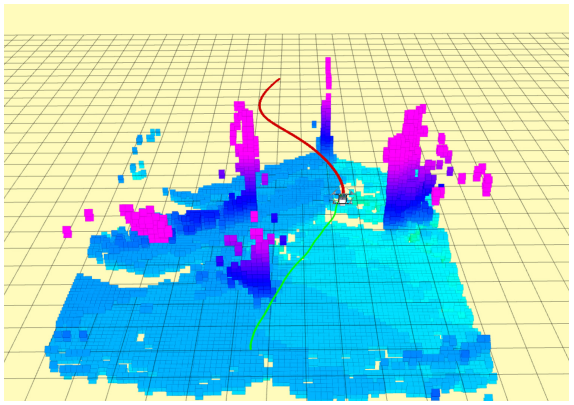


(b) Indoor test 2: passing beneath an arch.

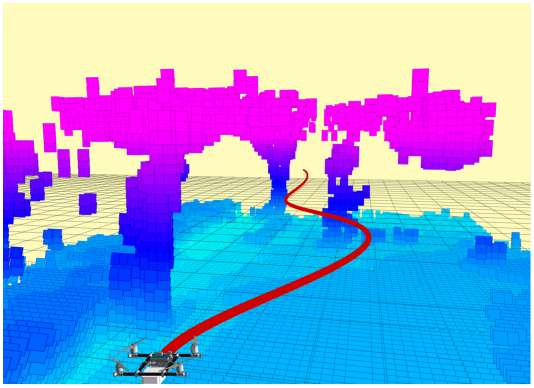
Figure 4.8. Autonomous flights in unknown indoor environments. Two tests in different environments are given in Figs. 4.8(a) and 4.8(b). The color code indicates the height of the obstacles. The red curve is the (re)generated trajectories, and the green curve is the path that the quadrotor tracked. The re-plan mechanism is triggered once new obstacles are detected, as is shown in the medium of the figures.

lizing Bernstein polynomial basis. We integrate the motion planning, state estimation, and dense mapping module into our self-developed quadrotor platform and present fully autonomous flights in both indoor and outdoor unknown environments to validate our method.

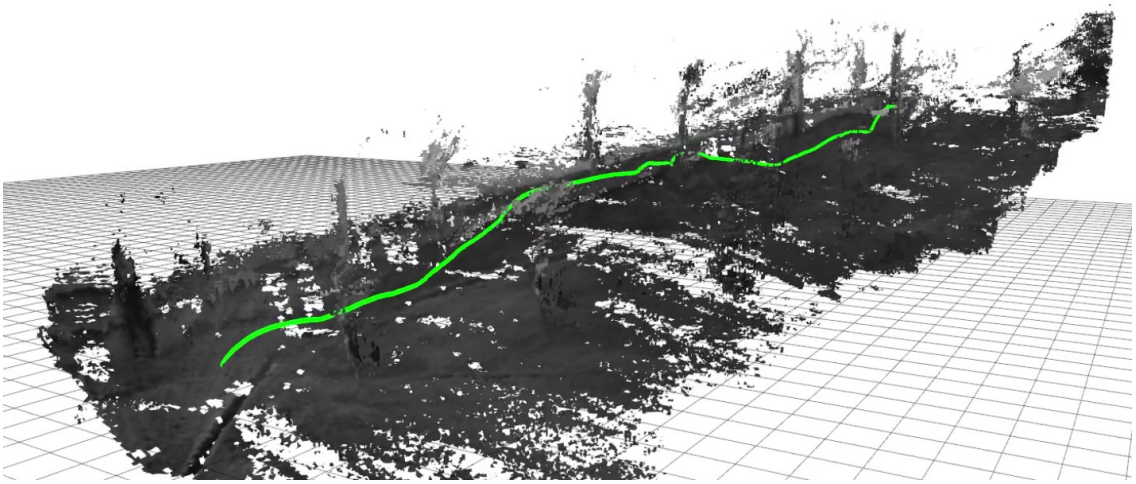
In this paper, the convex hull property we use is a conservative way to constrain the trajectory as well as its derivatives. This means the solution space of the trajectory is a subset of the flight corridor and asymptotically approaches the flight corridor infinitely close as the degree of the polynomial increases. In practice, it's a trade-off between the feasibility and the complexity of the problem. In the future, we plan to do more analysis and comparison between Bernstein and



(a) Trajectory generated in test 1



(b) Trajectory generated in test 2



(c) Overview of outdoor test 3

Figure 4.10. Autonomous flights in unknown outdoor environments. Marks can be interpreted the same as in before. Overall map and trajectory of a flight is shown in Fig. 4.10(c). Re-plan is triggered using strategy in 4.4.2 and only executed trajectories are shown for visualization.

monomial polynomial basis as well as the corresponding optimization methods. We also intend to challenge our quadrotor system in extreme situations such as large-scale or dynamic environments. Furthermore, we plan to extend our motion planning framework to quadrotor swarms.

CHAPTER 5

FLYING ON POINTCLOUDS: ONLINE MOTION PLANNING WITHOUT MAPS

5.1 Research Background and Motivation

In the previous two chapters, we introduce our research on the soft and hard constrained trajectory planning for MAV. The most fundamental pre-request for these methods is a map built by discretizing the surrounding environment of the robot. However, the map building process itself is time-consuming. Also, a discretized map introduces extra errors of the environments and therefore results in uncertainty for planning. In this chapter, we introduce our research on conductin MAV motion planning directly on point clouds without building a post-processed map.

In unknown cluttered environments, it is critical to online build dense maps for obstacle avoidance. Given the estimation of the vehicle state, many approaches have been proposed to convert depth measurements generated by onboard sensors into a global map. Representative methods include the voxel grid [19], Octomap [91], elevation map [12], etc. Each of these methods has pros and cons in particular environments. Voxel grids are suitable for fine-grained representation of small volumes, but the storage complexity scales poorly. Octomaps are memory-efficient when representing environments with large open spaces, at the expense of costly maintenance of the map structure. Meanwhile, elevation maps are suitable for representing human-made structures consisting of mostly vertical walls but are less efficient when describing natural and unstructured scenes.

In this chapter, instead of using the abovementioned post-processed maps, we opt to plan trajectories directly on point clouds' raw data. A point cloud, which is an underlying representation of the environment, doesn't need a discretization of the configuration space or complicated maintenance. It is essentially an unordered set of 3D raw data which can be obtained by many sensors, such as by camera in [92] and by LiDAR in [94]. The choice of planning directly on point clouds bypasses the costly map building and maintenance process and achieves the best adaptivity to different sensors in different environments, which is one of the critical requirements for search-and-rescue missions

in hazardous scenes. The concept of directly finding paths and generating feasible trajectories on point cloud data was first published in [29] and is significantly improved and extended later. We summarize the contributions of this method as follows:

1. An RRT*-based, anytime path finding method, named Safe-Region RRT*, for incrementally finding and refining a safe flight corridor directly on point clouds. The proposed method does not require an explicit map building nor computationally costly collision checking.

2. An optimization-based trajectory generation method, for generating safe and dynamically feasible trajectories within the flight corridor. This method utilizes properties of the Bernstein polynomial basis and formulates the trajectory generation problem as a second-order conic program (SOCP), which can be solved in polynomial time.

3. Integration of the proposed motion planning framework with state estimation, perception, and control modules into a fully autonomous quadrotor platform. Extensive field experiments and benchmarked comparisons are presented to validate the efficiency and robustness of the proposed method.

5.2 System Overview

5.2.1 Hardware Architecture

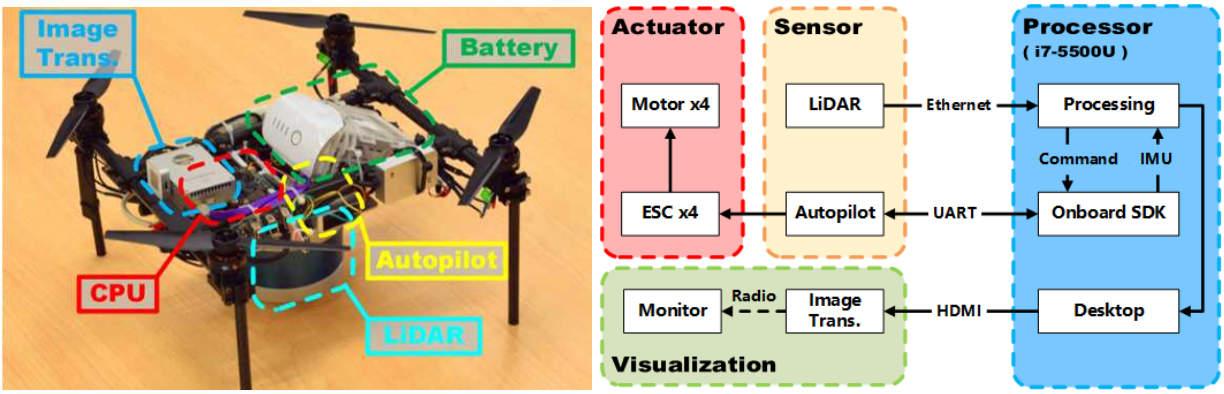
Compared to our previous platform in [29], in this paper, we develop a more compact and modular aerial robot (Fig. 5.1). We use a mini-computer that is powered by an Intel i7-5500U dual-core processor[1] running at up to 3.00 GHz and has an 8-GB memory and 64-GB SSD. All processing is done onboard on this mini-computer. Our quadrotor is equipped with a Velodyne VLP-16 Puck LITE[2] LiDAR for perception and a DJI A3 autopilot[3] for feeding inertial measurement unit (IMU) data and stabilizing the attitude. The autopilot communicates with the onboard computer by the intermediate layer, DJI onboard SDK[4]. We disable the GPS and magnetometer functionalities in

[1] https://ark.intel.com/products/85214/Intel-Core-i7-5500U-Processor-4M-Cache-up-to-3_00-GHz

[2] http://velodynelidar.com/vlp-16-lite.html

[3] http://www.dji.com/a3

[4] https://developer.dji.com/onboard-sdk



(a) The self-developed quadrotor platform. (b) The hardware architecture.

Figure 5.1. Our developed quadrotor platform. Some key devices include the onboard CPU, Velodyne LiDAR, DJI A3 autopilot, image transmission and intelligent battery are labeled in (a). The overall hardware and communication architecture is shown in (b).

the A3 due to their lack of reliability in cluttered indoor or even outdoor environments. We also remove the time-of-flight camera used in our previous platform to adapt to environments with uneven ground. For online visualization and monitoring, we use a DJI Lightbridge2 image transmission system to connect with the onboard computer. The quadrotor dynamic system consists of a DJI Snail motor with a 7-inch 7027s propeller[5] and DJI Takyon Z425-M electronic speed controller (ESC)[6]. The quadrotor is powered by the intelligent battery used in the DJI Phantom 4 Pro.[7]

5.2.2 Software Architecture

The complete software pipeline of our quadrotor system, from perception to planning and control, is shown in Fig. 5.2. Scan measurements from the 3-D LiDAR are used for the estimation of poses and the generation of globally-registered point clouds. In the perception layer, a generalized iterative closest point (ICP)-based method [93] is firstly used for frame-to-frame 6-Degrees of freedom (DOF) motion estimation. This incremental motion is then used as the initial guess for the following frame-to-map alignment to achieve global consistency of pose estimation. To provide high-rate and smooth state estimates for feedback control, we fuse the estimated pose with linear acceleration and angular velocity acquisition from the IMU using an extended Kalman filter (EKF). The

[5] https://www.dji.com/snail

[6] https://www.dji.com/takyon-z425-m-and-z415-m

[7] https://store.dji.com/product/phantom-4-pro-intelligent-battery-high-capacity

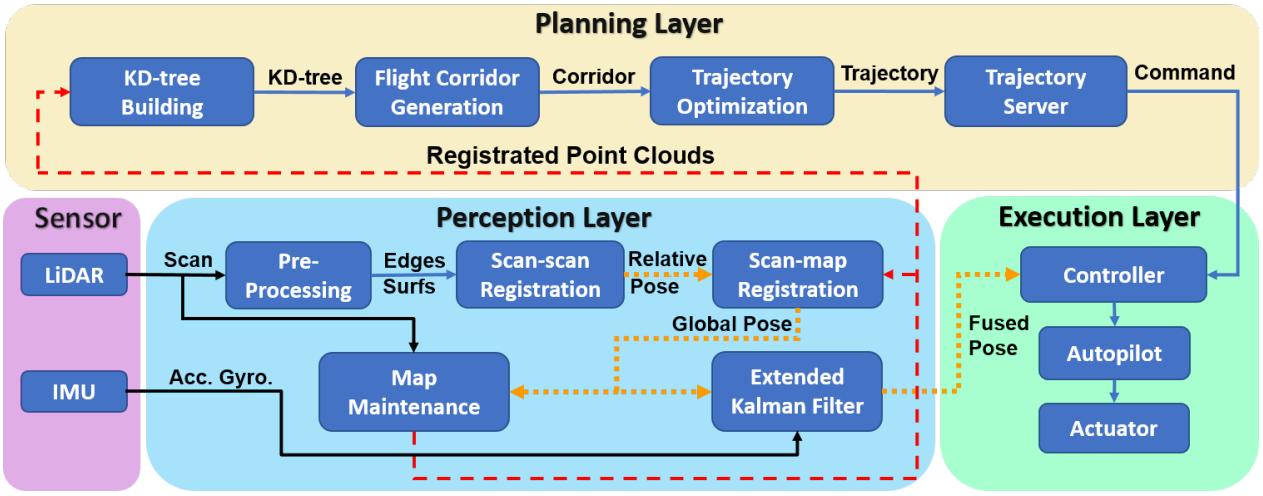


Figure 5.2. The software architecture (Sect. 5.2.2) of the quadrotor platform. In the perception layer, scan-scan registration and scan-map registration are performed to get the 6-DoF pose estimation and simultaneously generate registered point clouds. High-rate odometry is obtained by fusing the pose with IMU data. In the planning layer, the flight corridor is generated in the KD-tree of point clouds; then, a trajectory is optimized to stay safe and dynamically feasible. All processing is done onboard.

globally-registered point clouds are output for the planning layer. Based on the unordered point clouds, a sampling-based path finding method is proposed to generate a collision-free flight corridor using the efficient nearest neighbor search in the KD-tree. The corridor is a series of connected free spheres against surrounding obstacles. Using the flight corridor as geometric constraints and maximum velocity and acceleration as physical constraints, a collision-free and dynamically feasible piecewise Bézier curve that fits entirely within the flight corridor is generated. The generated trajectory is then sent to a trajectory server to calculate high-frequency desired states of the quadrotor. Finally, a feedback controller on SE(3) [47] is used for trajectory tracking, and the autopilot stabilizes the attitude of the quadrotor. The entire motion planning pipeline, from extracting a flight corridor to generating an optimal trajectory, is finished within several milliseconds in the onboard mini-computer.

5.2.3 Localization and Mapping

We now present our implementation of the laser-based state estimation and simultaneous mapping method. Our approach is adopted from [93], and it constitutes the perceptual foundation for a complete autonomous navigation system. The state estimation is initialized, and the origin of

the world-frame is fixed at the quadrotor take-off point. The 6-DOF relative motions between the LiDAR-frame and the world-frame is incrementally estimated, and the dense point clouds are registered at the same time.

The full pipeline of our laser-based state estimator and mapper is shown in our system architecture (Fig. 5.2). In the pre-processing sub-module, points are classified as edges and surfaces based on curvature and are extracted from each 360-degree 3-D range measurement. Then a generalized ICP scan matching algorithm is performed to do the relative pose estimation, by using the extracted planar and edge points in two scans. We use the same ICP algorithm for both scan-to-scan and scan-to-map alignment to obtain the global 6-DOF laser pose. The result of scan-to-scan alignment is used as the initial guess for the scan-to-map alignment, after which, points from the current scan are projected to the world-frame to build the global point cloud incrementally. Detailed explanations of the scan matching method can be found in [29] and [93]. Additionally, we mention two features in our maintenance of the globally registered point clouds:

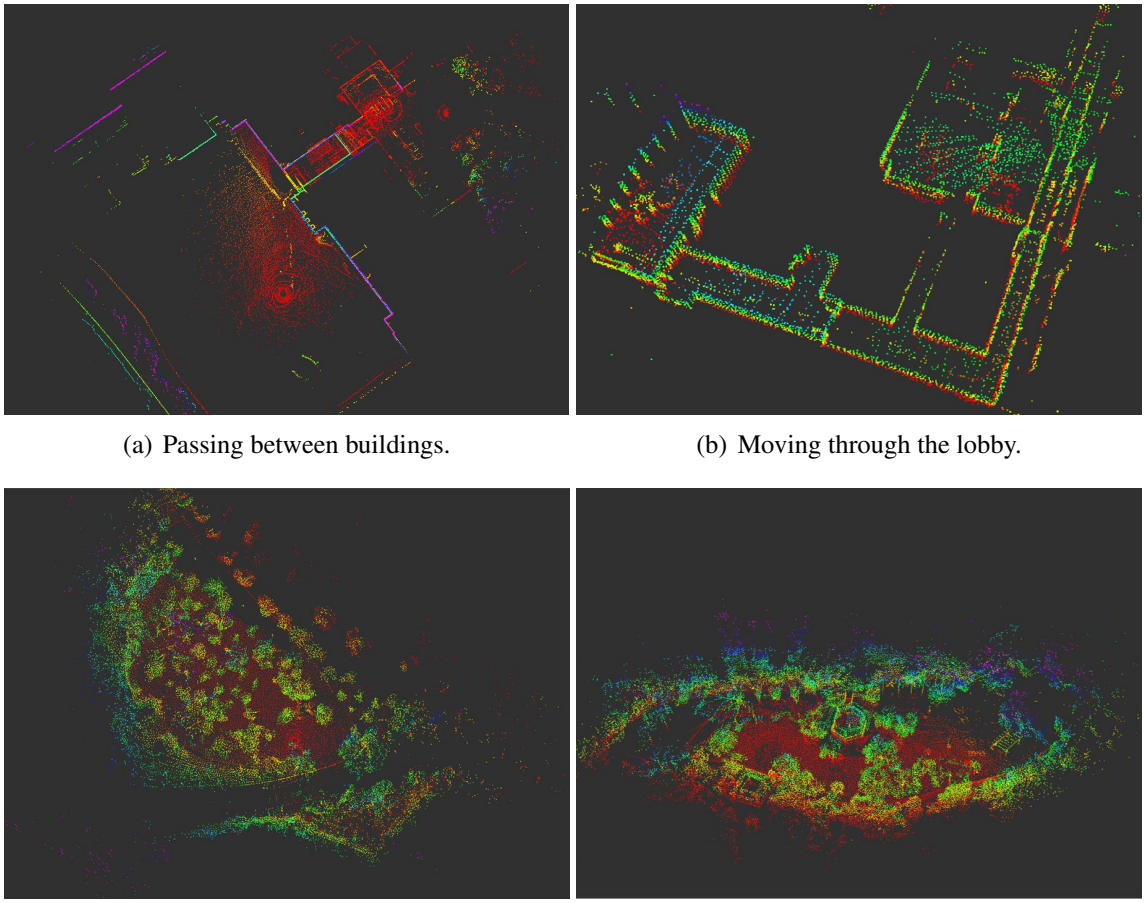
1. In order to bound the computational complexity for our path finding method (presented in Sect. 5.3), which operates directly on the point cloud raw data, we limit the spatial density of points in the point cloud. This strategy rejects redundant points that have very near coordinates.

2. For each point firstly added to the map, we set an initial weight (w). The weight is dynamically updated for points in each frame, to clear out moving objects and outlier measurements.

Based on the above two principles, we can define the update function for a point $\mathbf{x}_i^k$ in the i^{th} scan as follows:

$$w(x_{i+1}) = \begin{cases} \min(w(\mathbf{x}_i) - \alpha + \beta, \lambda_H), & \text{if } \exists\, \mathbf{x}_{i+1}, \quad s.t. \|\mathbf{x}_i - \mathbf{x}_{i+1}\| \leq \delta \\ \max(w(\mathbf{x}_i) - \alpha, \lambda_L), & \text{otherwise}, \end{cases} \tag{5.1}$$

where α and β are parameters that control the rate of decrease and increase of the weight, δ is a distance for determining whether the same point has been observed, and λ_H and λ_L are the upper and lower bounds of the weight, at which the point is considered as permanently added or removed from the map. Points that are repeatedly observed will eventually be fixed, while points that have received few observations, such as points on moving objects or outliers, will be removed after some



(a) Passing between buildings.

(b) Moving through the lobby.

(c) Mapping a dense forest.

(d) Circling around a pavilion.

Figure 5.3. Snapshots of point cloud maps built online. Color codes indicate the height of points. All data acquisition and processing are done onboard our quadrotor platform with an Intel i7 mini computer. More samples can be viewed in the experiments of this paper.

updates. The estimation and mapping module in our system is able to onboard run at more than $10Hz$ with a satisfactory density and accuracy.

To achieve the high-rate state estimation required for feedback control, we use a loosely coupled [54] extended Kalman filter (EKF) framework to fuse laser pose estimation with the IMU for velocity estimation, delay compensation, and a frame rate boost. In our sensor fusion framework, acceleration and angular velocity measurements from the IMU are used as the process model, while odometry from the LiDAR is used as the measurement model for the 6-DOF pose. The output of the EKF is directly used as the feedback in the closed-loop control. Some typical results of the LiDAR-based SLAM are given in Fig. 5.3.

5.3 Anytime Flight Corridor Generation on Point Clouds

5.3.1 KD-Tree-Based Point Cloud Representation

Our laser-based state estimation algorithm is performed utilizing the KD-tree [5] data structure for fast nearest neighboring points search. A balanced KD-tree with N points can be constructed in $O(NlogN)$ time with space complexity of $O(kN)$, and the M nearest neighbor search has a time complexity of $O(MlogN)$. For our laser-based state estimation and mapping, the KD-tree is essential for data association and point clouds registration. However, from the point of view of planning, the nearest neighbor search can also be used to find the distance from an arbitrary unoccupied location in the 3-D space to its nearest neighboring point (obstacle) on the map. This distance indicates the safe radius with respect to that location, from which a safe region in the shape of a sphere can be generated. Based on this basic idea, we propose a sampling-based path finding method to carve free space in environments for quadrotor navigations. Note that the KD-tree is a static data structure, which means dynamically inserting points into a KD-tree will make it unbalanced and inefficient. Therefore, every time the map is updated, we reconstruct the KD-tree. In our experiments, the time required for reconstructing a KD-tree of a $100m \times 100m \times 5m$ map is on average $8 \sim 10ms$ for our onboard computer, which is far less than the mapping and replanning frequency ($10Hz$). Thus, it is acceptable for real-time applications. Also, one can directly copy the KD-tree built for frame-to-map registration in the perception layer, for planning usage. As for planning missions on a vast scale, we limit the volume to which the point clouds can spread. This strategy results in a local map that slides with the quadrotor. Combined with the limit on the spatial density of points, as stated in Sect. 5.2.3, the number of points in the point cloud is, therefore, upper bounded.

5.3.2 Sampling-Based Random Safe-Region Tree Generation

As presented in Sect. 5.3.1, the safe radius of an arbitrary point in the map can be found by a fast nearest neighbor search in the KD-tree. And a sphere-shaped safe region is defined at this point. Based on this property, we develop an RRT*-based method, named safe-region RRT*, in this paper to generate corridors connecting the starting point to the target point. To begin, we follow the core idea of RRT* to construct a randomly sampled tree with vertices being the centers of sphere-shaped safe regions, and edges being the connectivity between such safe regions. As new

samples are generated, the tree is expanded and re-wired. The flight corridor then consists of a sequence of connected sphere-shaped safe regions which are determined by the proposed method. To improve the efficiency of the algorithm, we incorporate the heuristic-sampling strategy [27] in our random tree exploring procedure. After a feasible path connecting the start node to the target node is obtained, we use the path cost as a heuristic to prune the tree and reject further samples that cannot improve the path quality.

During the navigation of our quadrotor, when new measurements are obtained, and unmapped areas are discovered, some nodes in the tree may collide with new obstacles and become infeasible. In this paper, we do not destroy the tree and re-initialize it as in our previous work [29]. Instead, samples are added to the tree to approach the asymptotic optimality of the path, and infeasible nodes are online repaired in each replanning loop. In fact, the resulting random tree that consists of sphere-shaped safe regions can also be viewed as a topological map, which indicates the free space distribution from the start position to the target position in an environment.

Random Tree Expansion

The expansion of the random tree is the basis of our proposed algorithm and is shown in Algo. 1. Suppose we have the point cloud raw data $\mathcal{P}$ and the original solution space of planning $\mathcal{M}$. The random exploring tree $\mathcal{T}$ is initialized with the root node n_s located at the start position of the quadrotor. And the sampling domain Ω is initialized as M. A node n in the tree $\mathcal{T}$ represents a safe region in the free space with several properties, the 3D location $n.\mathbf{c}$ of its center, the length $n.r$ of its radius and the total path cost $n.f$ to the root node. After a new random coordinate $\mathbf{c}_r$ is sampled by function **Sample**(Ω) in Ω, **Nearest**$(\mathbf{c}_r, \mathcal{T})$ finds its nearest node n_n in the tree $\mathcal{T}$. Our algorithm ensures that in the generation of the tree, edges connecting vertices (sphere-shaped safe regions) are collision-free, by designing the **Steer**$(\mathbf{c}_r, n_n)$ function. As is shown in Algo. 2, the **Intersect**(c_r, n_n) function generates a ray from $\mathbf{c}_r$ to $n_n.\mathbf{c}$, and returns the intersection point of the ray with n_n's sphere. We set this coordinate as the center of the new node $n_r.\mathbf{c}$. We then perform a radius search **RadiusSearch**$(n_r.\mathbf{c}, \mathcal{P})$ to find the radius of the largest sphere centered at $n_r.\mathbf{c}$ against the point cloud raw data $\mathcal{P}$ and set it as $n_r.r$. The new node n_r is checked by **VolumeCheck**(n_r) to verify that it has sufficient volume for a quadrotor to travel through. If n_r qualifies, we search all its nearby connected nodes $\{n_{near}\}$ in $\mathcal{T}$. Then the parent node of n_r is chosen by **ParentChoose**$(n_r, n_n, \{n_{near}\})$ and the tree is locally rewired by **Rewire**$(n_r, \{n_{near}\}, \mathcal{T})$ in

the same way as the RRT* algorithm [39]. In our algorithm, when a sample is drawn, we create a new node only centered on the nearest node's sphere. Therefore the connection between every two nodes is naturally collision-free, and no collision checking is needed. Also, since the safe volume decides the steer distance at a sample point, the expansion rate of the tree varies according to the density of obstacles, making it particularly efficient for environments with large volumes of free space.

We also utilize the informed sampling scheme, which is introduced in [27] to improve the convergence rate towards optimality. If the new node contains the target within its safe region, as justified in the function **GoalContain**(n_r), and has a lower total path cost than the current best solution f_{best}, we update the sampling domain Ω to a hyper-ellipsoid subset of the space for further sampling. The function **Update**(f_{best}, Ω) updates the hyper-ellipsoid heuristic domain, with the root and target on its focal points. The transverse diameter of the hyper-ellipsoid is the distance of the best path found so far, and the conjugate diameter is the direct straight distance between the start and target coordinates, as shown in Fig. 5.4. At the end of each iteration, the time consumption is checked (**Time**()). The tree continuously expands until the maximum allowed time t_{max} has expired. The shortest path consisting of nodes connecting the start point to the target point, is extracted as the flight corridor $\mathcal{C}$.

Algorithm 1 Safe-region RRT* Expansion

Require $\mathcal{P}, \mathcal{M}$
$\mathcal{T} \leftarrow n_s, t \leftarrow 0, \Omega \leftarrow \mathcal{M}$
while $t \leq t_{max}$ **do**
  $\mathbf{c}_r \leftarrow$ **Sample**(Ω)
  $n_n \leftarrow$ **Nearest**$(\mathbf{c}_r, \mathcal{T})$
  $n_r \leftarrow$ **Steer**$(\mathbf{c}_r, n_n)$
  if VolumeCheck(n_r) **then**
    $\{n_{near}\} \leftarrow$ **Near**$(n_r, \mathcal{T})$
    $n_p \leftarrow$ **ParentChoose**$(n_r, n_n, \{n_{near}\})$
    $\mathcal{T} \leftarrow n_r \cup \mathcal{T}$
    $\mathcal{T} \leftarrow$ **Rewire**$(n_r, \{n_{near}\}, \mathcal{T})$
    if GoalContain(n_r) AND $n_r.f < f_{best}$ **then**
      $f_{best} \leftarrow n_r.f$
      $\Omega \leftarrow$ **Update**(f_{best}, Ω)
    end if
  end if
  $t \leftarrow$ **Time**()
end while
$\mathcal{C} \leftarrow \min($**PathExtract**$(\mathcal{T}))$
return $\mathcal{C}$

Algorithm 2 $n_r \leftarrow$ **Steer**$(n_n, \mathbf{c}_r)$

Require $\mathcal{P}$
$\mathbf{c} \leftarrow$ **Intersect**$(\mathbf{c}_r, n_n)$
$r \leftarrow$ **RadiusSearch**$(n_r.\mathbf{c}, \mathcal{P})$
$n_r.\mathbf{c} \leftarrow \mathbf{c}$
$n_r.r \leftarrow r$
return n_r

Trajectory Commitment and Branch Pruning

After the initial flight corridor is obtained, and the trajectory within it is generated (to be discussed in Sect. 5.4), the quadrotor starts to track the trajectory towards the target. Since the initialization of our safe region RRT* terminates in a very limited time, the generated flight corridor may have poor quality. Therefore, we would like to utilize the execution time of the quadrotor to continuously refine the corridor online, which finally leads to a much shorter trajectory. Also, path consistency is an essential issue when we incrementally add samples and refine the flight corridor. In this paper, we follow the principle of anytime RRT* [40] to commit (fix) part of the generated trajectory and focus on improving the rest of it. In this way, within an execution horizon, the flight corridor, as well as the trajectory, is consistent, as is shown in Fig 5.5. We denote the duration of the trajectory to be committed as t_m. The essential requirement for choosing t_m is that the committed trajectory must be inside the sensing horizon of the quadrotor. Since this part of the trajectory will be executed without future modifications, its feasibility must be guaranteed by up-to-now observations of onboard sensors. In our motion planning framework, after an initial flight corridor is found, the trajectory is generated and sent to the quadrotor to track. Then the trajectory in $[0, t_m]$, which is also called the committed trajectory, is fixed. We set a new root node of the tree as the node which contains the end coordinate of the committed trajectory. Moreover, we cut all other branches not connected to the new root. The sampling region Ω is updated accordingly. During the execution of the committed trajectory, the random tree is continuously expanded and -re-wired to improve the quality of the flight corridor. When the quadrotor arrives at the end of the committed trajectory, a new trajectory is generated based on the current best flight corridor maintained in the tree, and a new committed trajectory is fixed. The procedure is done iteratively until the quadrotor arrives at the global navigation target.

To focus on promising nodes that may lead to a better solution, we utilize the branch-and-bound mechanism to help efficiently expand and maintain the tree. Similar to the heuristic function used

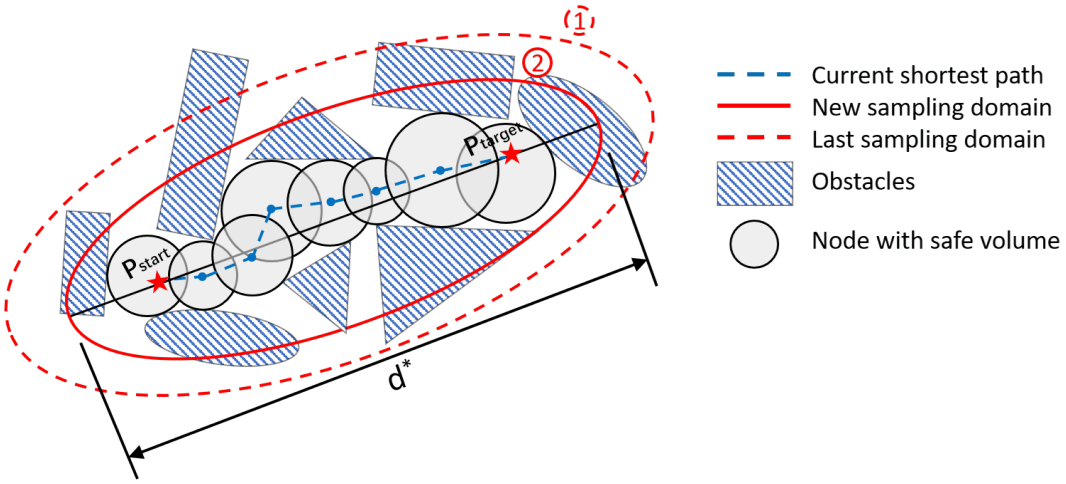


Figure 5.4. Heuristic sampling domain update. Red dashed ellipsoid with marker 1 is the last hyper-ellipsoid domain for generating samples. After a new better solution d^* has been found, the sampling domain shrinks to the red solid ellipsoid with marker 2. Red stars indicate the start point and the target point for the path finding mission. Gray spheres are safe regions found by sampling and nearest neighbor queries, and the blue dashed poly-line is the current best path discovered by a new sample.

in A* graph search, an under-estimate of the optimal cost for a node n to reach the target can be used as an admissible heuristic $n.h$. In our implementation, we use the Euclidean distance from the center coordinate of a node to the global planning target as an admissible heuristic. A node with a total estimated cost $n.f + n.h$ larger than the current best solution f_{best} is not considered promising and is deleted from the tree permanently.

Incremental Flight Corridor Refinement

For quadrotor applications, operating in partially or fully unknown environments is a common demand, especially for search-and-rescue missions where there is insufficient time for inspecting the working space beforehand. Therefore the onboard motion planning module should be able to react to unexpected obstacles detected in flight. To ensure the completeness of global navigation and achieve good motion consistency, we adopt an optimistic planning strategy, which treats all unknown space as free. When new unknown space is explored in the flights by onboard sensors, the globally registered point cloud is updated, as in Sect. 5.2.3, and nodes already in the random tree may be affected. If new obstacles are discovered inside a node, the node has to shrink or even become too narrow to be valid. However, checking every node in the tree when updating the point cloud is too costly. Also, there is no need to check the entire tree since most of the nodes

Algorithm 3 Safe-region RRT* Refinement

$t \leftarrow 0$
while $t \leq t^r_{max}$ **do**
  $\mathcal{T} \leftarrow$ **TreeExpand**$(\mathcal{T})$
  $t \leftarrow$ **Time**$()$
end while
$t \leftarrow 0$
$\{\mathcal{P}\} \leftarrow$ **PathExtract**$(\mathcal{T})$
while $t \leq t^e_{max}$ AND $\{\mathcal{P}\} \neq \varnothing$ **do**
  $\mathcal{C} \leftarrow \min(\{\mathcal{P}\})$
  if PathCheck$(\mathcal{C})$ **then**
    return $\mathcal{C}$
  end if
  $\{\mathcal{P}\} \leftarrow \{\mathcal{P}\} \backslash \mathcal{C}$
  $t \leftarrow$ **Time**$()$
end while
return $\varnothing$

Algorithm 4 PathCheck$(\mathcal{C})$

Require $\mathcal{P}$, $\mathcal{T}$
for $n_c \in \mathcal{C}$ **do**
  $n_c.r \leftarrow$ **RadiusSearch**$(n_c.c, \mathcal{P})$
  if VolumeCheck(n_c) **then**
    $\{n_{child}\} \leftarrow$ **Successor**(n_c)
    for $n_z \in \{n_{child}\}$ **do**
      if EdgeBreak(n_c, n_z) **then**
        $\mathcal{T} \leftarrow$ **BranchCut**$(\mathcal{T}, n_z)$
        return False
      end if
    end for
  else
    $\mathcal{T} \leftarrow$ **BranchCut**$(\mathcal{T}, n_c)$
    return False
  end if
end for
return True

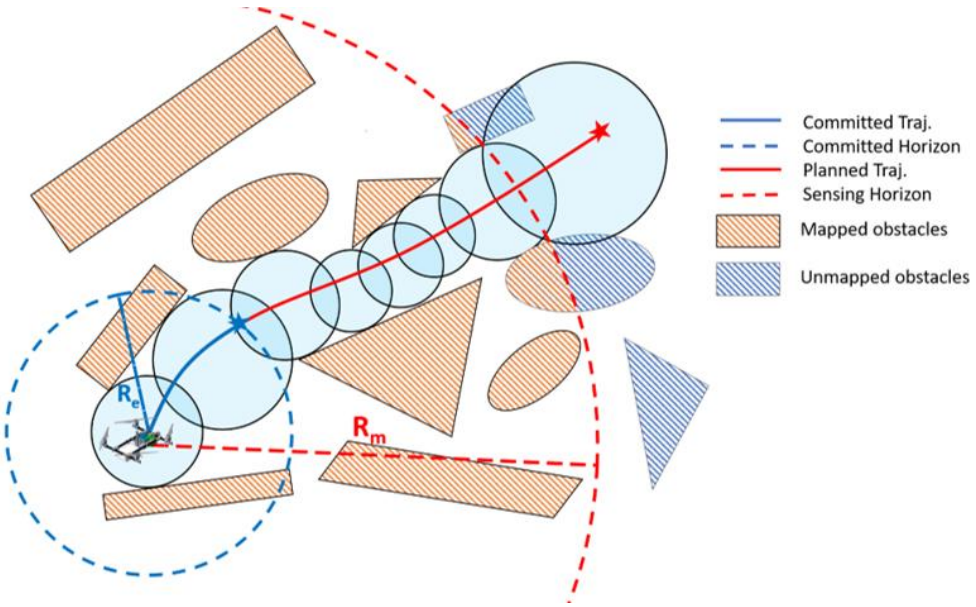


Figure 5.5. Trajectory commitment mechanism in a receding horizon. In this figure, the flight corridor is visualized as blue circles. The red curve is the generated trajectory to the global planning target, while the blue curve is the committed trajectory, which is reserved in a horizon for execution. The red dashed circle, and blue dashed arc indicates the sensing range and execution horizon, respectively. Mapped and unknown obstacles are in orange and blue hatching.

will not contribute to the optimal path. In this paper, we use the lazy-validity-checking strategy, similar to [36]. The main idea is that we only check the validity of the most promising nodes until a feasible optimal path (flight corridor) is found.

The incremental refinement procedure presented in Algo. 3 can be divided into two stages. First is the exploitation stages. Under a time limit t^r_{max}, new samples are continuously drawn, and the random tree is expanded, the same as in Algo. 1. After that, the re-evaluation stage starts. In this stage, all feasible paths are extracted as a set $\{\mathcal{P}\}$ from the tree. Under the lazy-validity-checking strategy, the optimal path $\mathcal{C}$ in $\{\mathcal{P}\}$ is checked. Details about **PathCheck**$(\mathcal{C})$ are given in Algo. 4. For each node n_c in C, the radius of the safe volume is updated against $\mathcal{P}$. If n_c fails in **VolumeCheck**(n_c), this path is labeled as infeasible and all branches of n_c are deleted from $\mathcal{T}$. Otherwise, if n_c only shrinks but is still feasible, the edges to all n_c's successor nodes are checked, and only branches with broken edges are deleted. The function **BranchCut**$(\mathcal{T}, n)$ used here deletes all branches of a node n from the tree $\mathcal{T}$. The re-evaluation process is terminated when $\{\mathcal{P}\}$ is empty, which means no path in the tree is feasible; or the pre-defined time t^e_{max} runs out. In our framework, a new trajectory is generated only if the quadrotor finishes the committed

trajectory. Therefore, when there exists no path, we delete the whole tree and re-initialize the safe region RRT* only if the quadrotor has finished the committed trajectory. Otherwise, Algo. 3 proceeds continuously, trying to find a feasible path by adding more and more samples.

5.4 Safe and Dynamically Feasible Trajectory Generation

As presented by Mellinger et al. [55], a quadrotor system enjoys the differential flatness property. The full state space of a quadrotor system can be reduced to independent 3-D positions, the yaw angle, and their derivatives. Trajectories of 3-D positions with derivatives bounded within the kinodynamic limits can, therefore, be followed by a properly designed geometric controller [47]. In this paper, since we use an omnidirectional LiDAR sensor, we leave out the planning of the yaw angle for brevity and represent the trajectory as a piecewise Bézier curve in each dimension μ out of *x, y, z.*

5.4.1 Piecewise Bernstein Basis Trajectory Formulation

In this paper, instead of using a monomial basis polynomial, we use the Berstein polynomial basis and represent the trajectory as a piecewise Bézier curve, which is a special case of a B-spline curve and has been introduced in Chap. 4. An m-segment piecewise Bézier curve in one dimension μ out of *x, y, z* can be written as follows:

$$f_{\mu}(t) = \begin{cases} s_1 \cdot \sum_{i=0}^{n} c_{\mu 1}^{i} b_n^i(\frac{t-T_0}{s_1}), & t \in [T_0, T_1] \\ s_2 \cdot \sum_{i=0}^{n} c_{\mu 2}^{i} b_n^i(\frac{t-T_1}{s_2}), & t \in [T_1, T_2] \\ \quad\vdots & \quad\vdots \\ s_m \cdot \sum_{i=0}^{n} c_{\mu m}^{i} b_n^i(\frac{t-T_{m-1}}{s_m}), & t \in [T_{m-1}, T_m], \end{cases} \tag{5.2}$$

where c_{ji} is the i^{th} control point of the j^{th} segment of the trajectory, $T_1, T_2, ..., T_m$ are the end times of each segment, the total duration of the Bézier curve is $T = T_m - T_0$, and $s_1, s_2, ..., s_m$ are the scale factors applied on each piece of the curve, to scale up the time interval from $[0, 1]$ to the allocated time $[T_{i-1}, T_i]$ of each segment. Note that we also multiply the scale factor in the position of each piece of the curve since, in practice, a scaled curve leads to better numerical stability for the following optimization program. The number of pieces of the trajectory is decided by the flight corridor found in Sect. 5.3. We assign each piece of the Bézier curve to one sphere of the flight

corridor.

Following the minimum snap formulation [55], the objective function we use for trajectory generation is the integral of the squared k^{th} derivative of the trajectory, which is written as

$$J = \sum_{\mu \in \{x,y,z\}} \int_0^T \left(\frac{d^k f_\mu(t)}{dt^k} \right)^2 dt. \tag{5.3}$$

J can be written in a quadratic formulation $\mathbf{c}^T \mathbf{Q}_o \mathbf{c}$, where $\mathbf{c}$ is a vector containing all control points c_{ij} in the *x, y, z* dimensions, and $\mathbf{Q}_o$ is the Hessian matrix of the objective function. In this paper we minimize the jerk along the trajectory, so k is 3. Suppose in the μ dimension the j^{th} segment of the trajectory is denoted as $f_{\mu j}(t)$. Then the pure Bézier curve (without scaling, as in Eq. **??**) in interval $[0, 1]$ is $g_{\mu j}(t)$. Denote $(t - T_j)/s_j = \tau$. Then in an axis μ, the minimum jerk objective function of the j^{th} segment is derived as

$$\begin{aligned} J_{\mu j} &= \int_0^{s_j} \left(\frac{d^k f_{\mu j}(t)}{dt^k} \right)^2 dt = \int_0^1 s_j \left(\frac{s_j \cdot d^k (g_{\mu j}(\tau)}{d(s_j \cdot \tau)^k} \right)^2 d\tau \\ &= \frac{1}{s_j^{2k-3}} \cdot \int_0^1 \left(\frac{d^k g_{\mu j}(t)}{d\tau^k} \right)^2 d\tau. \end{aligned} \tag{5.4}$$

The relationship between the control points $\mathbf{c}$ of the Bernstein basis and the coefficients $\mathbf{p}$ of the monomial basis at a given order n is fixed by a mapping matrix $\mathbf{M}$, i.e $\mathbf{p} = \mathbf{M} \cdot \mathbf{c}$. For instance, $n = 6$, we can derive

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -6 & 6 & 0 & 0 & 0 & 0 & 0 \\ 15 & -30 & 15 & 0 & 0 & 0 & 0 \\ -20 & 60 & -60 & 20 & 0 & 0 & 0 \\ 15 & -60 & 90 & -60 & 15 & 0 & 0 \\ -6 & 30 & -60 & 60 & -30 & 6 & 0 \\ 1 & -6 & 15 & -20 & 15 & -6 & 1 \end{bmatrix}. \tag{5.5}$$

Therefore, the Hessian matrix $\mathbf{Q}_\mu$ in one dimension, which only depends on the degree n and the scale s_j, can be derived in a closed-form following the classical minimum snap formulation [55]. And the overall objective $\mathbf{Q}_o$ is the sum of $\mathbf{Q}_\mu$ in each axis. We leave out the details of the $\mathbf{Q}_o$ for brevity. Note that at a given order n, the transformation between the Bernstein basis and monomial basis, as well as the formulation of the objective function, are all fixed and can be calculated

offline. Thus no extra overhead is introduced by using the Bernstein polynomial basis to formulate the problem.

5.4.2 Enforcing Constraints

For the piecewise trajectory generation problem, we must enforce a number of constraints to ensure the smoothness, safety, and dynamical feasibility of the trajectory. For each piece of the piecewise Bézier curve, higher-order derivatives of it can be represented by linear combinations of corresponding lower-order control points, which is written as

$$a_{\mu j}^{0,i} = c_{\mu j}^{i}, a_{\mu j}^{l,i} = \frac{n!}{(n-l)!} \cdot (a_{\mu j}^{l-1,i+1} - a_{\mu j}^{l-1,i}), \quad l \geq 1, \tag{5.6}$$

where l is the order of the derivative and n is the degree of the Bernstein basis.

Waypoint Constraints

The trajectory has to pass through several waypoints, such as the start and target positions of the quadrotor, as well as their derivatives up to n^{th} order. According to Property 1, a Bézier curve always passes the first and last control points. Therefore, waypoint constraints are directly enforced by setting equality constraints on corresponding control points. For a fixed l^{th} ($l \leq n$)-order derivative (waypoints) $d_{\mu j}^{(l)}$ that exists at the beginning[8] of the μ dimension, the j^{th} piece of the trajectory, we have:

$$a_{\mu j}^{l,0} \cdot s_j^{(1-l)} = d_{\mu j}^{(l)}. \tag{5.7}$$

Continuity Constraints

The trajectory must be continuous at all the ϕ^{th} ($0 \leq \phi \leq k-1$) derivatives at the connecting point between two pieces of the trajectory to ensure smoothness. The continuity constraints are enforced by setting equality constraints between corresponding control points in two consecutive curves, also based on Property 1. For the j^{th} and $(j+1)^{th}$ pieces of the curve, to set a continuity

[8]The constraint for a waypoint at the end of a curve is in the same formulation and is omitted here.

constraint at the ϕ^{th} order, we can write the equation as

$$a_{\mu j}^{\phi,n} \cdot s_j^{(1-\phi)} = a_{\mu,j+1}^{\phi,0} \cdot s_{j+1}^{(1-\phi)}. \tag{5.8}$$

Safety Constraints

Thanks to the convex hull property of the Bézier curve, which is illustrated in Property 3, an entire Bézier curve is confined within the convex hull formed by all its control points. Thus we can add safety constraints to force all control points of one segment to be inside the corresponding sphere generated in Sect. 5.3. For one segment of the trajectory, since the sphere is a convex space, and all control points are inside it, then the convex hull of these control points is surely inside the sphere, which means the entire segment of the curve is confined within the sphere. The safety constraints are applied by quadratic constraints on all control points. For control points of the j^{th} segment, constraints are

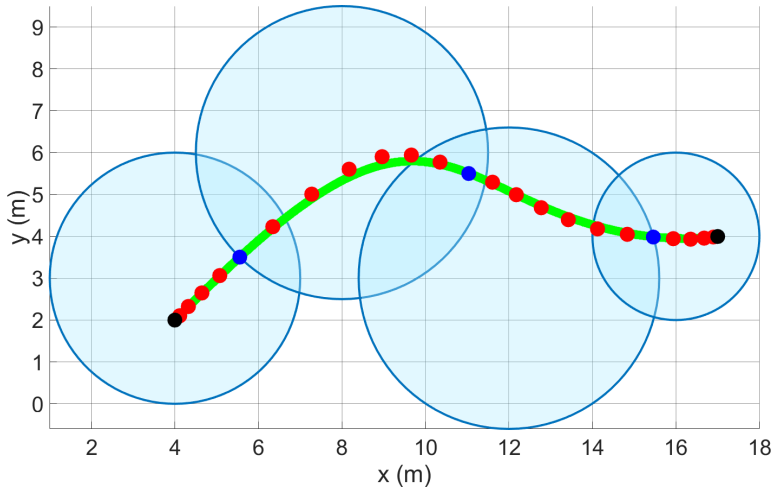
$$\sum_{\mu \in \{x,y,z\}} (c_{ji}^{\mu} - p_j^{\mu})^2 \leq r_j^2, \tag{5.9}$$

where p_j and r_j are the center and radius of the corresponding sphere in the j^{th} piece of the trajectory.

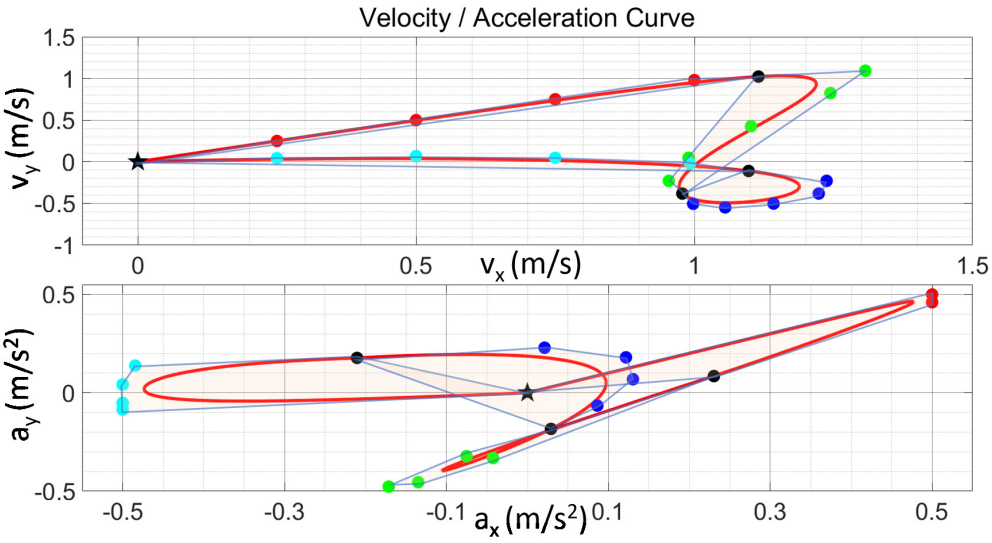
Dynamical Feasibility Constraints

Similarly to the safety constraints, utilizing Property 3 and 4, we can enforce hard constraints on higher-order derivatives of the entire trajectory. Constraints are introduced to bound the derivatives of the curve in each dimension μ. In this paper, the velocity and acceleration of the quadrotor are constrained in $[v_m^-, v_m^+]$ and $[a_m^-, a_m^+]$ to make the generated trajectory dynamically feasible. For control points of the j^{th} segment, we have

$$\begin{aligned} &v_m^- \leq n \cdot (c_{\mu j}^i - c_{\mu j}^{i-1}) \leq v_m^+, \\ &a_m^- \leq n \cdot (n-1) \cdot (c_{\mu j}^i - 2c_{\mu j}^{i-1} + c_{\mu j}^{i-2})/s_j \leq a_m^+. \end{aligned} \tag{5.10}$$



(a)



(b)

Figure 5.6. An illustration of a piecewise Bézier curve generated in the flight corridor. In (a), the curve is in green. The control points which are enforced by waypoint constraints and continuity constraints are shown in black and blue, and other intermediate control points are red. The corridor is shown in blue circles. In (b), velocity and acceleration of the trajectory are plotted in the x-y plane, with the limits set as $\pm 1.5 m/s$ and $\pm 0.5 m/s^2$. Control points with different colors indicate being in different pieces of the trajectory. Black dots indicate points on joint positions between two consecutive pieces.

5.4.3 Trajectory Optimization Formulation

The waypoint constraints and continuity constraints are directly formulated as linear equality constraints ($\mathbf{A}_{eq}\mathbf{c} = \mathbf{b}_{eq}$). The safety constraints are several convex quadratical inequality constraints

($\frac{1}{2}\mathbf{c}^T\mathbf{Q}_i\mathbf{c}+\mathbf{a}_i\mathbf{c} \leq \mathbf{r}_i$), and the higher-order dynamical constraints are formulated as linear inequality constraints ($\mathbf{A}_{ie}\mathbf{c} \leq \mathbf{b}_{ie}$); here $\mathbf{c} = [\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_m]$. To sum up, the trajectory generation problem is written as

$$\begin{aligned} \min \quad & \frac{1}{2}\mathbf{c}^T\mathbf{Q}_0\mathbf{c} \\ \text{s.t.} \quad & \frac{1}{2}\mathbf{c}^T\mathbf{Q}_i\mathbf{c} + \mathbf{a}_i\mathbf{c} \leq r_i, \;\; i = 1, \cdots, n \\ & \mathbf{A}_{eq}\mathbf{c} = \mathbf{b}_{eq}, \\ & \mathbf{A}_{ie}\mathbf{c} \leq \mathbf{b}_{ie}, \end{aligned} \tag{5.11}$$

where all quadratic matrices $\mathbf{Q}_i$ are positive semi-definite. The trajectory generation problem is formulated as a typical quadratically constrained quadratic program (QCQP), and can be solved in polynomial time by general off-the-shelf convex solvers. An illustration of the generated trajectory is given in Fig. 5.6 to show constraints enforced on control points.

In our trajectory generation framework, the dynamical feasibility of the quadrotor is guaranteed by bounding differences of positions between consecutive control points. Therefore, for a piece of the trajectory assigned in a large free volume (i.e., a long curve), the required order of the curve is high, and for a curve in a small free volume (i.e., a short curve), the order can be low. In this paper, we assign high-order Bézier curves for long segments and low-order Bézier curves for short segments, instead of setting uniform curve orders in all pieces of the trajectory, as in our previous work. By doing this, we can introduce fewer optimizing variables into the program, since representing higher-order curves requires more control points.

In practice, to ensure the constrained optimization program is always feasible and to improve the quality of the solution, one necessary condition is setting the order of the piecewise Bézier curve high to provide sufficient freedom in the solution space. Unfortunately, this requirement puts the program into a dilemma since a high order of the Bézier curve means more variables are included, which makes the scale of the problem very large. The numerical issue becomes severe, and the QCQP easily becomes to be ill-posed on a large scale, making the program hard to solve. To address the numerical issue, we propose an approach to reformulate the trajectory optimization problem further.

5.4.4 Conic Optimization Reformulation

To address the numerical issue of the program and improve efficiency, we convert the above QCQP to a SOCP. Compared to the QCQP, modeling the optimization as a SOCP gives not only a faster solving time but also better numerical stability. The SOCP has sounder duality theory, making it more able to report accurate dual information, and thus be robust to numerical errors. Also, formulating a problem as a SOCP indicates that it is theoretically convex, making the check of numerical convexity simpler for most of the convex solvers. For a detailed discussion about the robustness and efficiency of QCQPs and SOCPs, one can refer to [2].

For our QCQP formulation in Eq. 5.11, given a quadratical constraint in the canonical form

$$\frac{1}{2}\mathbf{c}^T\mathbf{Q}_i\mathbf{c} + \mathbf{a}_i\mathbf{c} \leq r_j, \tag{5.12}$$

we can convert it to a rotated second-order cone by introducing additional variables. Define w_i such that

$$w_i = r_i - \mathbf{a}_i\mathbf{c}. \tag{5.13}$$

Then an equalivent form of the quadratical constraint is derived as

$$\begin{aligned} \frac{1}{2}\mathbf{c}^T\mathbf{Q}_i\mathbf{c} &\leq w_i, \\ r_i - \mathbf{a}_i\mathbf{c} &= w_i. \end{aligned} \tag{5.14}$$

Since all $\mathbf{Q}_i$ in Eq. 5.11 are positive semi-definite, they can always be factorized as

$$\mathbf{Q}_i = \mathbf{F}_i^T \cdot \mathbf{F}_i \tag{5.15}$$

by Cholesky decomposition or other methods. Then by combining Eq. 5.14 and Eq. 5.15, we can derive

$$\begin{aligned} (\mathbf{F}_i \cdot \mathbf{c})^2 &\leq 2w_i, \\ r_i - \mathbf{a}_i\mathbf{c} &= w_i, \end{aligned} \tag{5.16}$$

which can be further re-written in an equaivalent conic formulation:

$$(w_i, t_i, \mathbf{y}_i) \in \mathbf{K}_r^{k_y+2}, \tag{5.17}$$

$$\begin{cases} w_i = r_i - \mathbf{a}_i\mathbf{c}, \\ t_i = 1, \\ \mathbf{y}_i = \mathbf{F_i} \cdot \mathbf{c}, \end{cases} \tag{5.18}$$

where k_y is the dimensionality of $\mathbf{y}_i$, and $\mathbf{K}_r^{k_y+2}$ indicates a rotated second-order cone with the dimensionality of $k_y + 2$. An n-dimensional rotated quadratic cone $\mathbf{K}_r^n$ is matimatically defined as $\{\mathbf{x} \in \mathcal{R}^n \mid 2x_1x_2 \geq x_3^2 + \ldots + x_n^2, x_1, x_2 \geq 0\}$. $w_i, t_i, \mathbf{y}_i$ are additional slackness variables. Similarly, the objective $\frac{1}{2}\mathbf{c}^T\mathbf{Q}_0\mathbf{c}$ is also reformulated as a rotated second-order cone. Finally, the trajectory optimization problem is converted to a typical SOCP with an affine objective function, and affine and conic constraints:

$$\begin{aligned} \text{min} \quad & w_0 \\ \text{s.t.} \quad & (w_i, t_i, \mathbf{y}_i) \in \mathbf{K}_i^{r,k_y+2}, \quad i = 0, \cdots, n \\ & w_i = r_i - \mathbf{a}_i\mathbf{c}, \qquad\qquad i = 1, \cdots, n \\ & \mathbf{A}_{eq}\mathbf{c} = \mathbf{b}_{eq}, \\ & \mathbf{A}_{ie}\mathbf{c} \leq \mathbf{b}_{ie}, \end{aligned} \tag{5.19}$$

where $t_i = 1$ and $\mathbf{y}_i = \mathbf{F}_i \cdot \mathbf{c}, i = 0, \cdots, n$. Note that here all the factorized matrices $\mathbf{F}_i$ are constant, which depends only on the order of a Bézier curve and can be calculated offline. Therefore, no additional computing cost is included by online reformulating the program. Compared to the original QCQP formulation, although more variables are introduced as slackness variables, the better numerical stability and convergence of second-order cones often dominate the quality of the solution in practice. The SOCP formulation significantly improves the robustness and reduces solving time, which are verified in our experiments in Sect. 5.5.2.

5.5 Results

5.5.1 Implementation Details

In this section, we compare our proposed method against benchmarks and present extensive field experiments. The planning method proposed in this paper[9] is implemented in C++11 using a general convex solver Mosek[10]. Moreover, the benchmark methods used in this paper are implemented by us following the paper [10, 71]. For fair comparisons, parameters in the benchmark methods are tuned to achieve the best performances and balance the efficiency and success rate. We validate our system in both indoor and outdoor cluttered environments, which reflect the difficulty and complexity in most field search-and-rescue missions. In our experiments, the information of the environments is previously unknown to the quadrotor, and all processing is done onboard. The flights of the quadrotor in all trials are fully autonomous, without any human operations or interventions. In our onboard tests, the Velodyne 3-D LiDAR runs at $20Hz$ and outputs 15,000 points in each scan. The state estimation module runs at the same rate as the sensor, while the registered point clouds for motion planning are updated at $10Hz$. The re-planning frequency is also $10Hz$. Although the Velodyne LiDAR can detect obstacles as far as $100m$, the sensing range in our system is set as $30m$ to ensure the accuracy of the range measurements, reduce the amount of data, and mimic the normal performance of common range sensors in search-and-rescue applications. The commitment time t_m is set as $6s$, considering the sensing range and the maximum allowed speed of the quadrotor. The time limit for initializing our path finding module is $80ms$. In each re-planning loop, the time budgets for refining and re-evaluating the path are $60ms$ and $40ms$, respectively. The maximum radius of the safe region in the flight corridor is $5m$.

5.5.2 Numerical Tests of Trajectory Generators

Instead of using the monomial polynomial basis, we use the Bernstein polynomial basis and represent the trajectory as piecewise Bézier curves (Sect. 5.4) to avoid the iterative procedure for constraining the trajectory. Furthermore, we reformulate the QCQP to an equivalent SOCP. Here we validate the superiority of such basis selection and problem reformulation by comparing the

[9] Source code of the proposed motion planning framework will be released in `https://github.com/HKUST-Aerial-Robotics/pointcloudTraj` after the publishing of this paper.

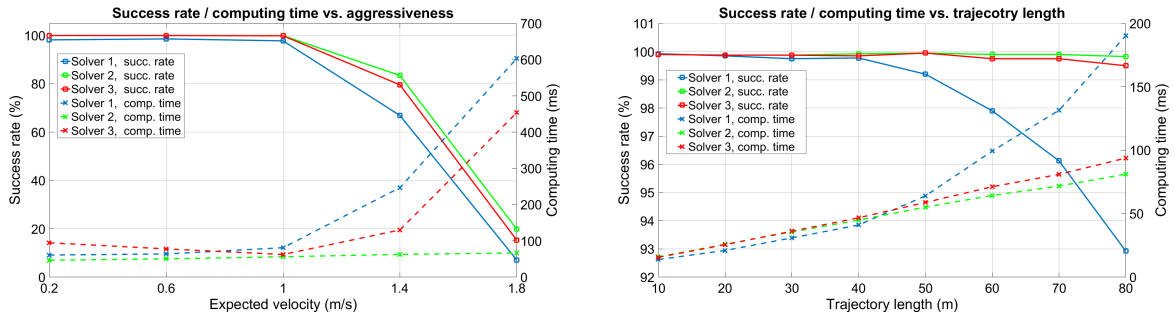
[10] `https://www.mosek.com`

trajectory generated in:

1. the monomial polynomial basis with the iterative constraining strategy [29];
2. the Bernstein polynomial basis with the problem in the QCQP formulation (Sect.5.4.3);
3. the Bernstein polynomial basis with the problem in the equivalent SOCP formulation (Sect. 5.4.4).

We construct numerous instances to apply the above three solvers and compare the solutions. We randomly generate ten maps, with the fixed obstacle number as 400. The limit of iterations in the monomial polynomial QCQP solver is set at 5. The maximum allowed velocity and acceleration are fixed as $2m/s$ and $2m/s^2$. Although in practice a sophisticated heuristic is useful for proper time allocation, in the comparison we use the average velocity to calculate the allocated time because using the average velocity is convenient to indicate the expected aggressiveness of trajectories.

The results are shown in Fig. 5.7, and demonstrate two important conclusions. Firstly, formulating the trajectory generation problem in a non-iterative hard-constrained formulation has a higher chance to find feasible optimal solutions. Moreover, the SOCP formulation has a higher success rate than the QCQP, especially at high aggressiveness, where the program is often considered “hard" to solve. The reason is that, in general, a SOCP is much more numerically stable than a QCQP, and more easily converges to the global optimal solution. Secondly, the proposed SOCP formulation requires less computational time than others, especially when the problem is “hard". Even if the problem is indeed infeasible, the SOCP formulation enjoys numerical stability and strong duality and determines the infeasibility much more easily. The iterative monomial solver needs the most time to determine the status of the problem.

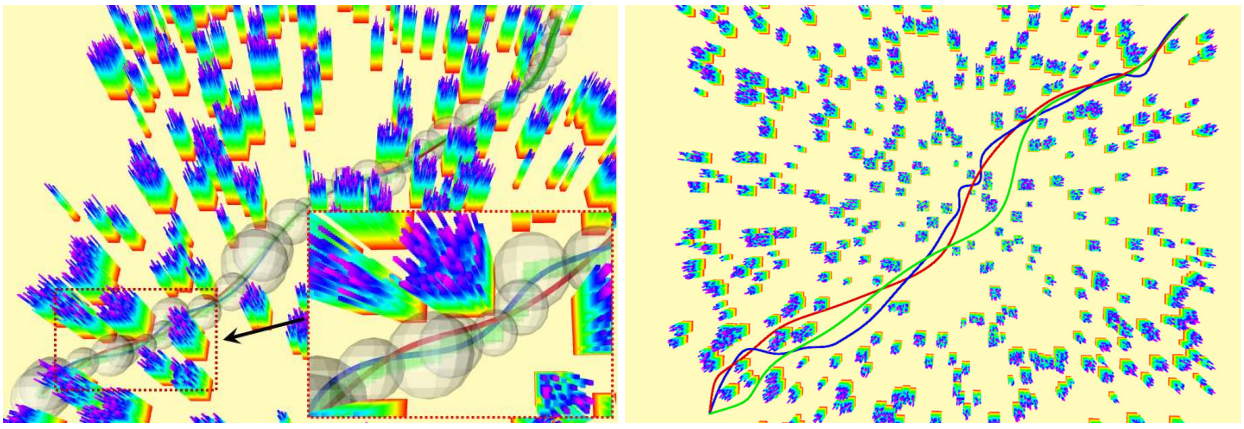


(a) Trajectories generated against varying aggressiveness.

(b) Trajectories generated against varying length.

Figure 5.7. The results of generating trajectories against different expected average velocities in (a) and against different expected trajectory lengths in (b). Success rate and computing time are compared. Solver 1 refers to the monomial basis iterative formulation used in our previous work [29], solver 2 is the Bernstein basis QCQP formulation given in Sect. 5.4.3, and solver 3 refers to the conic solver we finally derived in Sect. 5.4.4. The success rate for each method is shown by cube markers and the computing time is represented by cross markers.

5.5.3 Benchmark Results



(a) Comparison of trajectory optimization.

(b) Comparison of planning framework.

Figure 5.8. The comparison of the trajectory optimization in (a) and the comparison of the complete planning framework in (b). In (a), red and blue curves are generated by our proposed method and **Benchmark 1** [10], respectively. White transparent spheres indicate our flight cooridor and green transparent cubes are the path constraints in **Benchmark 1**. In (b), red, green and blue curves are respectively generated by our previous method [29], our proposed method, and **Benchmark 2** [71].

We present benchmarked comparisons against several other works. Firstly, we compare the proposed trajectory optimization method against [10], which also guarantees the safety and kinody-

namical feasibility by solving a single convex optimization program. Secondly, we compare the complete motion planning framework in this paper (including front-end and back-end) against the state-of-the-art system-level quadrotor planning work [71] and our previous work [29]. Trajectories generated in comparisons are shown in Fig 5.8.

Comparison of the Trajectory Optimization

We compare the back-end trajectory optimization in our proposed motion planning framework against the benchmark method from [10], which is denoted as **Benchmark 1**. In the comparison, we set the velocity limit of the quadrotor as $2m/s$ and the acceleration limit as $2m/s^2$. Unlike our method, in the benchmark method, the acceleration and velocity are not both defined by the physical limit. Instead, a conservative velocity limit is derived based on a minimum path clearance and the acceleration limit. We set the minimum path clearance at $0.5m$.

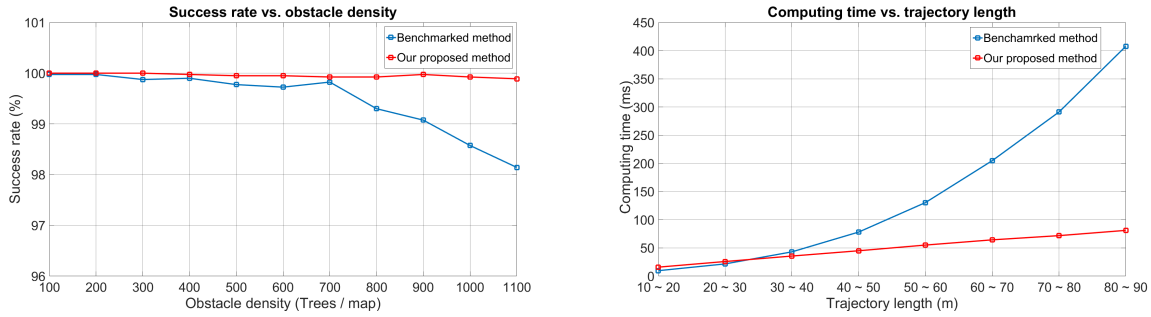
We first fix the obstacle density as 500 trees/map to conduct an overall comparison. We randomly generate ten maps and do 400 trials on each one. As shown in Table 5.1, **Benchmark 1** is over-conservative compared to our method. Since the velocity bound is decided by the minimum path clearance, in a cluttered environment, the quadrotor can never obtain a relatively high speed. Even though we set a substantially higher path clearance ($0.5m$) compared to the value ($0.035m$) used in the original paper [10], the average velocity of the generated trajectory is far lower than the physical limit. Moreover, the average velocity can never be tuned higher due to the over-conservative formulation used to derive the safety guarantee. For the same reason, the lower resulting objective in **Benchmark 1** is obtained by taking a much longer time to finish the trajectory.

We also present the comparisons of the success rate vs. obstacle density and the computational time vs. trajectory length. We generate trajectories in maps with obstacle densities ranging from 100 trees to 1100 trees per map. Then we fix the obstacle density as 600 trees per map and generate trajectories with different lengths by choosing the location of the target coordinates. Results are given in Figs. 5.9(a) and 5.9(b). As presented in the figure, the performance of our proposed method is not sensitive to the obstacle density nor the trajectory length. At high obstacle density and trajectory length, the benchmark method, which builds the flight corridor in a very conservative way, introduces many more decision variables into the QP. Therefore numerical issues become serious and result in failures or a much longer solving time. In contrast, our proposed method is

Table 5.1. Comparison of Trajectory Optimization

Method	Lenth (m)	Obj.	Time (ms)	Ave. Vel.	Ave. Acc.	Max. Vel.	Max. Acc.	Succ. Rate (%)
Proposed	80.023	3.3064	88.60	0.8874	0.0995	1.7568	0.9355	99.88
Benchmark 1	82.024	0.9206	333.21	0.3815	0.0282	0.9804	0.7452	99.65

almost surely able to find a safe and kinodynamically feasible optimal solution, not only because of the mathematically safe and kinodynamic guarantee achieved by the construction but also because of the numerical robustness in the conic formulation.



(a) Successful rate against varying obstacle density. (b) Computing time against varying length.

Figure 5.9. The comparison of the trajectory optimization against varying obstacle density (a) and trajectory length (b). Results of our proposed method and benchmark method [10] are respectively in red and blue. 400 trials are conducted for each case.

Comparisons of the Planning Framework

We also conduct system-level comparisons with both the front-end path finding and back-end trajectory generation sub-modules. The comparisons are made against our previous method and the state-of-the-art quadrotor planning framework [71], which is often called the waypoints-based method or unconstrained QP method and is denoted as **Benchmark 2** here. In **Benchmark 2**, RRT* is used to find a collision-free path which consists of a series of waypoints, followed by an unconstrained QP to get the solution of the minimum-jerk/snap trajectory in closed-form. The safety of the trajectory is achieved by iteratively adding new waypoints in the middle of the segment of the trajectory where a collision occurs. The path finding method in our previous work is equivalent to the initialization stage of the safe region RRT* method proposed in this paper. Therefore their performances do not have many differences in this test. We set a time budget for both

Table 5.2. Comparison of Motion Planning Framework

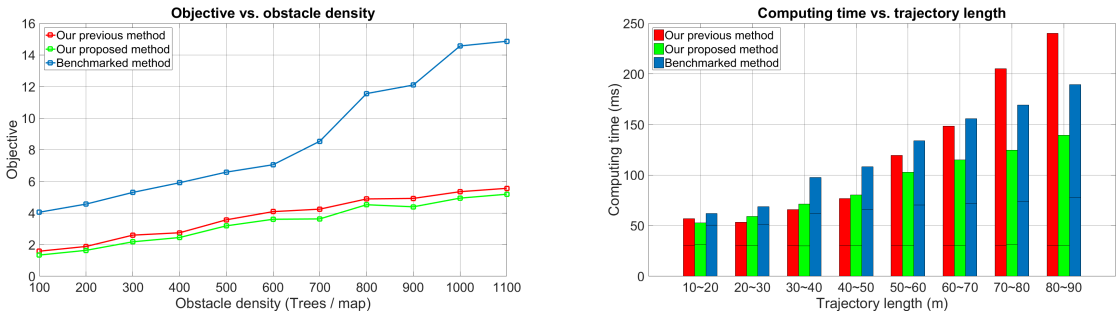
Method	Lenth (m)	Obj.	Path Time (ms)	Traj. Time (ms)	Path Rate (%)	Traj. Rate (%)	Kino. Rate (%)
Proposed	79.420	3.455	50.91	84.28	99.35	99.08	99.08
Previous	79.553	3.593	50.44	102.75	99.23	94.50	94.50
Benchmark 2	81.871	25.366	97.43	73.47	100.00	98.35	76.35

our previous and proposed methods to find a path using their front-end. Then we use the RRT* implementation in the Open Motion Planning Library (OMPL)[11] to find a path for the **Benchmark 2**. For a fair comparison, we set the termination condition of the RRT* at a comparable path length (within 102%) to our proposed path finding module. The loop limit of the iterative minimum-jerk trajectory generator in **Benchmark 2** is set as 20, which is decided by balancing the success rate and computation time.

We first conduct an overall comparison to show the detailed performances of each method. Results are presented in Table 5.2. We randomly generate ten different maps which have 500 trees/map. In each map, 400 tests are performed with a random target. Maximum velocity and acceleration are $2m/s$ and $2m/s^2$, respectively. The time for each piece of the trajectory is allocated using an average velocity of $1m/s$, and the time budget of our front-end modules for finding the path is $50ms$. The most noticeable difference is that compared to our corridor-based methods, the waypoint-based method generates trajectories with much higher objective values since in our method the geometric constraints which are introduced by the flight corridor give much more freedom for the optimization. Also, the iterative procedure in the benchmark method tends to force the trajectory to oscillate, or in extreme situations, zigzag to avoid obstacles. This property results in unnecessary jerky trajectories with higher objectives. For the same reason, the benchmark method tends to generate longer trajectories. It also finds the path more slowly than ours but more quickly generates collision-free trajectories with a comparable success rate. In Table 5.2, the trajectory success rate means the collision-free rate, distinct from the kinodynamic feasibility rate. However, our proposed method is advanced on the hard constrained physical feasibility guarantee and thus has a much higher certainty in ensuring kinodynamic feasibility.

We also compare the performances against varying obstacle density and trajectory length. The

[11] http://ompl.kavrakilab.org/

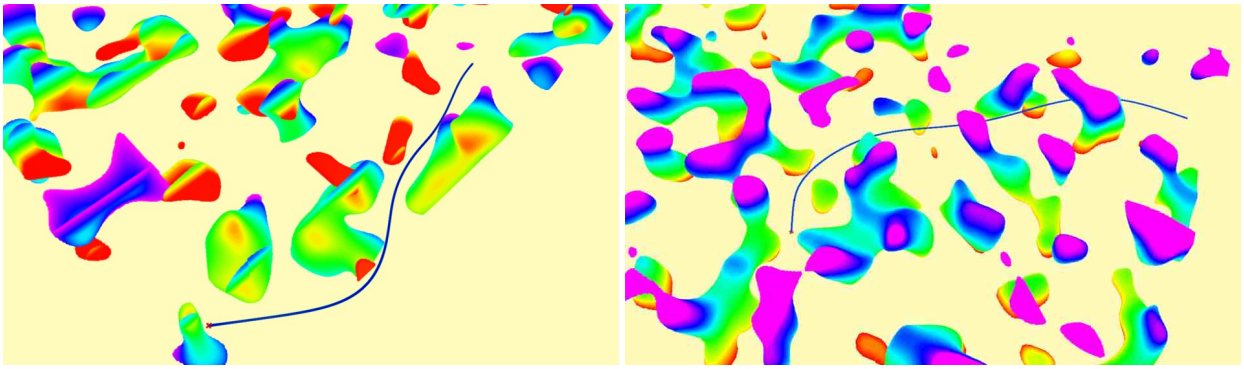


(a) Objective against varying obstacle density.

(b) Computing time against varying trajectory length.

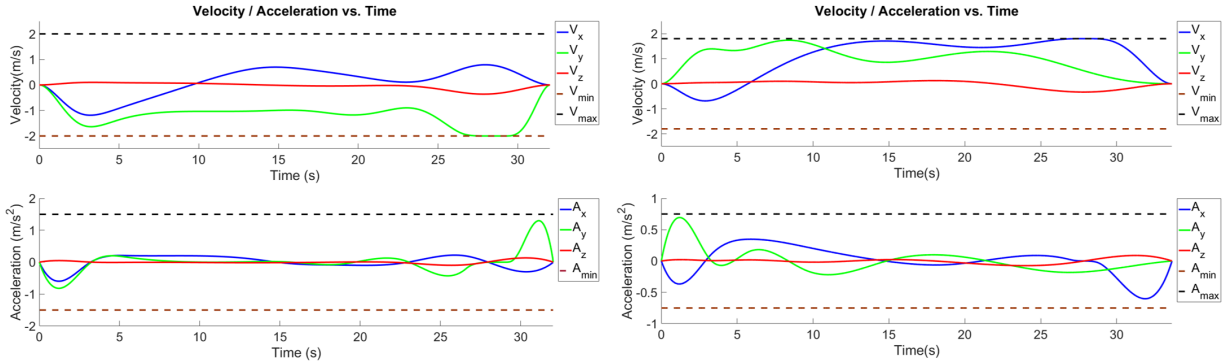
Figure 5.10. The comparison of the generated trajectories against varying obstacle densities (a) and trajectory length (b). Results of our proposed method are shown in green. Results of our previous method are plotted in red. And blue lines and bars indicate the results of the benchmark method. In (b), times below and above the horizon black line in each bar indicate time consumed in path finding and trajectory generation, respectively.

objective of **Benchmark 2** is much higher than those of both our previous and proposed methods, as shown in Fig. 5.10(a). Also, it increases much more rapidly when the map becomes denser. The reason is that the iterative “adding waypoints" strategy is ineffective, especially in environments with a high obstacle density, since in each iteration, additional waypoints are added considering only the local information of the trajectory. In contrast, our method requires the trajectory to stay within the corridor and constructs the safety guarantee globally. In Fig. 5.10(b), we fix the obstacle density as 600 trees/map and compare the computing time against the length of the trajectory. In the figure, the time above the horizontal line represents the overhead in the trajectory generation, and the time below is that in path finding. The benchmark method needs slightly more time to find a comparable path. This is because in our algorithm no collision checking is needed, as stated in Sect. 5.3, so a significant amount of overhead is saved. For trajectory generation, our proposed method takes more time when the trajectory is short. But as the length of the trajectory increases, the computing time of the benchmark method and our previous method increases much more rapidly than that of our proposed method. Since our previous method and the benchmark method are both iteratively solved, when the trajectories become longer or, the environments become denser, finding a feasible solution becomes much more difficult. Even if no feasible solution exists, these two methods can only detect the infeasibility after iterations, while our proposed method only needs to solve the program once, no matter how complicated the corridor is or whether a feasible solution exists.



(a) Trajectory generated with low aggressiveness.

(b) Trajectory generated with high aggressiveness.



(c) Velocity/acceleration with low aggressiveness.

(d) Velocity/acceleration with high aggressiveness.

Figure 5.11. Two instances of trajectory generation. Velocities and accelerations in x, y, z axis are plotted against time in the bottom row of the figure and are shown to be entirely constrained within limits. Time is allocated using low aggressiveness in (a) and (c) and high aggressiveness in (b), (d).

5.5.4 Simulated Flights

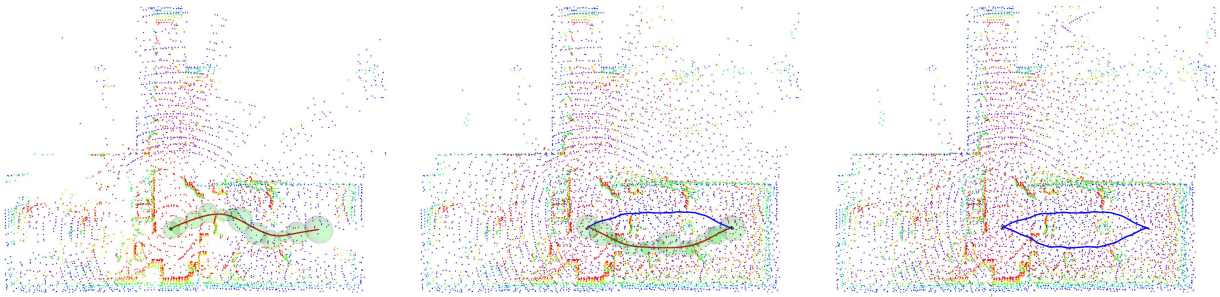
We also validate the hard kinodynamic constraints enforced on trajectories in simulation. As is shown in Fig. 5.11, two instances are presented in a random Perlin-noised-map. In the 1^{st} test (Fig. 5.11(a)), time is allocated based on a low average velocity ($1m/s$) and the kinodynamic bounds are $\pm 2m/s$ and $\pm 2m/s^2$. The 2^{nd} test (Fig. 5.11(a)) is relatively difficult with an aggressive time allocation ($1.5m/s$) and very tight kinodynamic bounds ($\pm 1.8m/s$ and $\pm 0.75m/s^2$). As presented in the figure, velocities and accelerations are completely bounded within the corresponding maximum/minimum limits without any violation. Details of the simulated flights are presented in the attached video.

5.5.5 Indoor Experiments

Indoor experiments are done to validate that our quadrotor platform is stable and agile enough to handle small-scale indoor situations. Obstacles are randomly deployed in our laboratory, and the quadrotor is expected to autonomously pass through them to given targets without any prior knowledge about the environments. The kinodynamic limits for velocity and acceleration are $\pm 2m/s$ and $\pm 1m/s^2$. And we reserve $0.15m$, considering the size of the quadrotor, and $0.2m$, accounting for control error, as a safety margin for each safe region in the flight corridor. We present two indoor tests here, a simple case in the flight area and a complicated case through the workspace in our laboratory. Snapshots of the flights and overview of the tracking trajectory are shown in Fig. 5.12. For online visualization of all generated trajectories and the re-planning mechanism in the flight, we refer readers to the video attachment.

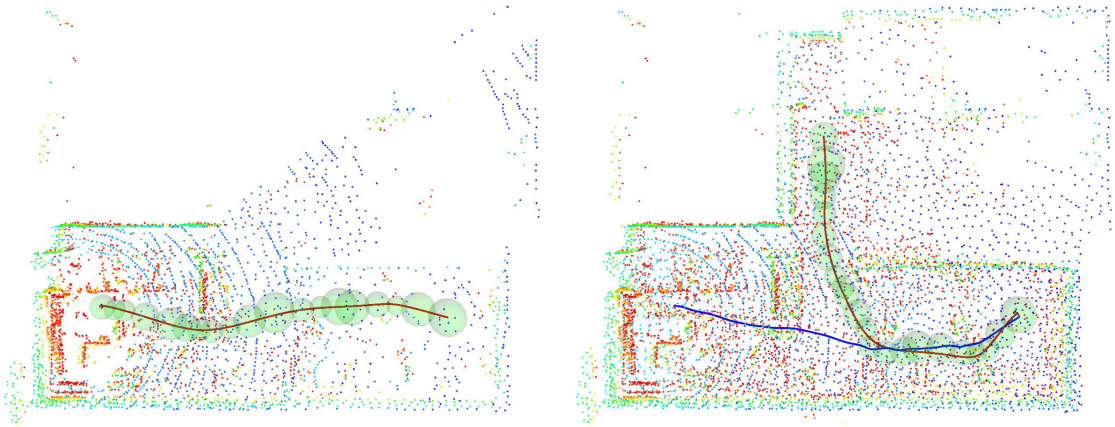
(a) Snapshot of the quadrotor in the simple test. (b) Snapshot of the quadrotor in the complex test.

Figure 5.12. Our autonomous quadrotor platform in the indoor tests.

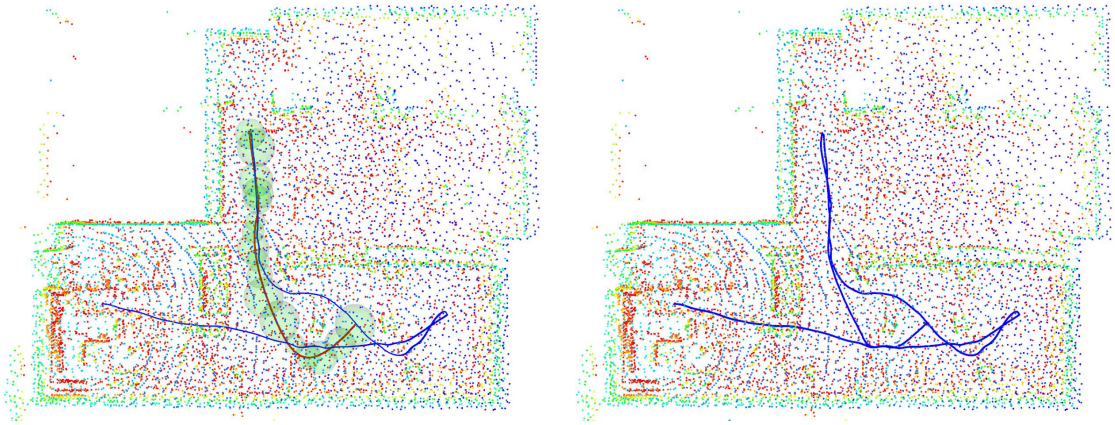


(a) The 1^{st} target. (b) The 2^{nd} target. (c) Overview of the flight

Figure 5.13. Trajectories generated in the simple indoor autonomous flight. Two targets are given sequentially in this experiment. The generated trajectory and the tracking path of the quadrotor are shown as the red and the blue curve, respectively. The flight corridor of the quadrotor is shown as a series of transparent green spheres. The quadrotor is shown in a black mesh model.



(a) Trajectory generated after the 1^{st} target received. (b) Trajectory generated after the 2^{nd} target received.



(c) Trajectory generated after the 3^{rd} target received. (d) Overview of the flight.

Figure 5.14. Trajectories generated in the complicated indoor autonomous flight test. Markers are interpreted as the same in Fig. 5.13. Three navigating targets are given in this experiment.

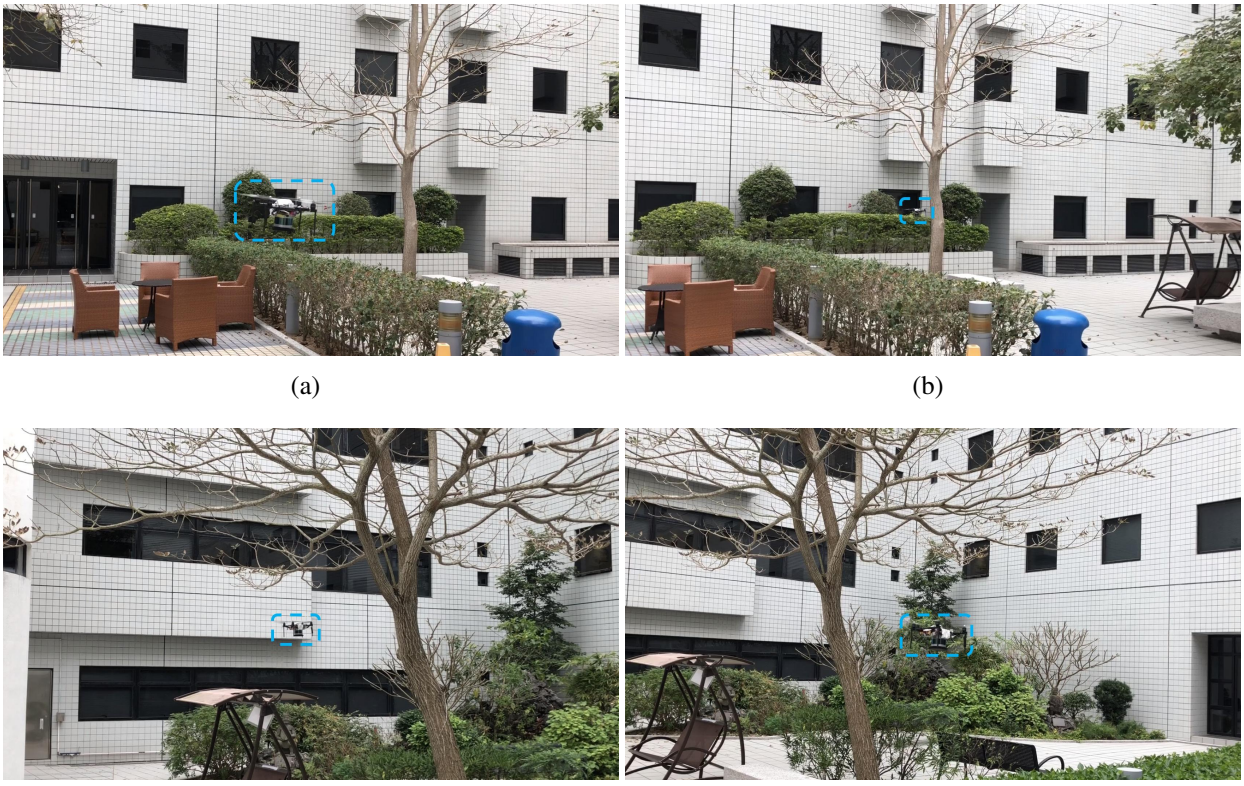
5.5.6 Outdoor Experiments

Search-and-rescue missions are often required to be conducted outdoors but in GPS-denied environments, such as cluttered forests or collapsed buildings after a disaster. In this section, we validate that our proposed motion planning framework is applicable in the -above-mentioned cases. We select several outdoor environments for testing, a human-made structured environment, an unstructured environment that includes a small building that the quadrotor can enter and leave, and an unstructured dense forest. The parameter settings for conducting the outdoor experiments are the same as in the indoor experiments, except the time limit for initializing the path finder is set higher, as $100ms$, due to the larger scale of the environments. The functionalities of the GPS and

magnetometer in the DJI A3 autopilot of our quadrotor are disabled.

Structured Environment

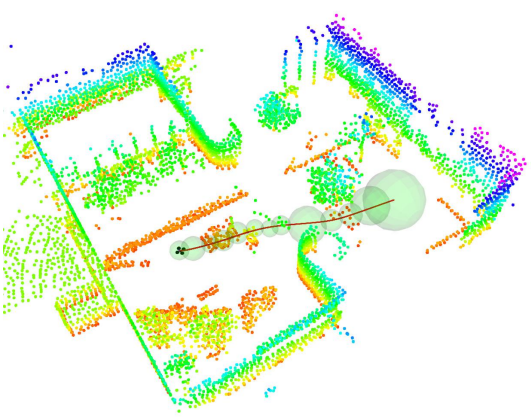
First of all, we conduct the outdoor experiments in a garden courtyard, which is human-made and highly structured. Information about the environment is previously unknown to the quadrotor and all processing, including estimation, mapping and planning, are done online and onboard. In the experiment, three navigation targets are sent to the quadrotor sequentially, and 20 trajectories are generated in the flight with an average computing time of $34.76ms$. The total traversal distance of the quadrotor is $73.71m$. Photos of the garden courtyard and the quadrotor in flight are shown in Figs. 5.15(a) and 5.15(c). The overview of the flight is shown in Figs. 5.17(a) and 5.17(b). Note that point clouds on the ground are removed for clear visualization.



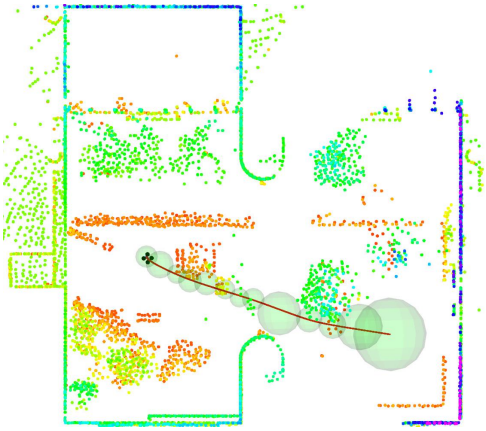
(a) (b)

(c) (d)

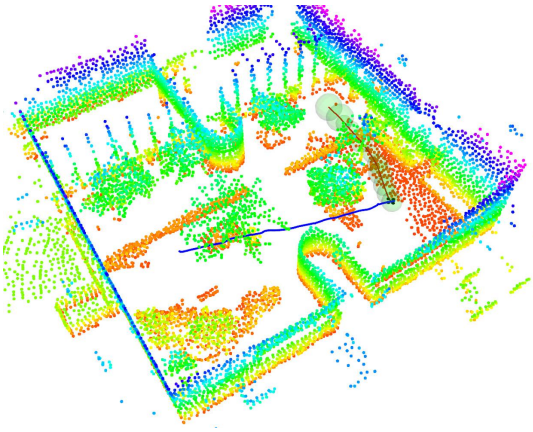
Figure 5.15. Snapshots of the autonomous flights in a garden lobby. In the testing scene, obstacles are of various types and shapes, including trees, chairs, bushes, and swings. The information of the environment are previously unknown to our quadrotor, and all operations are done in onboard without human interventions.



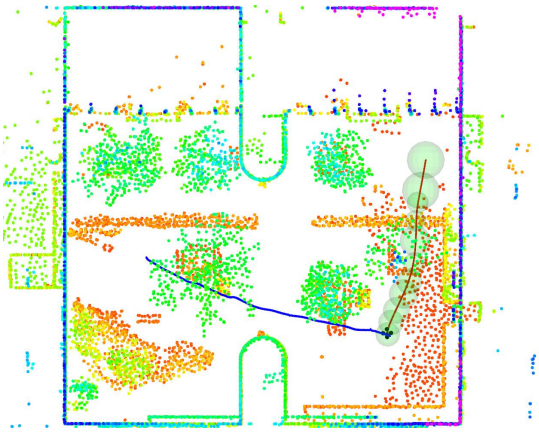
(a) Trajectory generated after the 1^{st} target received.



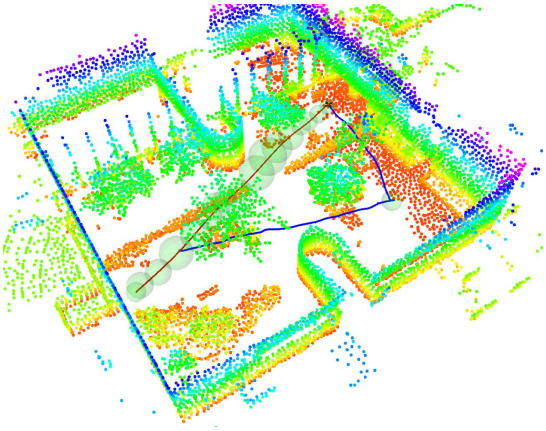
(b) Down-ward view.



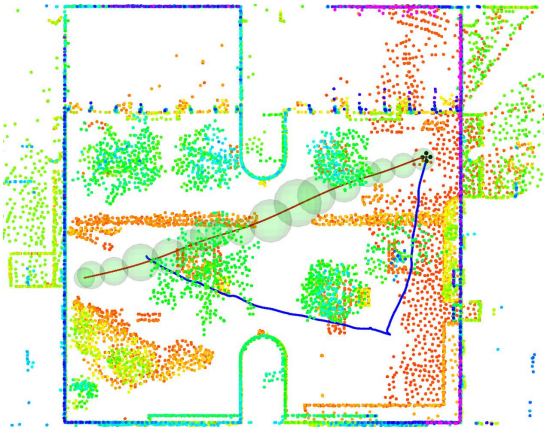
(c) Trajectory generated after the 2^{nd} target received.



(d) Down-ward view.

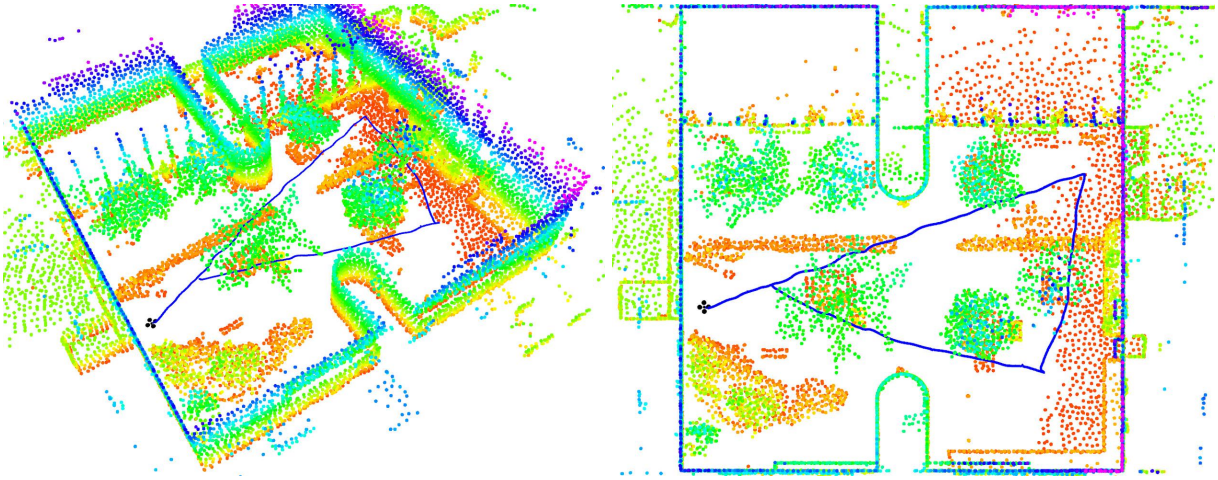


(e) Trajectory generated after the 3^{rd} target received.



(f) Down-ward view.

Figure 5.16. Trajectories generated in the outdoor autonomous flight test in a structured environment. Markers are interpreted as the same in Fig. 5.13. Three navigating targets are given sequentially in this experiment. Trajectories generated after receiving targets are visualized. In these figures, points on the ground floor of the point clouds are removed, and the quadrotor mesh model is scaled-up for clear visualization.



(a) Overview of the autonomous flight. (b) Down-ward view.

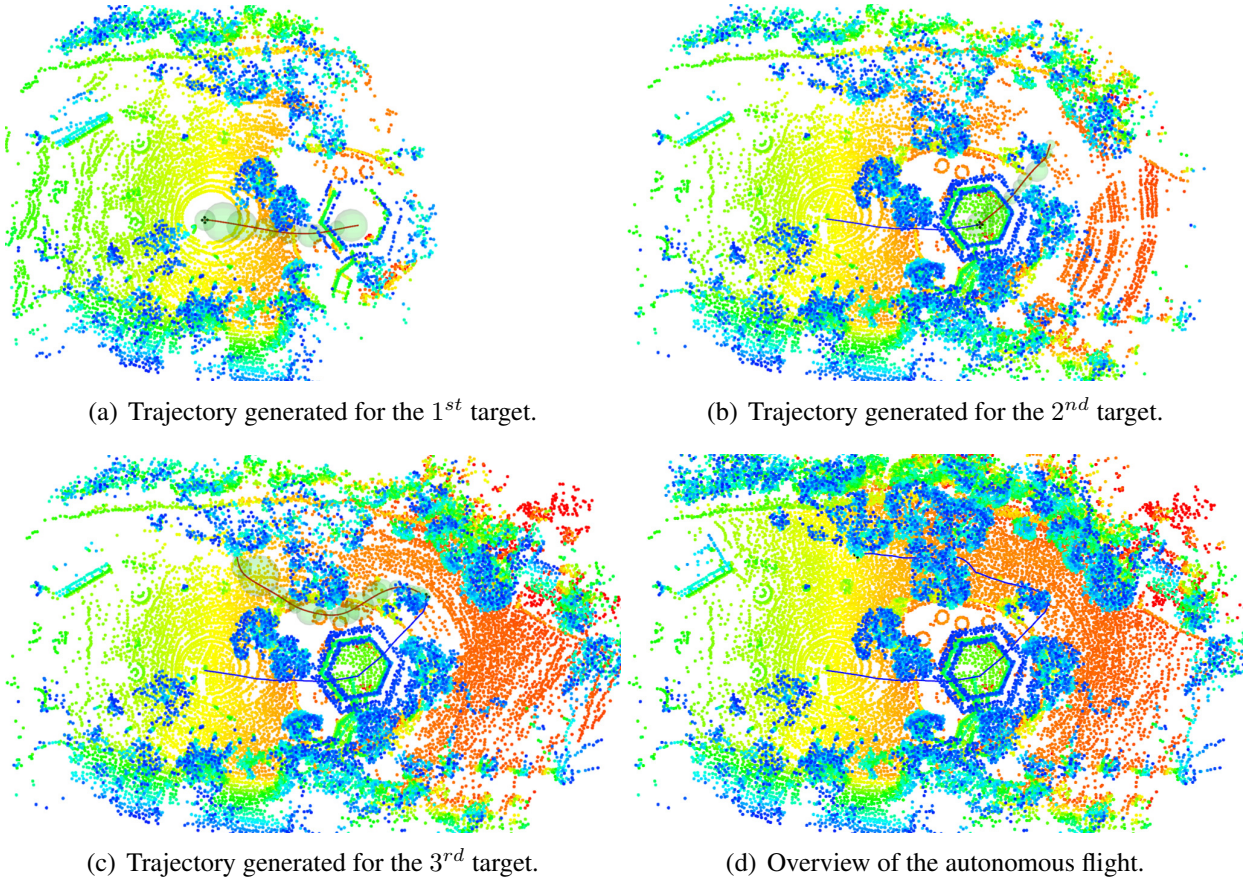
Figure 5.17. Overview of the quadrotor flights in the testing scene. The tracking path of the quadrotor is shown as the blue curve. Points on the ground floor of the point clouds are removed, and the quadrotor mesh model is scaled-up for clear visualization.

Unstructured Environment

We also test our autonomous system in an unstructured environment that includes a small building (a pavilion) surrounded by water. We assign planning targets for the quadrotor to fly inside and exit the building. Parameter settings are the same as in the previous outdoor experiment. In the experiment, three navigation targets are sent to the quadrotor sequentially, and 18 trajectories are generated in the flight with an average computing time of $29.71ms$. The total traversal distance of the quadrotor is $64.82m$. The results of the experiment are shown in Fig. 5.19. Here we only show trajectories generated after receiving targets.

(a) Snapshot of the quadrotor in the test. (b) Snapshot of the quadrotor in the test.

Figure 5.18. Autonomous flights in the unstructured outdoor environment.



(a) Trajectory generated for the 1^{st} target.

(b) Trajectory generated for the 2^{nd} target.

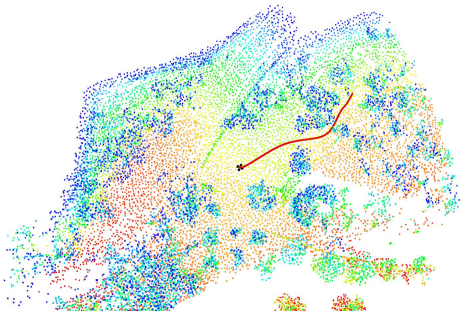
(c) Trajectory generated for the 3^{rd} target.

(d) Overview of the autonomous flight.

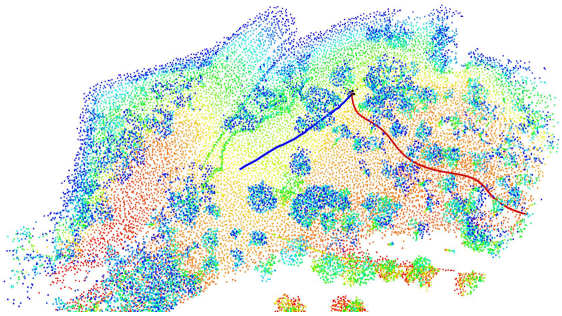
Figure 5.19. Outdoor autonomous flight tests in an unstructured environment. Trajectories generated after receiving targets are visualized.

Dense Forest

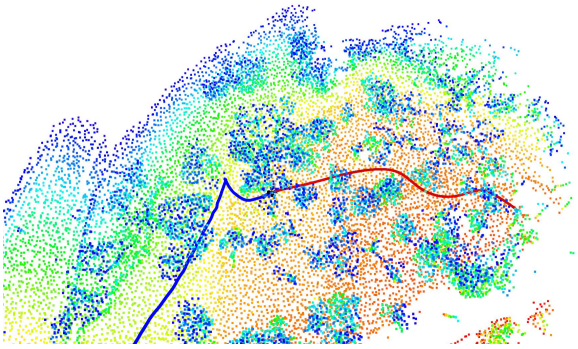
In this experiment, we validate our autonomous flight system in a dense forest on a rugged hillside. This evidences that our system can operate in full 3D environments with a high density of obstacles. In this trial, four navigation targets are sent to the quadrotor sequentially, and 33 trajectories are generated in the flight with an average computing time of $37.79ms$. The total traversal distance of the quadrotor is $131.74m$. Photos of the environment and the quadrotor in flight are shown in Fig. 5.21. The four trajectories generated after receiving targets are shown in Fig. 5.20. We also perform more tests of fully autonomous flight which are not discussed in the paper but are included in the video, to validate the robustness of the proposed method.



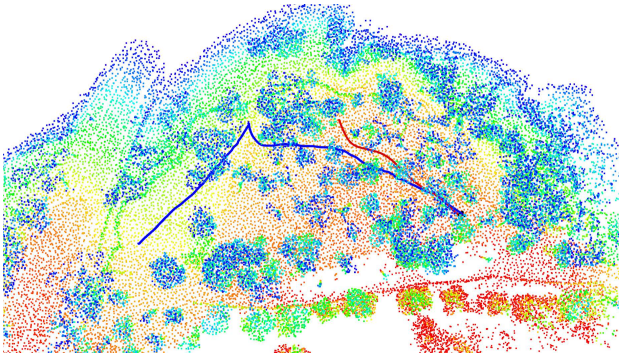
(a) Trajectory generated after the 1^{st} target received.



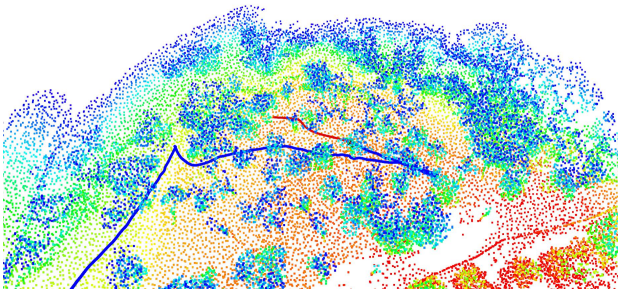
(b) Trajectory generated after the 2^{nd} target received.



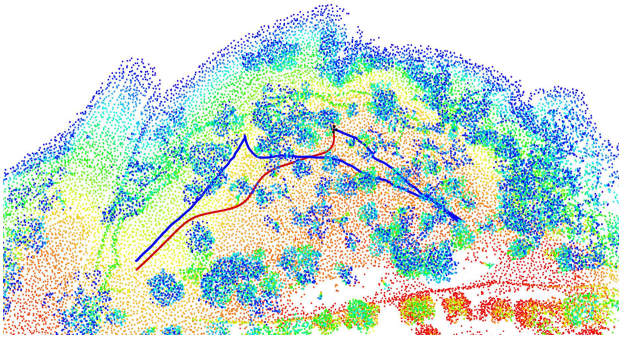
(c) Trajectory re-planed towards the 2^{nd} target.



(d) Trajectory generated after the 3^{rd} target received.

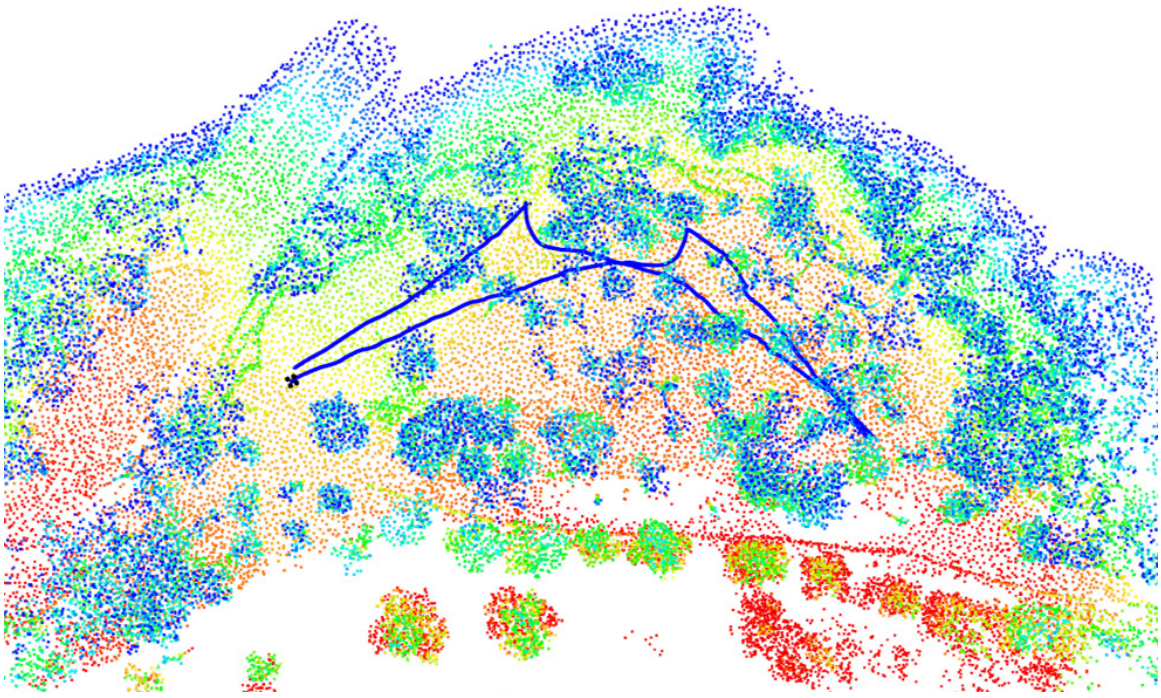


(e) Trajectory re-planed towards the 3^{rd} target.



(f) Trajectory generated after the 4^{th} target received.

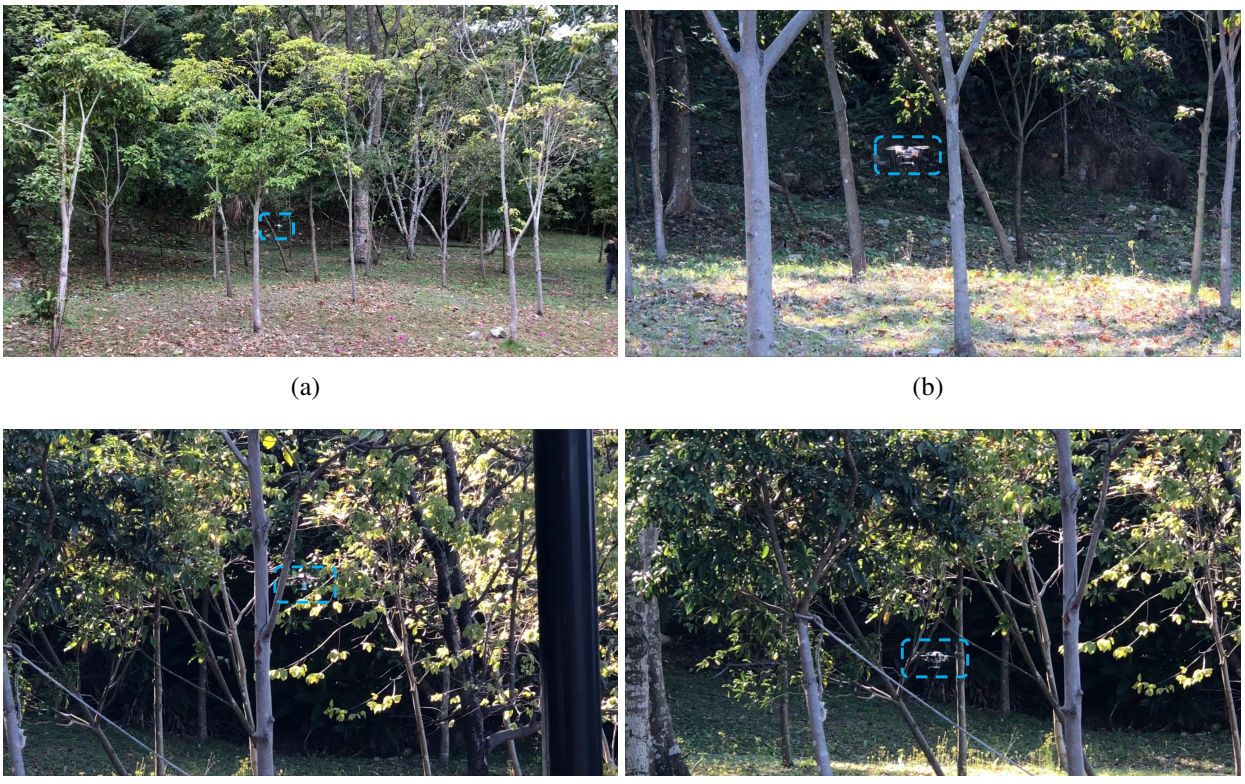
Figure 5.20. Trajectories generated in the outdoor autonomous flight tested in a complex forest. Markers are interpreted as the same in Fig. 5.16. Four navigating targets are given sequentially in this experiment. Trajectories generated after receiving targets and visualized in Figs. 5.20(a), 5.20(b), 5.20(c) and 5.20(d). Two trajectories re-generated in the flight are shown in Figs. 5.20(e) and 5.20(f). For clear visualization, the flight corridor is not shown in these figures, and points above $5m$ of the point clouds are removed. The quadrotor mesh model is scaled-up for the same reason.



(a) Overview of the autonomous flight in the dense forest.

(b) Real scene of the forest.

Figure 5.21. Overview of the quadrotor flights in the dense forest. Color code of obstacles indicates height. The tracking path of the quadrotor is shown as the blue curve. For clear visualization, points above $5m$ of the point clouds are removed, and the quadrotor mesh model is scaled-up. The total traversal distance of the quadrotor is $131.74m$ in this test.



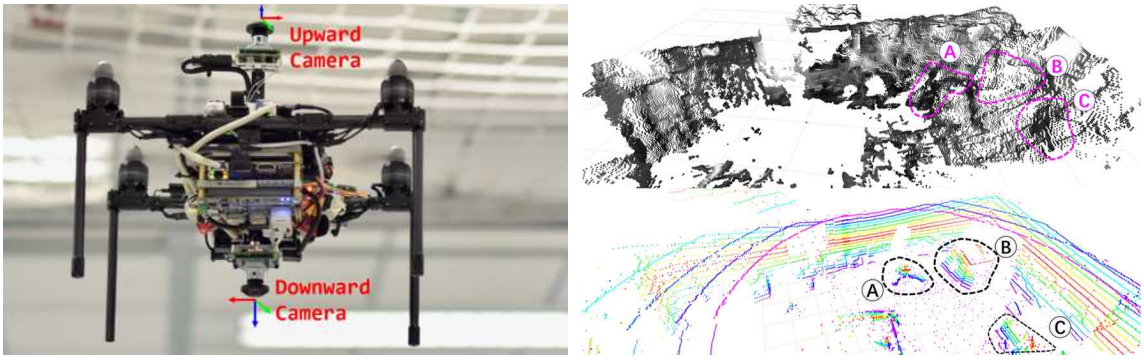
(a) (b)

(c) (d)

Figure 5.22. Snapshots of the autonomous flights in the unknown cluttered forest, corresponding to Fig. 5.20. The testing scene locates on a sloped hillside with an uneven ground to mimic the search-and-rescue mission in a full 3D unstructured, complex environments. Obstacles are at a very high density with various types and shapes, including trees, stones, and bushes. The information of the environment is previously unknown to our quadrotor, and all operations are done in onboard without human interventions.

5.5.7 test with a Degraded Sensor

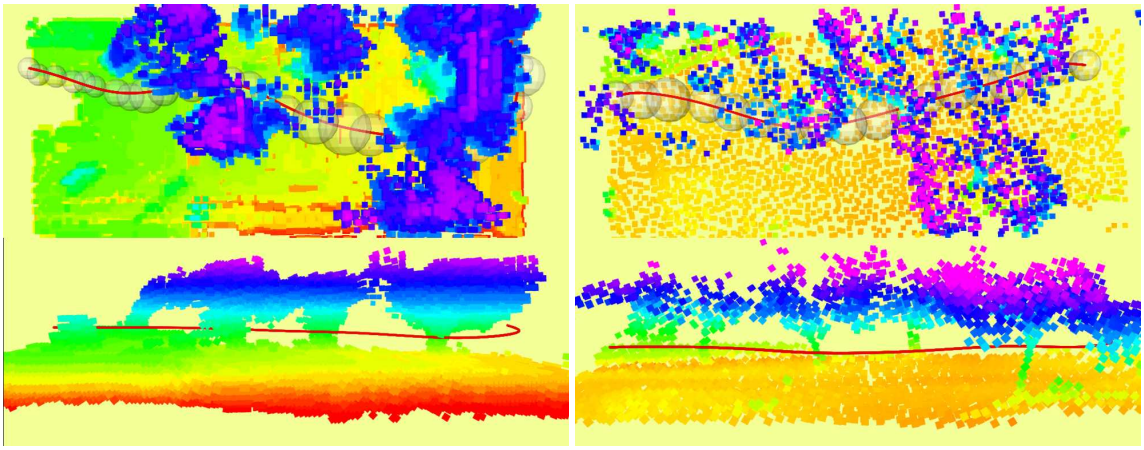
In the above sections, we present flight tests of a quadrotor equipped with a Velodyne VLP-16, which is a precise depth sensor. Here we validate that our proposed method is also robust enough to deal with relatively poorer depth measurements. We test our motion planning framework in a quadrotor equipped with the omnidirectional vision [33], as shown in Fig. 5.23. The quadrotor is equipped with dual fisheye cameras with one upward-facing and another downward facing. The localization is done by visual-inertial fusion [68] in the onboard CPU, and the depths of each pixel in images are estimated using stereo matching in an additional GPU.



(a) The dual fisheye quadrotor platform. (b) Comparing the depth with LiDAR measurements.

Figure 5.23. The dual fisheye quadrotor system in (a) and the comparison of the depth measurements against LiDAR in (b). In (b), A, B and C show corresponding objects.

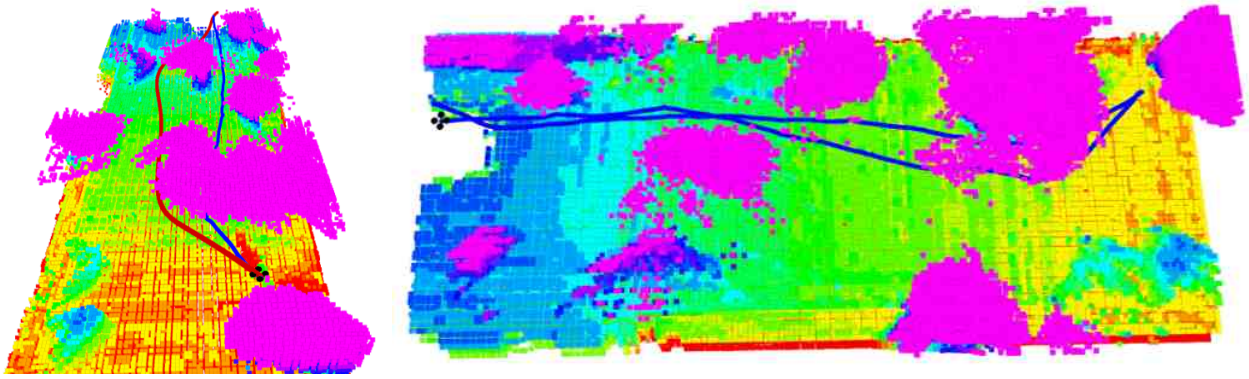
Compared to the above-mentioned LiDAR-based mapping system, the pixel-wise depth estimation by the fisheye cameras is significantly poorer and has a nonnegligible latency. It takes a much longer time for converting the sensor input to a global map. Four pairs of stereo images with resolution 409×260 from the front, rear, left and right directions are processed at the same time, and then the estimated depth measurements are fused into spatially hashed voxels using truncated signed distance field (TSDF) fusion [41]. Similar to point clouds, the spatially hashed voxels output by TSDF fusion are also unordered, and each voxel can be treated as a point in point clouds. Therefore, our proposed planning method is particularly suitable for this mapping system, and it bypasses the building of another post-processed map structure, such as the grid map.



(a) Planning with the dual fisheye cameras. (b) Planning with the LiDAR.

Figure 5.24. comparison of the flight corridor and the trajectory generated using the dual fisheye cameras in (a), and using the Velodyne LiDAR in (b). Outliers in the vision-based mapping system occupy much free space and result in a longer flight corridor and trajectory.

The overall latency from the images input to the spatially hashed voxels output is around $300 \sim 400ms$, while our LiDAR-based system only requires less than $100ms$ to update the globally registered point cloud. Also, the sensing range of the vision-based navigation system is very limited at around $4 \sim 5m$, because of the onboard computational limitation and the resolution of images. Therefore in the experiment, we set a relatively low speed ($0.8m/s$) for the quadrotor. Another issue is accuracy. The dual fisheye system builds a map denser than by the LiDAR, as shown in Fig. 5.24, because for every pixel, the depth is calculated. However, it generates a much more noisy result with many false-positive depth estimations. In Fig. 5.24, outliers in the depth estimation occupy much free space in the environment, thus reduces the feasible space for finding the flight corridor. Therefore, the planner usually generates a longer trajectory to pass through all obstacles when it uses the camera rather than the LiDAR ($35.4m$ $vs.$ $29.7m$). The result of the autonomous flight is presented in Fig. 5.25 and more details are included in the video.



(a) Trajectory generated in flight.

(b) Overview of the quadrotor path.

Figure 5.25. Autonomous flight test with an vision-based quadrotor. The trajectory generated for the 2^{nd} navigation is shown in (a), and the overview of the experiment is shown in (b). Markers are interpreted the same as in Fig. 5.20. Online visualization is included in the video.

5.6 Conclusion and Future Work

5.6.1 Conclusion

In this paper, we propose a motion planning framework for online trajectory generation for quadrotor navigation with full autonomy in unknown cluttered environments. We first develop an au-

[11] https://developer.nvidia.com/embedded/buy/jetson-tx2

tonomous quadrotor platform equipped with a 3-D LiDAR and an IMU to localize itself and depict its surroundings. Then we propose a method that directly operates on globally registered point clouds raw data, without building post-processed maps to find a collision-free flight corridor. Next, an optimization-based trajectory generation method with the guarantee of smoothness, safety, and dynamical feasibility is proposed to generate a trajectory entirely within the flight corridor. Finally, we integrate our proposed motion planning method with state estimation, mapping, and quadrotor control into the quadrotor platform. All processing is run onboard and in real-time. We validate the efficiency and robustness of our proposed method by comparing it with benchmark methods and testing in real-world experiments. Extensive field experiments are conducted in various types of environments, such as indoor environment, human-made structured scene, natural unstructured indoor-outdoor environment, and dense, uneven forest. These experiments validate that our proposed method, as well as the fully autonomous quadrotor system, is applicable to use in complex environments that may be inaccessible or dangerous for human beings. This capability is crucial for search-and-rescue missions where no functional facilities can be relied on, or human operators face danger.

5.6.2 Future Work

In the future, we aim to develop an autonomous exploration system based on our current work, to achieve large-scale unsupervised inspection without pre-specified navigation targets. In this way, after the exploration, maps of unknown environments can be autonomously built, and during the investigation, our planning framework can be used to avoid possible collisions.

Since our proposed method directly operates on point clouds raw data without map maintenance, the proposed motion planning is naturally plug-in-and-use on many other platforms which produce point clouds. In fact, we have already applied our previous method from [29] to a vision-based quadrotor [92] and ground vehicle [76] in various types of applications. In the future, we are going to implement our proposed method, which is an advance on our previous method, on other platforms and applications.

CHAPTER 6

OPTIMAL TIME ALLOCATION FOR QUADROTOR TRAJECTORY GENERATION

6.1 Research Background and Motivation

Up to now, we have introduced several methods for generating safe and dynamic feasible trajectories for MAV navigating in complex environments. These methods majorly focus on optimizing (wrapping) the trajectory in $R(3)$ space to avoid obstacles, while at the same time, adding extra conditions to ensure the kinodynamic feasibility. In these methods, temporal optimality is not the first concern, and time allocation is obtained by heuristic at the most time. In this chapter, instead, we look into the time parametrization of the MAV planning and seek for optimizing the time profile of an aerial trajectory.

For quadrotor motion planning and trajectory generation, the piecewise polynomial-based trajectory has been widely adopted since [55] and [71], because of its superior representative capability and concise formulation. Although variants of piecewise polynomial trajectory have been proposed, time allocation is still a bottleneck for it. We can use the scenario of a car/quadrotor racing in a sharp turn for illustration. No matter how high is the speed before entering the sharp turn, the acceleration of the vehicle has to be bounded while turning. In this case, a poor time allocation would take a significantly longer time for passing the sharp turn. In this paper, we present a framework to generate minimum time trajectories under the constraints of physical limits. The proposed framework is decoupled into two stages. Firstly we generate trajectories in the spatial aspect. Instead of parameterizing the trajectory directly by time, we generate time-independent trajectories in a virtual domain, with fixed geometric distributions. Secondly, we bridge the spatial trajectory to the temporal information. The relations between the virtual variable to time variable is found by minimizing the total flight time globally considering the dynamical limits. Therefore vehicles with our proposed time allocation method can pass the sharp turn with minimal time; related tests are given in Sect. 6.4.3.

This work is motivated by the observation that for a robotics trajectory generation application, most of the cases, the geometrical and temporal information are not necessarily coupled. For safety consideration, the trajectory should be wrapped to avoid obstacles [70] [58] or be bounded within free space [15] [32]. To this end, the geometrical properties of the trajectory are the only concern. On the other side, for dynamical feasibility consideration, temporal information of the trajectory, such as velocity and acceleration of the quadrotor should be bounded within the kinodynamic limits. In many works, although the kinodynamic is constrained at the same time with the generation of the geometrical trajectory [32] [10], it is highly dependent on the time allocation and is often too conservative to fully utilize the actuators for high-speed navigations. In this paper, we propose a method to get the optimal time allocation for quadrotor trajectory generation.

6.2 Safe Spatial Trajectory Generation

We decouple the MAV planning problem into spatial and temporal two different layers. The safe trajectory generated in the spatial layer is the foundation of the following time optimization. In this section, we briefly review several representative methods for generating safe spatial trajectories, as stated before in Chaps. 3 4 5.

6.2.1 Corridor-Based Method

As presented in Chaps. 4 5, we firstly extract the free space in environments to form a flight corridor consisting of convex shapes and then constrain piecewise Bézier curve entirely within these convex shapes. The Bézier curve consists of Bernstein polynomial basis with respect to a virtual parameter s is written as:

$$B_j(s) = c_j^0 b_n^0(s) + \ldots + c_j^n b_n^n(s) = \sum_{i=0}^{n} c_j^i b_n^i(s), \tag{6.1}$$

where $[c_j^0, c_j^1, ..., c_j^n]$ is the set of control points of the j^{th} piece of the Bézier curve. $b(n)$ is the Bernstein polynomial basis, n is the degree. Suppose we denote the piecewise Bézier curve in one dimension μ out of *x, y, z* as $f_\mu(s)$. In our applications, since we only want to fix the geometrical shape of the trajectory in s domain, we simply set the duration of s in each piece of the trajectory

as 1. For generating a minimal jerk curve, the objective is written as:

$$J = \sum_{\mu}^{x,y,z} \int_0^{s(T)} \left(\frac{d^3 f_\mu(s)}{ds^3} \right)^2 ds, \tag{6.2}$$

where T is the total time of the trajectory and $s(T)$ is the corresponding s value. Note here although the minimum jerk objective can efficiently smooth the trajectory, it has a totally different physical meaning compared to the one in time domain. In practice, the minimal 3^{rd} order objective in s domain often results an un-necessary longer trajectory since we set $s \in [0, 1]$ for all pieces of the trajectory ignoring the length of the segment. To overcome this disadvantage, we add a minimal arc length regularization term into this objective. The arc length of a given curve $f_\mu(s)$ is:

$$L = \int_0^{s(T)} \sqrt{f'_x(s)^2 + f'_y(s)^2 + f'_z(s)^2} ds, \tag{6.3}$$

which is a line integral and can not be written in a closed form. We instead use L^2 as the regularization term and the objective function is $w_s \cdot J + w_l \cdot L^2$, where w_s and w_l are relative weights for minimizing jerk in s domain and length of the curve. The detailed objective is in a quadratic formation and is omitted here for brevity.

$$\sum_{\mu}^{x,y,z} \int_0^{s(T)} \left(w_s \cdot \frac{d^3 f_\mu(s)}{ds^3}^2 + w_l \cdot \frac{df_\mu(s)}{ds}^2 \right) ds, \tag{6.4}$$

Utilizing the convex hull property, the entire curve can be bounded within the flight corridor by adding safety constraints on all control points. The trajectory generation problem is a quadratic program (QP), which can be solved in polynomial time. For details about the formulation and the application, we refer readers to our previous publication [32].

6.2.2 Gradient-Based Method

For another category of the trajectory generation, as in Chap 3, the trajectory is optimized as a nonlinear optimization which takes into account the smoothness, safety and dynamical feasibility of the trajectory. By using the spatial-temporal decoupling strategy, the dynamical feasibility terms can be dropped. Thus the objective function can be written as $f = w_s \cdot J + w_c \cdot C$, where J is exactly

the same as Equ.3. The collision cost C is the line integral of the arc length along the trajectory and can be discretized for numerical calculation. The collision cost C in s domain is:

$$\begin{aligned} C &= \int_0^{s(T)} c(p(s))\sqrt{f'_x(s)^2 + f'_y(s)^2 + f'_z(s)^2}ds \\ &= \sum_{k=0}^{N} c(p(s_k))\sqrt{f'_x(s_k)^2 + f'_y(s_k)^2 + f'_z(s_k)^2}\delta s, \end{aligned} \tag{6.5}$$

where $c(p)$ is the penalty function and is designed as an exponential function of distance in our previous work [28]. The trajectory generation problem is in a nonlinear optimization formulation. By using numerical optimization techniques such as Levenberg-Marquardt method, the nonlinear optimization results in a smooth and collision-free trajectory. Note here that the dropping of dynamical feasibility terms significantly reduces computation time and improves the convergence rate, which is detailed in 6.4.1.

6.3 Minimum-Time Temporal Trajectory Generation

6.3.1 Representing the Temporal Trajectory

For a piecewise spatial trajectory defined in virtual domain s, the geometric properties such as the x, y, z positions and their derivatives are fixed. However, the kinodynamic properties of the quadrotor such as the velocity and acceleration are decided by a piecewise mapping function which maps each s value to a time value t. In this paper, we name the trajectory of velocity and acceleration in the time domain as temporal trajectory. Our motivation in this section is to design a mapping function which can generate motion as fast as possible without violating the kinodynamic limits of the quadrotor. Suppose we have already solved the spatial trajectory $f_\mu(s)$ and denote the relation between the variable s of the trajectory to time t as a function $s(t)$. Obviously, $s(t)$ is also a piecewise function which can be written as follows:

$$s(t) = \begin{cases} s_0(t), & s_1(0) = 0, s_0(T_0) = 1, s_0 \in [0,1], \\ s_1(t), & s_1(0) = 0, s_1(T_1) = 1, s_1 \in [0,1], \\ \vdots & \vdots \\ s_m(t), & s_m(0) = 0, s_m(T_m) = 1, s_m \in [0,1], \end{cases} \tag{6.6}$$

where $T_0, T_1, ...T_m$ are unknown time durations in each pieces of the temporal trajectory.

Another obvious observation is that $s(t)$ must be a monotonically increasing function since physically variable time can only increase. Therefore the following condition holds:

$$\dot{s}(t) \geq 0. \tag{6.7}$$

By chain rule, we can re-write the temporal trajectory, i.e. the velocity and acceleration of the quadrotor as:

$$\begin{aligned} v_\mu(t) &= \dot{f}_\mu(s) = f'_\mu(s) \cdot \dot{s}, \\ a_\mu(t) &= \ddot{f}_\mu(s) = f'_\mu(s) \cdot \ddot{s} + f''_\mu(s) \cdot \dot{s}^2. \end{aligned} \tag{6.8}$$

Here we use the notation $\dot{c} = dc/dt, \ddot{c} = d^2c/dt^2$ to indicate taking the 1^{st} and 2^{nd} order derivative of a variable c with respect to time, and use $c' = dc/ds, c'' = d^2c/ds^2$ for derivatives with respect to s.

6.3.2 Piecewise Minimum-Time Optimization

Objective

The objective J of the temporal trajectory generation is to minimize the total time, which is written as:

$$J = T = \int_0^T 1 dt. \tag{6.9}$$

With the fact $\dot{s} = ds/dt$ and in each piece of the trajectory $s_i(t) \in [0, 1]$, the time T can be re-written as:

$$T = \sum_{i=0}^{m} \int_0^1 \frac{1}{\dot{s}_i} ds. \tag{6.10}$$

In minimum-time optimal control problem [84], the energy of the control input is penalized in the objective function to trade-off the time optimality and control extremeness. In our problem, we can also incorporate the regularization on changing rate of s in our objective:

$$J = \sum_{i=0}^{m} \int_0^1 \Big(\frac{1}{\dot{s}_i} + \rho \cdot \ddot{s}_i{}^2 \Big) ds, \tag{6.11}$$

where ρ is the weighting parameter. Following the direct transcription method in [85], we introduce two additional functions $a(s)$ and $b(s)$ which are also piecewise and satisfy:

$$a(s) = \ddot{s}, \tag{6.12}$$

$$b(s) = \dot{s}^2. \tag{6.13}$$

Then the objective is derived as to minimize:

$$J = \sum_{i=0}^{m} \int_0^1 \Big(\frac{1}{\sqrt{b_i(s)}} + \rho \cdot a_i(s)^2 \Big) ds, \tag{6.14}$$

where the subscript i of $a_i(s)$ and $b_i(s)$ shares the same meaning as in s_i. Several observations obviously hold following the formulation above, including:

$$b_i(s) \geq 0, \tag{6.15}$$

$$b_i'(s) = 2 \cdot a_i(s). \tag{6.16}$$

Continuity Constraints

The generated spatial-temporal trajectory must be continuous at several order derivatives to meet the requirements in quadrotor control. According to our experience and the actuator model of the quadrotor, we require the continuity up to acceleration holds between each pieces of the trajectory.

The continuity of position is achieved by the spatial trajectory in Sec. 6.2. For velocity and acceleration, the continuities are enforced by setting equality constraints on the joint point between two consecutive pieces of the temporal trajectory, for the i^{th} and $(i+1)^{th}$ piece trajectories, in dimension $\mu \in \{x, y, z\}$ we have:

$$f_{i,\mu}'(1) \cdot \sqrt{b_i(1)} = f_{i+1,\mu}'(0) \cdot \sqrt{b_{i+1}(0)}, \tag{6.17}$$

$$\begin{aligned} & f_{i,\mu}'(1) \cdot a_i(1) + f_{i,\mu}''(1) \cdot b_i(1) \\ =& f_{i+1,\mu}'(0) \cdot a_{i+1}(0) + f_{i+1,\mu}''(0) \cdot b_{i+1}(0) \end{aligned} \tag{6.18}$$

Kinodynamic Constraints

To ensure the kinodynamic feasibility, we enforce constraints on velocity and acceleration in each piece of the trajectory. Constraints hold in $s \in [0, 1]$ at x, y, z dimensions are written as:

$$-v_{max} \leq f'_{i,\mu}(s) \cdot \sqrt{b(s)} \leq v_{max}, \tag{6.19}$$

$$-a_{max} \leq f'_{i,\mu}(s) \cdot a(s) + f''_{i,\mu}(s) \cdot b(s) \leq a_{max}, \tag{6.20}$$

where v_{max} and a_{max} are the kinodynamic limits of the quadrotor decided by the actuator models.

Boundary Constraints

In practice, the trajectory is generated from the initial state of the quadrotor to a final state (normally but not necessarily zero state). For the temporal trajectory, the boundary condition is to meet the initial velocity and acceleration a^0, v^0, and the terminal velocity and acceleration a^f, v^f. The constraints are:

$$f'_{0,\mu}(0) \cdot \sqrt{b_0(0)} = v^0_\mu, \tag{6.21}$$

$$f'_{m,\mu}(1) \cdot \sqrt{b_m(1)} = v^f_\mu, \tag{6.22}$$

$$f'_{0,\mu}(0) \cdot a_0(0) + f''_{0,\mu}(0) \cdot b_0(0) = a^0_\mu, \tag{6.23}$$

$$f'_{m,\mu}(1) \cdot a_m(1) + f''_{m,\mu}(1) \cdot b_m(1) = a^f_\mu, \tag{6.24}$$

Note that if the initial and final states are both static, there must exist a solution satisfying the boundary constraints. Otherwise, the optimization program may be infeasible.

6.3.3 Convex SOCP Reformulation

To make the above optimization problem easily solvable, we would like to re-formulate the objective and constraints under the disciplined convex formalism. To this end, the key step is to discretize the virtual parameter s into grids [85] for each piece of the trajectory. Then $a(s)$ is regarded as piecewise constant in each grid of s and therefore $b(s)$ is piecewise linear according to Equ. 6.16. For each piece of the trajectory, $s_i \in [0, 1]$ is discretized to s_i^k, where $k = 0, 1, ..., K$, according to a given number K. We have $s_i^k - s_i^{k-1} = \Delta s = 1/K$. For each piece, $a_i(s)$ and $b_i(s)$ are modeled by introducing discrete a_i^k and b_i^k, for which b_i^k is assigned at each s_i^k and a_i^k is

assigned at the middle of s_i^k and s_i^{k+1}. Then $b(s)$ is written as piecewise linear function:

$$b(s) = b_i^k + \frac{b_i^{k+1} - b_i^k}{\Delta s} \cdot (s - s_i^k). \tag{6.25}$$

According to Equs. 6.15 and 6.16, we have

$$b_i^k \geq 0, \tag{6.26}$$

$$b_i^{k+1} - b_i^k = 2 \cdot \Delta s \cdot a_i^k. \tag{6.27}$$

The objective function in Equ. 6.14 is derived as:

$$J = \sum_{i=0}^{m} \sum_{k=0}^{K-1} \left(\frac{2}{\sqrt{b_i^{k+1}} + \sqrt{b_i^k}} + \rho \cdot {a_i^k}^2 \right) \cdot \Delta s, \tag{6.28}$$

which is equivalent to the quadratic function:

$$\sum_{i=0}^{m} \sum_{k=0}^{K-1} \left(2 \cdot d_i^k + \rho \cdot {a_i^k}^2 \right) \cdot \Delta s, \tag{6.29}$$

by introducing slack variables d_i^k, plus additional constraints:

$$\frac{1}{\sqrt{b_i^{k+1}} + \sqrt{b_i^k}} \leq d_i^k, \;\; k = 0, ...K-1, i = 0, ...m. \tag{6.30}$$

Equ. 6.30 are transformed to quadratic forms as

$$\frac{1}{c_i^{k+1} + c_i^k} \leq d_i^k, \qquad k = 0, ...K-1, i = 0, ...m. \tag{6.31}$$

$$c_i^k \leq \sqrt{b_i^k}, \qquad k = 0, ...K, i = 0, ...m. \tag{6.32}$$

by introducing slack variables c_i^k in each segment.

Equ. 6.31 is equivalent to the typical formulation of rotated quadratic cones:

$$2 \cdot d_i^k \cdot \left(c_i^{k+1} + c_i^k \right) \geq \sqrt{2}^2, \tag{6.33}$$

which is shorthanded in

$$(d_i^k, c_i^{k+1} + c_i^k, \sqrt{2}) \in Q_r^3. \tag{6.34}$$

And Equ. 6.32 can be written in the typical formulation of quadratic cones:

$$\left(b_i^k + 1\right)^2 \geq \left(b_i^k - 1\right)^2 + \left(2 \cdot c_i^k\right)^2, \tag{6.35}$$

and be shorthanded as

$$(b_i^k + 1, b_i^k - 1, 2c_i^k) \in Q^3. \tag{6.36}$$

Finally, an additional slack variable t is added to convert the quadratic objective in Equ. 6.29 to an affine form as

$$\sum_{i=0}^{m} \sum_{k=0}^{K-1} 2\Delta s \cdot d_i^k + \rho \cdot \Delta s \cdot t, \tag{6.37}$$

with a rotated quadratic cone

$$2 \cdot t \cdot 1 \geq \sum_{i=0}^{m} \sum_{k=0}^{K-1} (a_i^k)^2, \tag{6.38}$$

i.e.

$$(t, 1, \mathbf{a}) \in Q_r^{2+K\cdot(m+1)}, \tag{6.39}$$

where $\mathbf{a} = [a_0^0, a_0^1, ..., a_m^{K-1}]$ is the vectorized variable consisting of all a_i^k assigned to each segment.

Up to now, the original objective function in Equ. 6.9 is converted to an affine function (Equ. 6.37) with several induced rotated second-order cones (Equ. 6.34 and 6.39) and second-order cones (Equs. 6.36). Back to constraints in the original problem, the kinodynamic constraints (Sec. 6.3.2) are written in discrete form as:

$$-v_{max} \leq f'_{i,\mu}(s_i^k) \cdot \sqrt{b_i^k} \leq v_{max}, k = 0, ...K. \tag{6.40}$$

$$-a_{max} \leq f'_{i,\mu}(s_i^{k+1/2}) \cdot a_i^k + f''_{i,\mu}(s_i^{k+1/2}) \cdot b_i^{k+1/2} \leq a_{max}, k = 0, ...K-1 \tag{6.41}$$

where $s_i^{k+1/2} = (s_i^k + s_i^{k+1})/2$ and $b_i^{k+1/2} = (b_i^k + b_i^{k+1})/2$ corresponding to a_i^k, for $i = 0, ..., m$. Equation 6.41 is in affine and Equation 6.40 can also be re-written as affine:

$$f'^2_{i,\mu}(s_i^k) \cdot b_i^k \leq v_{max}^2, \tag{6.42}$$

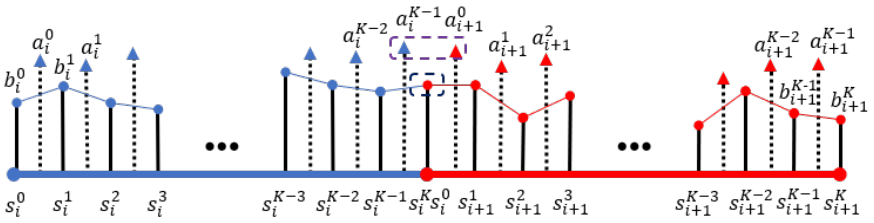


Figure 6.1. b^k and a^k assigned in the i^{th} (in blue) and $(i+1)^{th}$ (in red) pieces of the temporal trajectory corresponding to . b^k is indicated by circles while a^k is indicated by triangle. The continuity constraint in velocity is enforced at the joint point between two segments and is shown in the black rectangle. And the constraint on acceleration is enforced in the purple rectangle between a_i^{K-1} and a_{i+1}^0.

Continuity constraints (Sec. 6.3.2) on velocity are written in discrete form as:

$$f'_{i,\mu}(s_i^k) \cdot \sqrt{b_i^K} = f'_{i+1,\mu}(s_{i+1}^0) \cdot \sqrt{b_{i+1}^0}. \tag{6.43}$$

Since the spatial trajectory is continuous at the joint position, which means $f'_{i,\mu}(s^k) = f'_{i+1,\mu}(s^0)$. Equation. 6.43 is equivalent to

$$b_i^K = b_{i+1}^0. \tag{6.44}$$

Unlike b_i^k is evaluated exactly at s_i^k, a_i^k is evaluated at $(s_i^k + s_i^{k+1})/2$, so no a_i^k is assigned exactly at the joint position of each two pieces of the temporal trajectory. Instead, the last element a_i^{K-1} in the i^{th} segment is separated by Δs with the first element a_{i+1}^0 in the $(i+1)^{th}$ segment, as shown in Fig. 6.1. Thus we set in-equality constraints to bound the transitions of accelerations between every two segments. Besides, although we assume a^k is piecewise constant, we would like to bound the changing rate of acceleration considering the aggressiveness in control. We write the affine in-equality constraints for bounding the transitions of acceleration whether within one segment or across two segments:

$$\begin{aligned} -\lambda_a \leq & f'_{i,\mu}(s_i^{k+1/2}) \cdot a_i^k + f''_{i,\mu}(s_i^{k+1/2}) \cdot b_i^{k+1/2} - \\ & f'_{i,\mu}(s_i^{k+3/2}) \cdot a_i^{k+1} - f''_{i,\mu}(s_i^{k+3/2}) \cdot b_i^{k+3/2} \leq \lambda_a, \end{aligned} \tag{6.45}$$

$$\begin{aligned} -\lambda_a \leq & f'_{i,\mu}(s_i^{0+1/2}) \cdot a_i^0 + f''_{i,\mu}(s_i^{0+1/2}) \cdot b_i^{0+1/2} - \\ & f'_{i-1,\mu}(s_{i-1}^{K-1/2}) \cdot a_{i-1}^{K-1} - f''_{i-1,\mu}(s_{i-1}^{K-1/2}) \cdot b_{i-1}^{K-1/2} \leq \lambda_a, \end{aligned} \tag{6.46}$$

where λ_a is a given parameter to limit the changing rate of acceleration along the temporal trajec-

tory. Note here λ_a is not jerk, since time difference between s_i^k and s_{i+1}^k cannot be determined in the optimization.

For boundary constraints, velocity constraints are applied directly as:

$$f'_{0,\mu}(0) \cdot \sqrt{b_0^0} = v_\mu^0, \tag{6.47}$$

$$f'_{m,\mu}(1) \cdot \sqrt{b_m^k} = v_\mu^f. \tag{6.48}$$

For acceleration's boundary conditions, we set in-equality constraints for the same reason as in Equs. 6.45 and 6.46.

$$-\lambda_a \leq f'_{0,\mu}(s_0^{1/2}) \cdot a_0^0 + f''_{0,\mu}(s_0^{1/2}) \cdot b_0^{1/2} - a_\mu^0 \leq \lambda_a, \tag{6.49}$$

$$\begin{aligned} -\lambda_a \leq & f'_{m,\mu}(s_m^{K-1/2}) \cdot a_m^{K-1} + \\ & f''_{m,\mu}(s_m^{K-1/2}) \cdot b_m^{K-1/2} - a_\mu^f \leq \lambda_a. \end{aligned} \tag{6.50}$$

Up to now, the objective is written in affine ($\mathbf{h}^T\mathbf{d} + \rho \cdot t$) alongside with second order cones. In-equality and equality constraints are abstracted and written in affine ($\mathbf{A}_{eq} \cdot \mathbf{x}$ and $\mathbf{A}_{eq} \cdot \mathbf{x} = \mathbf{b}_{eq}$), where $\mathbf{d}$ consists of all variables d^k and $\mathbf{x}$ vectorizes all optimizated variables a^k, b^k, c^k, d^k. We can write the temporal trajectory generation problem as:

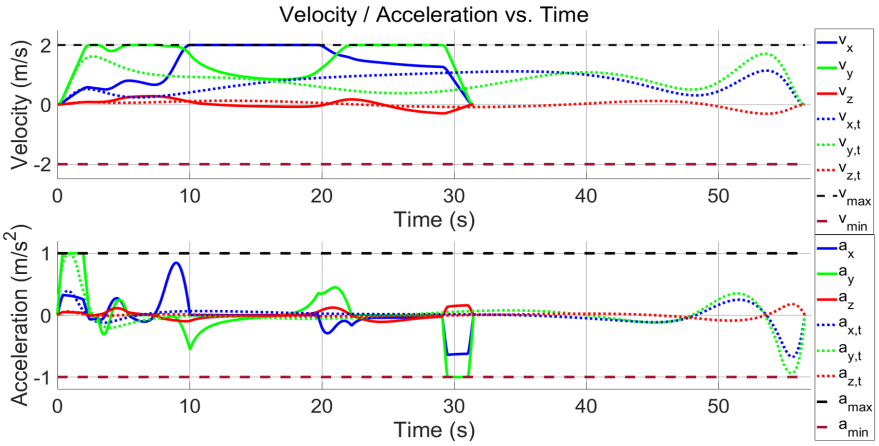
$$\begin{aligned} \min \quad & \mathbf{h}^T\mathbf{d} + \rho \cdot t, \\ \text{s.t.} \quad & \mathbf{A}_{eq} \cdot \mathbf{x} = \mathbf{b}_{eq}, \\ & \mathbf{A}_{ie} \cdot \mathbf{x} \leq \mathbf{b}_{ie}, \\ & (t, 1, \mathbf{a}) \in Q_r^{2+K\cdot(m+1)}, \\ & (d_i^k, c_i^{k+1} + c_i^k, \sqrt{2}) \in Q_r^3, k = 0, 1, ..., K-1, \\ & (b_i^k + 1, b_i^k - 1, 2c_i^k) \in Q^3, k = 0, 1, ..., K, \end{aligned} \tag{6.51}$$

which is a typical convex Second Order Cone Program (SOCP) and can be solved in polynomial time by off-the shelf convex solver.

6.4 Results

6.4.1 Comparisons with Time-parameterized trajectory

We present the comparisons between the proposed time optimization method in this paper to our previous work [32], in which the kinodynamic limits are enforced based on given time allocations. In the comparison w_s and w_l are set as 1.0 and 2.0. The velocity and acceleration limits are $\pm 2m/s$ and $\pm 1m/s^2$. Results are given in Fig. 6.2. We can see from the results that our proposed method gives a much shorter time for finishing the trajectory respecting the kinodynamic limits. In the test, the optimal time calculated by the proposed method in this paper is $31.47s$ and the time based on the heuristic in our previous work is $56.52s$, respectively. The generated velocities and accelerations approach the limits as much as possible by using our proposed approach. In our previous method [32], although trajectories generated based on a given time allocation can be entirely constrained within the physical bound, it is almost always no possibility to fully utilize the quadrotor's actuators to do a full-speed cruise. In our previous work, the constraints on kinodynamic feasibility are met easily when one of the segments of the trajectory has reached its limit. Moreover, if an over-aggressiveness time allocation is given, the trajectory generation problem is infeasible at all. While in this paper, we optimize the time allocation globally, and the fastest motion concerning the kinodynamical bounds are finally obtained.



(a) Velocities and accelerations against time.

Figure 6.2. Comparisons of the proposed method against our previous method [32]. Trajectories are generated in a random forest with target randomly selected. Trajectories using previous method are in blue curves and trajectories with time optimization are in green curves. Resulted velocities and accelerations against time are given in Fig. 6.2(a).

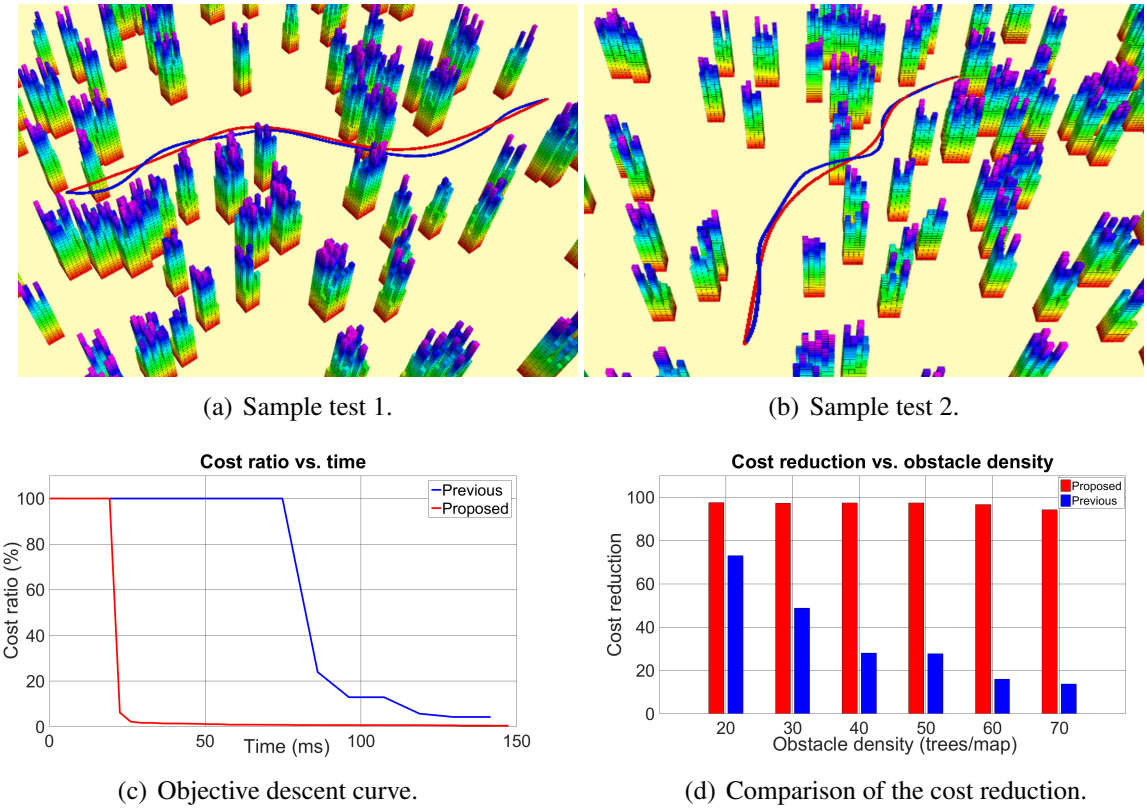
Decoupling the trajectory generation problem into spatial and temporal layers not only achieves time optimality but also benefits the optimization procedure in the solely spatial aspect. Here we present numerical tests by using our previous gradient-based method [28] to show the differences with/without the spatial-temporal decoupling. Under the proposed framework, the dynamical feasibility cost term is removed from the nonlinear optimization, which makes convergence significantly improved. We do 500 random tests in each different obstacle density. As shown in Fig. 6.3, by separating the temporal information, the objective value converges much faster, and the resulted trajectories are not only smoother but also have higher clearance. We also compare the cost reduction, which is defined as the dropping ratio of the objective from the beginning of the nonlinear optimization, in a given time $50ms$. As in Fig. 6.3(d), by separating the temporal trajectory, the objective converges to much lower values in all obstacle densities quickly, while the time-dependent trajectory converges much slower, especially when the obstacle density is high. The reason is that the removal of the dynamic feasibility cost terms saves much time of evaluation in each iteration, and also reduces the non-convexity of the objective function.

6.4.2 Study on Parameter Settings

Here we show the results of optimizing the temporal trajectory with different parameter settings. Two key parameters affecting the performance most are the control weighting coefficients ρ which regularizes $\ddot{s}$, and K which decides the resolution of $b(s)$ and $a(s)$.

Varying ρ

The weighting coefficient ρ of the control energy regularization term balances the expectation of minimal time and aggressiveness of control effort. Results of applying different ρ values on a fixed spatial trajectory with $K = 50$ are given in Figs. 6.4(a) to 6.4(d). As is shown in the result, when $\rho = 0$, the resulted temporal trajectory has the most aggressiveness. There are peaks in acceleration and sharp transitions in velocity. As ρ increases, the acceleration and velocity become smoother, and all sharp transitions are eliminated, as a cost, the total time of the trajectory is longer.



(a) Sample test 1.

(b) Sample test 2.

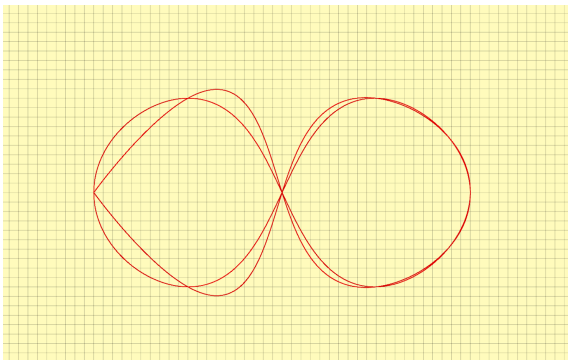
(c) Objective descent curve.

(d) Comparison of the cost reduction.

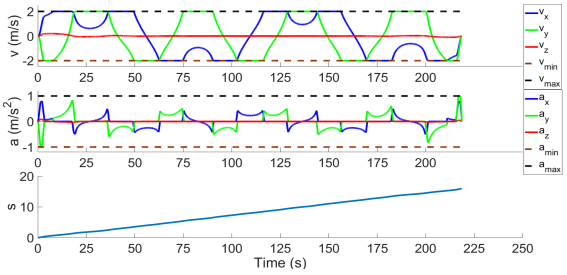
Figure 6.3. Comparisons of the generated trajectories w\ and w\o temporal optimization. Trajectories are generated in a random forest with targets randomly selected as in Figs. 6.3(a), 6.3(b). The comparison of the objective dropping is in 6.3(c), which corresponds to the instance in 6.3(b). The trajectory with spatial-temporal decoupling is in red curve and the other is in blue curve. The result of the reduction of final objective cost is given in Fig. 6.3(d).

Varying K

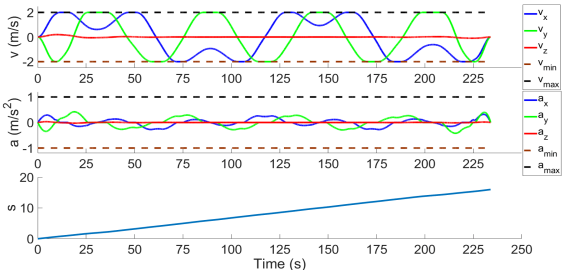
In our approach, the parameter K directly decides the number of grids in $s \in [0, 1]$. Here we present results of optimizing the temporal trajectory with different K values with $\rho = 10$. As in Fig. 6.4, increasing K improves the optimality of the final solution significantly when K is small. However, when K is large, the improvement is not very notable. The number of K can be viewed as the resolution of the optimal solution. As more grids in the s domain are divided, more freedom (decision variables) is provided to solve the problem with the cost that the computational complexity increases. According to our experience, the computational time of the SOCP increases linearly with K. Moreover, we suggest to set K in $30 \sim 50$ for normal applications.



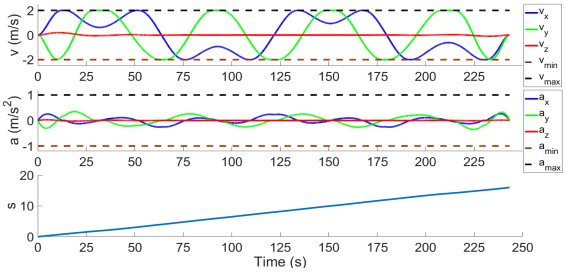
(a) Spatial trajectory.



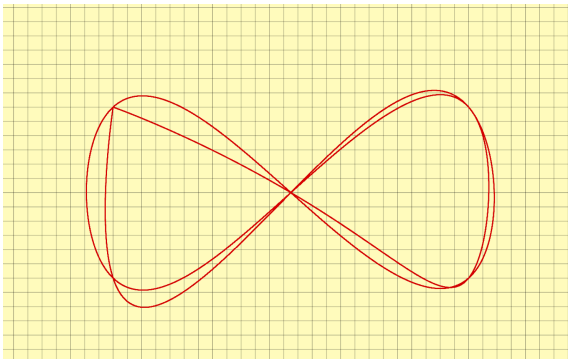
(b) $\rho = 0$



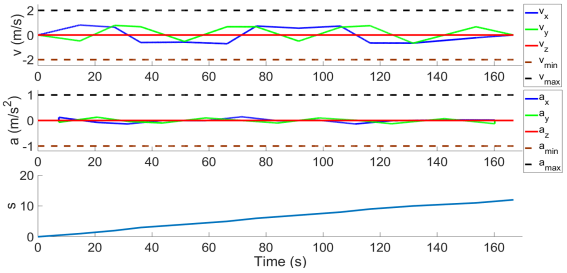
(c) $\rho = 1$



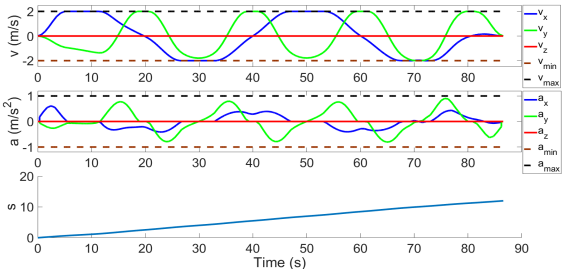
(d) $\rho = 10$



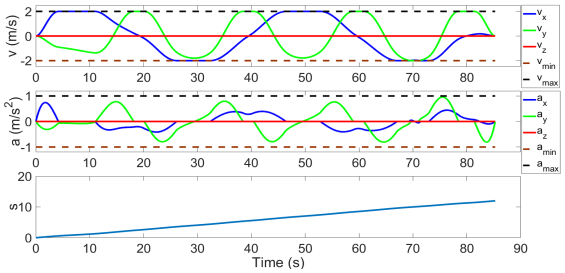
(e) Spatial trajectory.



(f) $K = 1$

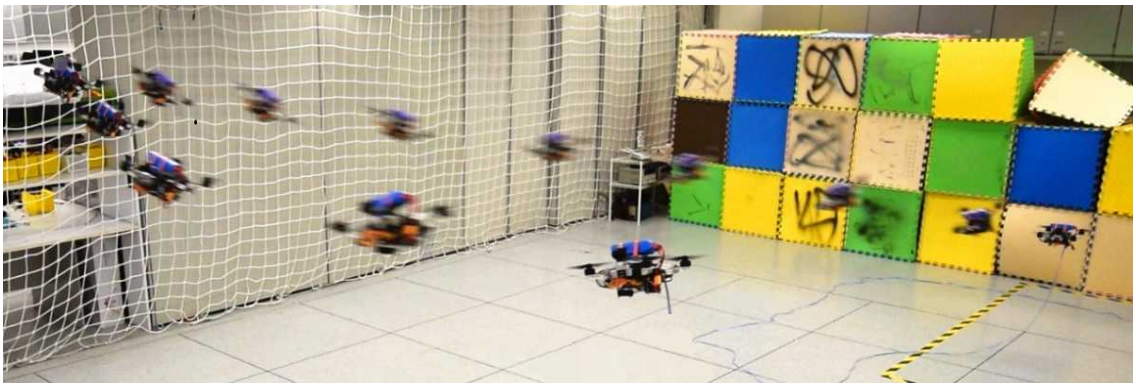


(g) $K = 10$



(h) $K = 80$

Figure 6.4. Comparisons of the proposed method with different control energy regularization weight ρ and with different discretization number K. In Figs. 6.4(a) $\sim$ 6.4(d), K fixes at 50 and ρ varies. Higher ρ results in a smoother acceleration profile with longer total time. In Figs. 6.4(e) $\sim$ 6.4(h), ρ fixes at 10 and K varies. Higher K means finer resolution of the final solution, which results in a shorter total time.



(a) Composite image of the indoor quadrotor flight.

(b) Composite image of the outdoor quadrotor flight.

Figure 6.5. Composite images of the quadrotor flying in indoor and outdoor experiments. Our quadrotor platform equipped with a monocular camera and an IMU along with an Intel i7 CPU. We demonstrate fast-speed flights by using our proposed method in both indoor and outdoor environments. The video recording experiments is available in the attachment of this paper.

6.4.3 On-board Flight Tests

The method proposed in this paper[1] is implemented in C++11 using a general convex solver Mosek[2]. The flight experiments are done on a self-developed quadrotor platform which is specially designed for high-speed flights. All processings are done on an onboard dual-core 2.40 GHz Intel i7-5500U processor, which has 8 GB RAM and 64 GB SSD. In the test, several waypoints are sent to the quadrotor, and a spatial trajectory in the virtual domain s with fixed geometric shape is generated onboard. The spatial trajectory is then be parameterized to time t by using our proposed time optimization method, and the quadrotor is commanded to track the trajectory. The procedure

[1]Source code of the proposed method will be released in `https://github.com/HKUST-Aerial-Robotics/TimeOptimizer` after the publishing of this paper.

[2]`https://www.mosek.com`

repeats in several loops. We assume our quadrotor has different kinodynamic limits and optimize the minimal-time temporal trajectory to approach such limits in each loop. In experiments, the kinodynamic limits are raised in each loop.

Indoor Test

We conduct indoor flights with the pose feed by the indoor localization system OptiTrack[3]. In this test, a series of waypoints are sent to the quadrotor to generate a spatial trajectory. Initially, the kinodynamic limits v_{max} and a_{max} are $1m/s$ and $0.5m/s^2$ in velocity and acceleration. And λ_a and ρ are set as 0.05 and 0. K is set as 30. The quadrotor repeats the tracking of the trajectory. In each loop, v_{max}, a_{max} and λ_a are increased by $1m/s$, $0.5m/s^2$ and 0.1 until velocity limit reaches $6m/s$ and acceleration limit reaches $6m/s^2$. The result presents the profiles of velocity and acceleration in the 12^{th} loop of the flight is given in Fig. 6.6. Average computing time in generating the optimal temporal trajectory is $62.41ms$.

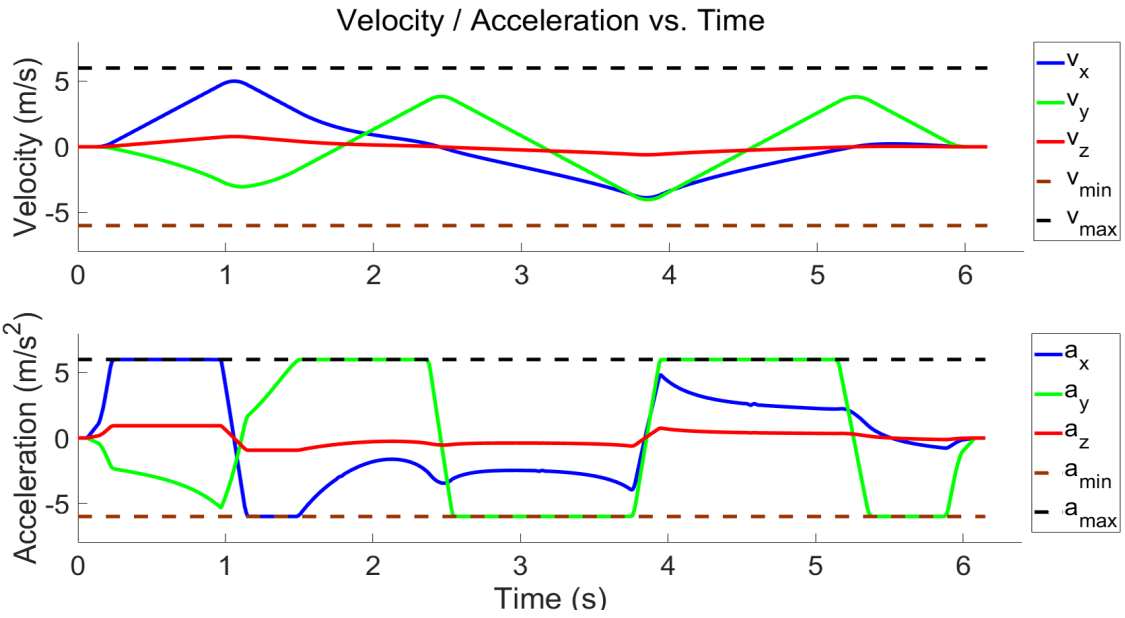


Figure 6.6. The result of a sample flight in the indoor experiment. In the flight the maximum velocity and acceleration allowed for the quadrotor are set as $6m/s$ and $6m/s^2$. The generated highest velocity and acceleration in one axis are $5.11m/s$ and $6m/s^2$.

[3] http://optitrack.com/

Outdoor Test

We use visual-inertial fusion [68] with the minimum sensing suite: one camera and one IMU, for pose feedback in outdoor experiments. The kinodynamic limits are increased by $1.5.0m/s$ and $2.0m/s^2$ from the initial value of $2.0m/s$ and $3.0m/s^2$. Other parameters are the same as indoor tests. We set virtual obstacles in the outdoor experiment to mimic the complex trajectory in fully autonomous flights, as shown in Fig. 6.7. A snapshot shows the flight is in Fig. 6.5(b). The result presents the profile of velocity and acceleration in the 3^{rd} loop of the flight is given in Fig. 6.8. The average computing time in generating the optimal temporal trajectory is $108.94ms$ in this experiment. We also make an experiment as a response to the sharp turning case in before. The result is shown in Fig. 6.9. More details and online visualizations about the indoor and outdoor experiments are given in the attached video.

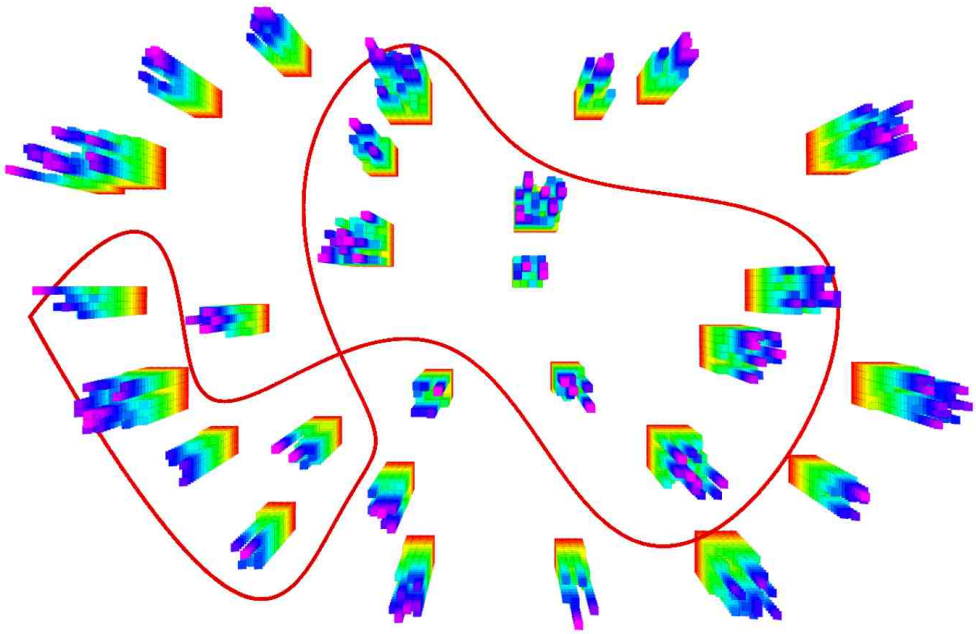


Figure 6.7. The spatial trajectory generated in the outdoor environment with virtual obstacles placed randomly. Our quadrotor platform does not equip online mapping capabilities. Therefore we place virtual obstacles in this experiment to mimic the real applications in fully autonomous navigation.

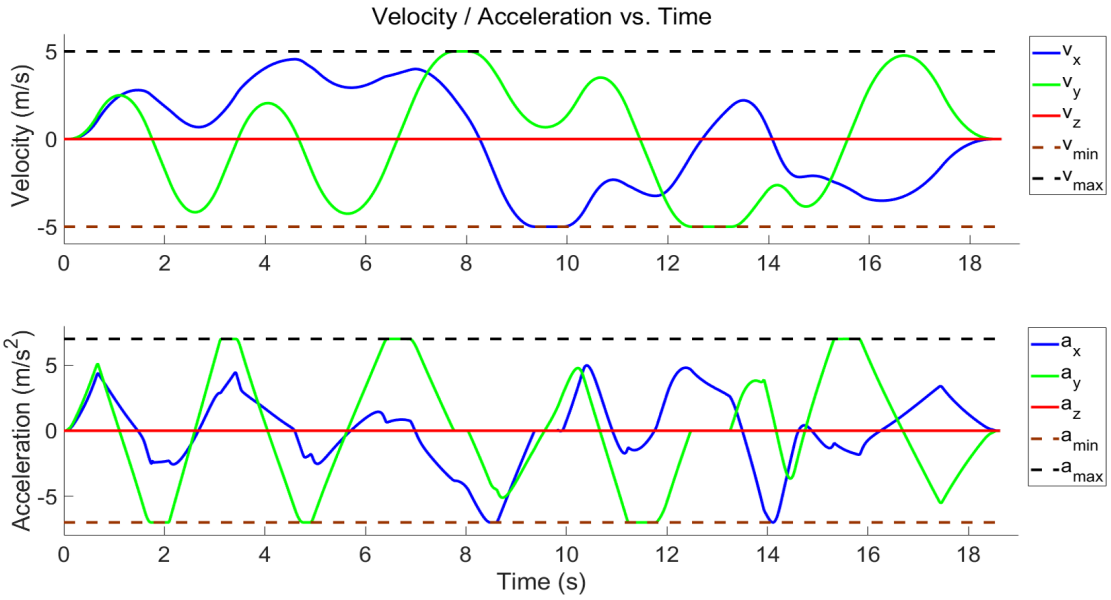


Figure 6.8. The result of a sample flight in the outdoor experiment. In the flight, the maximum velocity and acceleration allowed for the quadrotor are set as $5.0m/s$ and $7.0m/s^2$. The generated highest velocity and acceleration in one axis are $5.0m/s$ and $7.0m/s^2$.

Figure 6.9. The experiment of the sharp turning. The color of the trajectory indicates the magnitude of speed. Red color indicates the highest speed ($5m/s$) while the green indicates the lowest speed in the flight. The quadrotor is shown in a black model. The velocity and acceleration of the vehicle are shown in green and yellow arrows.

6.5 Discussion

In this chapter, we propose a method to generate the piecewise minimal-time polynomial trajectory for quadrotors. We decouple the traditional time-dependent trajectory generation problem into spatial and temporal layers and propose a method to generate time-optimal trajectory efficiently. The effectiveness and efficiency of the proposed method are validated in both numerical simulations and onboard experiments. The decomposition and optimization achieve the full usage of actuators for quadrotors to cruise in full-speed. We plan to integrate the proposed method into a complete

autonomous quadrotor with online perception like our previous work [32], to achieve high-speed autonomous flights in cluttered environments. The biggest challenge is, when the initial and final states of the temporary optimization are not both zero, there may not exists a feasible solution. We are going to investigate the feasibility in the future.

CHAPTER 7

TEACH-REPEAT-REPLAN: A FRAMEWORK ENABLES AUTONMOUS DRONE RACING

7.1 Research Background and Motivation

As the development of autonomy in aerial robots, Micro Aerial Vehicle (MAV) has been more and more involved in our daily life. Among all applications that emerged in recent years, quadrotor teach-and-repeat has shown significant potentials in aerial videography, industrial inspection, and human-robot interaction. In this paper, we investigate and answer the problem of what is the best way to incorporate a human's intention in autonomous and aggressive flight, and what is a flexible, robust and complete aerial teach-and-repeat system.

There is a massive market for consumer drones nowadays. However, we observe that most of the operators of consumer drones are not professional pilots and would struggle in generating their ideal trajectory for a long time. In some scenarios, such as the drone racing or aerial filming, a beginner-level pilot is impossible to control the drone to finish the race safely or take an aerial video smoothly unless months of training. Also, there is considerable demand in applying drones to repetitive industrial inspections or search-and-rescue missions, where human provides a preferable routine. In these situations, demonstrating a desirable trajectory and letting the drone to repeat it is a common wish. However, the taught trajectory generated by an unskilled pilot is usually incredibly hard or dynamically infeasible to repeat, especially in some cluttered environments. Moreover, most of the vision-based teach-and-repeat applications [30], [21], [26], such as our previous work [30], are sensitive to changing environments. In [30], even the environment changes very slightly, the global map has to be rebuilt, and the teaching has to be redone.

Based on these observations, instead of asking the drone to follow the human-piloted trajectory exactly, we only require the human operator to provide a rough trajectory with an expected topological structure. Such a human's teaching trajectory can be arbitrarily slow or jerky, but it captures the rough route the drone is expected to fly. Our system then autonomously converts this poor teaching trajectory to a topological equivalent and energy-time efficient one with an expected

aggressiveness. Moreover, during the repeating flight, our system locally observes environmental changes and re-plans sliding-windowed safe trajectories to avoid unmapped or moving obstacles. In this way, our system can deal with changing environments. Our proposed system extends the classical robotics teach-and-repeat framework and is named as *teach-repeat-replan*. It is complete, flexible, and robust.

In our proposed system, the surrounding environment is reconstructed by onboard sensors. Then the user's demonstrated trajectory is recorded by virtually controlling the drone in the map with a joystick or remote controller. Afterward, we find a flight corridor that preserves the topological structure of the teaching trajectory. Global planning is decoupled as spatial and temporal planning sub-problems. Having the flight corridor, an energy-optimal spatial trajectory that is guaranteed to be safe, and a time-optimal temporal trajectory that is guaranteed to be physically feasible, are iteratively generated. In repeating, while the quadrotor is tracking the global spatial-temporal trajectory, a computationally efficient local map [34] is fused onboard by stereo cameras. Based on local observations, our proposed system uses a sliding-window fast re-planning method [95] to avoid possible collisions. The re-planning module utilizes gradient information to locally wrap the global trajectory to generate safe and kinodynamic feasible local plans against unmapped or moving obstacles.

The concept of generating optimal topology-equivalent trajectories for quadrotor teach-and-repeat was first proposed in our previous research [30]. In [30], once the repeating trajectory is generated, the drone executes it without any other considerations. In that work [30], the environment must remain intact during the repeating, and the localization of the drone is assumed to be perfect. These requirements are certainly not guaranteed in practice, therefore, prevent the system from being applied widely. In this paper, we extend the framework of the classical teach-and-repeat and propose several new contributions to make our system complete, robust, and flexible. Contributions are listed as:

1. We advance our flight corridor generation method. The flight corridor we use now provides much more optimization freedom compared to our previous work [30]. The improvement of the flight corridor facilitates the generation of more efficient and smooth global trajectories. Moreover, we propose methods to accelerate the corridor generation on both CPU and GPU.

2. We introduce our previous works on online mapping [34] and re-planning [95] into our sys-

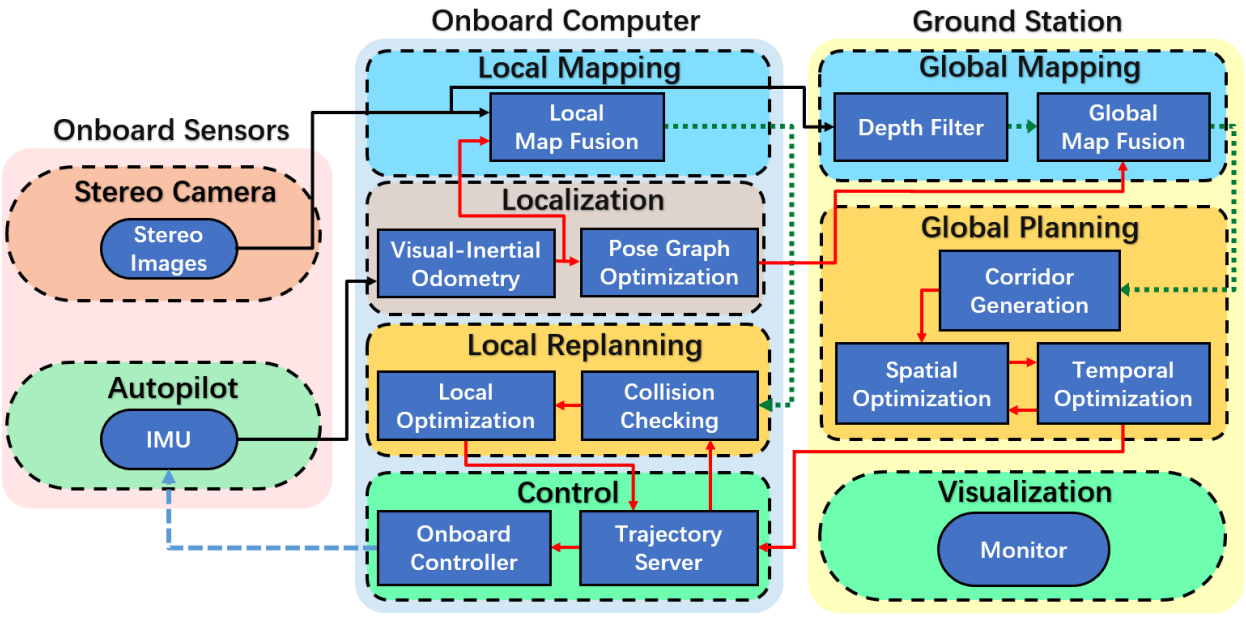


Figure 7.1. The software architecture of our quadrotor system. Global mapping, planning, and visualization are running on a ground station, while state estimation, local sensing, and re-planning are running onboard.

tem, to improve the robustness against errors of global maps, drifts of localization, and environmental changes and moving obstacles.

3. We present a whole set of experiments and comparisons in various scenarios to validate our system.

4. We release all components in the system as open-source packages, which include local/global planning, perception, and localization, and onboard controller.

7.2 System Overview

7.2.1 System Architecture

The overall software and hardware architecture of our quadrotor system are shown in Fig. 7.1 and 7.2. The global mapping, flight corridor generator, and global spatial-temporal planner are done on an off-board computer. Other online processings are running onboard on the drone during the flight. Before teaching, the global map is built by onboard sensors. During teaching, a flight corridor is generated by inflating the teaching trajectory. Then the spatial and temporal trajectories are optimized iteratively within the flight corridor under a coordinate descent scheme [90]. The

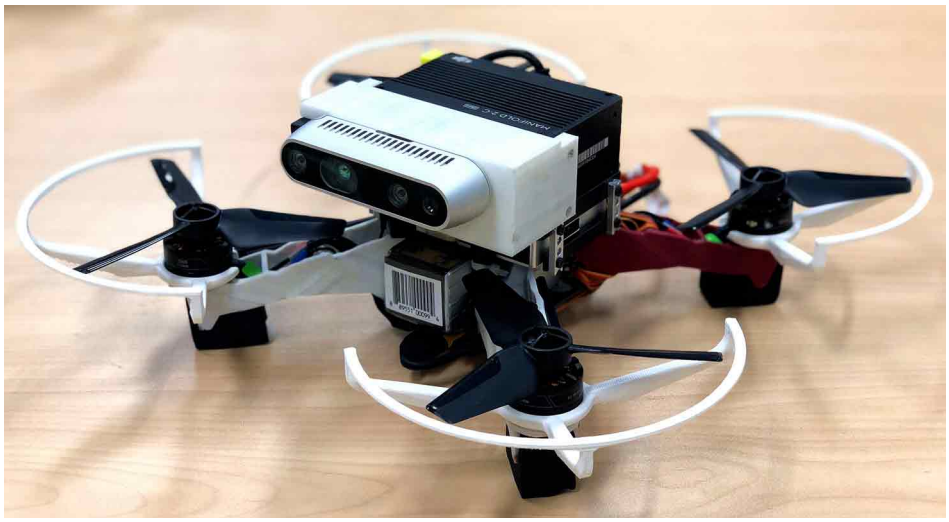


Figure 7.2. The hardware setting of our autonomous drone system.

local planner using gradient-based optimization is running onboard to avoid unexpected obstacles observed in the repeating flights. For trajectory tracking, we use a geometric controller [47]. And the attitude is stabilized by the autopilot.

7.2.2 Globally Consistent Localization and Mapping

We use VINS [68], a robust visual-inertial odometry (VIO) framework, to localize the drone. Moreover, the loop closure detection and global pose graph optimization are used in our system, to globally correct the pose estimation. The global mapping is done by fusing depth measurements from the stereo cameras with the pose estimation. By using our previous research on deformable map [86], our global mapping module maintains a series of sub-maps with each attached to a keyframe. In this way, the map is attached to the pose graph and is therefore globally driftless. During the mapping, when a loop closure is detected, keyframes in the global pose graph are corrected, and all sub-maps are deformed accordingly. The global pose graph optimization is also activated during the repeating. When loop closure is detected, the pose of the drone is corrected accordingly to eliminate the drift.

7.2.3 Global Spatial-Temporal Planning

For an extremely poor teaching trajectory, both the geometric shape and time profile of it is far from optimal and therefore useless, or even harmful for conducting optimization. However, the topolog-

ical information of the teaching trajectory is essential since it reflects the human's intention. To preserve the topological information, we group the free space around the teaching trajectory to form a flight corridor (Sect. 7.3). The corridor contains the teaching trajectory within it, shares the same topological structure, and provides large freedom for optimization. It's hard to concurrently optimize a trajectory spatially and temporally in the flight corridor. However, generating a safe spatial trajectory given a fixed time allocation (Sect. 7.4.1) and optimizing the time profile of a fixed spatial trajectory (Sect. 7.4.2) are both conquerable. Therefore, we iteratively optimize the trajectory in the space-time joint solution space by designing a coordinate descent [90] framework. An objective with weighting energy and time duration is defined for optimization. We firstly generate a spatial trajectory whose energy is minimized, then we use the temporal optimization to obtain the optimal time profile of it. The optimal time profile is used to parametrize a trajectory again for spatial optimization. The spatial-temporal optimizations are done iteratively until the total cost cannot be reduced any more.

7.2.4 Local Collision Avoidance

In practice, the accumulated drift of VIO is unavoidable, and the recall rate of loop closure is unstable. Although we have built a dense global map, when the drift is significant and not corrected by loop detection in time, the quadrotor may have collisions with obstacles. Moreover, the environment may change or contain moving obstacles. Our previous work [30] has to re-build the map when changes happen and can not deal with dynamic obstacles. To resolve the above issues, we integrate our previous local map fusion module [34] into our system to detect collisions locally and serves the local trajectory optimization. Also, we propose a sliding-window local replanning method based on our previous research on quadrotor local planning [95], to avoid collisions on the flight.

In the repeating phase, the drone controls its yaw angle to face its flying direction and build a local map by stereo cameras. We consistently check the local trajectory within a replanning time horizon. If collisions along the local trajectory are reported, replanning is triggered to wrap the trajectory out of obstacles by gradient-based optimization [95].

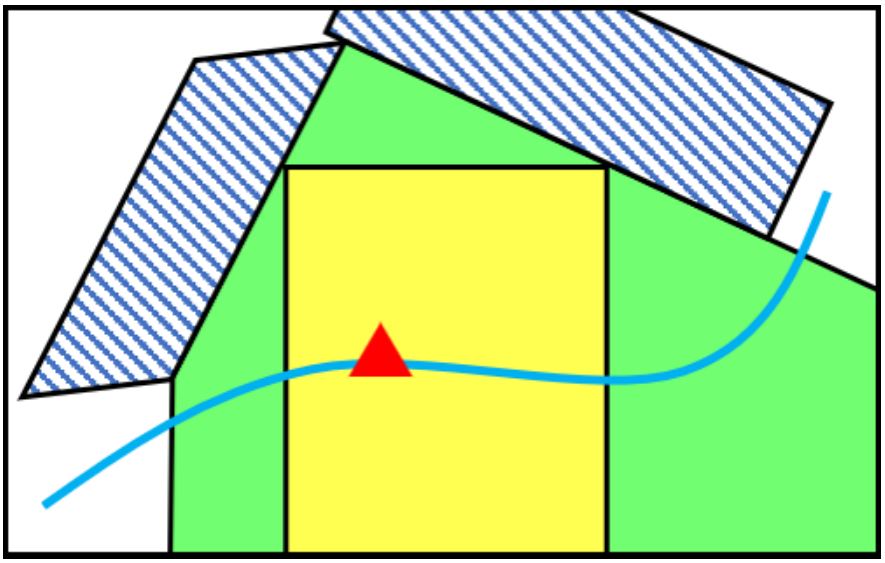
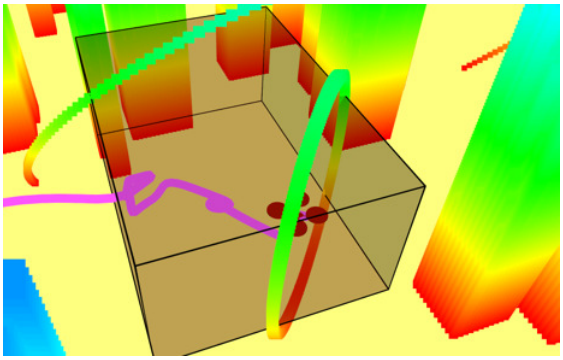


Figure 7.3. An illustration of free space captured by an axis-aligned cube and a general polyhedron. Obstacles are shown in dashed lines. The blue curve is the teaching trajectory of humans. The red triangle is the seed for finding local free space. The axis-aligned cube and a corresponding general convex polyhedron are shown in yellow and green, respectively.

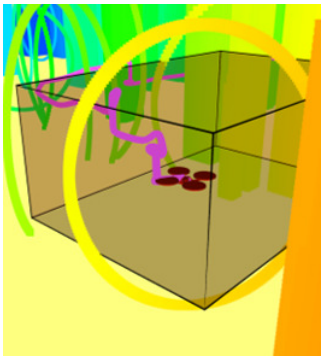
7.3 Flight Corridor Generation

As stated in Sect. 7.2.3, the first step of our global planning is to build a flight corridor around the teaching trajectory for spatial-temporal trajectory optimization. In our previous work [30], the flight corridor is constructed by finding a series of axis-aligned cubes, which may sacrifice much space, especially in a highly nonconvex environment, as is shown in Fig 7.3. A more illustrative comparison is shown in Fig. 7.4, where the convex polyhedron captures much more free space than the simple cube. Using simple axis-aligned cubes significantly limit the solution space of trajectory optimization, which may result in a poor solution. What's more, in situations where the free space is very limited, such as flying through a very narrow circle, a cube-based corridor [30] may even fail to cover all teaching trajectory and result in no solutions existing in the corridor. Therefore, to utilize the free space more sufficiently and adapt to even extremely cluttered maps, we propose a method to generate general, free, large convex polyhedrons.

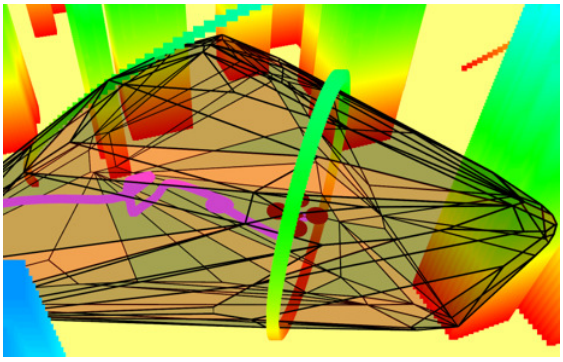
Since the human's teaching trajectory may be arbitrarily jerky, we cannot assume there is a piecewise linear path to initiate the polyhedron generation, as in [53]. Also, we make no requirements on the convexity of obstacles in the map as in [14]. Our method is based on convex set clustering, which is similar to [7], but is different and advanced at:



(a) Axis-aligned cube, side view



(b) Front view



(c) General convex polyhedron, side view

(d) Front view

Figure 7.4. The comparison of an axis-aligned cube and a general convex polyhedron. The cube and the polyhedron are generated to the largest volume they may have, from the same seed coordinate. The way to inflate the cube is stated in our previous paper [30]. The method to find the general free polyhedron will be detailed later in Sect. 7.3.1.

1. We make no assumption on the growing directions of convex clusters and generate completely collision-free polyhedrons based on our dense occupancy map.

2. We introduce several careful engineering considerations which significantly speed-up the clustering.

3. We fully utilize the parallel structure of this algorithm and accelerate it over an order of magnitude in GPUs.

4. We introduce a complete pipeline from building the convex polyhedron clusters to establishing constraints in trajectory optimization.

7.3.1 Convex Cluster Inflation

The core algorithm for the construction of the flight corridor is to find the largest convex free polyhedron at a given coordinate. In this paper, we use an occupancy grid map to represent the environment. Each polyhedron in the flight corridor is the convex hull of a voxel set, which is convex and contains only free voxels. The voxel set is found by clustering as many free voxels as possible around an arbitrary seed voxel. In this paper, we name the voxel set as *convex cluster*, and the process of finding such a set as *convex cluster inflation*. Our method for finding such a *convex cluster* is based on the definition of the convex set:

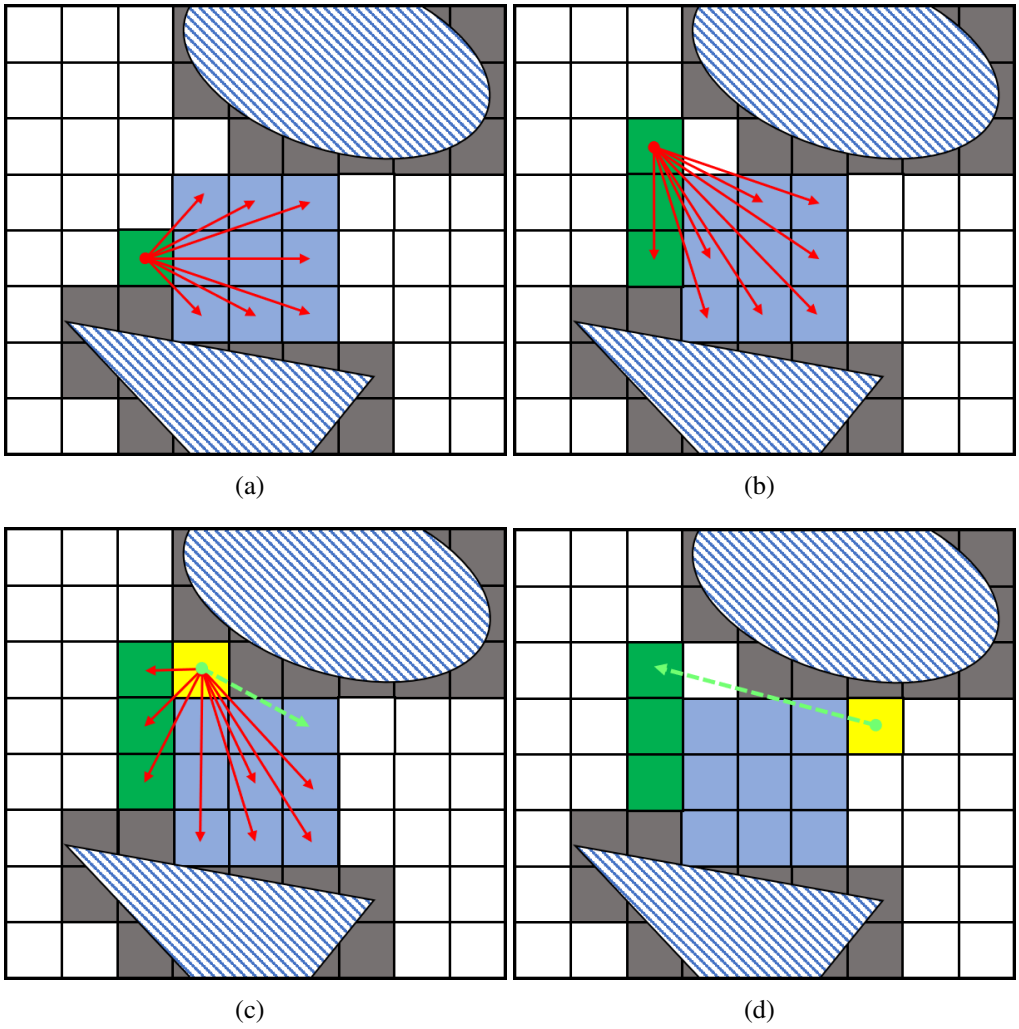


Figure 7.5. An illustration of the *convex cluster inflation*. In (a) and (b), all qualified neighbor voxels are added to the *convex cluster*. In (c) and (d), since an occupied voxel occludes a ray (the green arrow) to one of the clustered voxels, the testing voxel (in yellow) is excluded to the cluster.

Algorithm 5 Convex Cluster Inflation

Notation:
Seed Voxel: p^s, Grid Map: $\mathcal{M}$, Convex Cluster: $\mathcal{C}$,
Candidate Voxel Set: $\mathcal{C}^+$, Active Voxel Set: $\mathcal{C}^*$
Input: p^s, $\mathcal{M}$
function CONVEX_INFLATION(p^s, $\mathcal{M}$)
    $\mathcal{C} \leftarrow \{p^s\}$
    $\mathcal{C}^* \leftarrow \varnothing$
    $\mathcal{C}^+ \leftarrow$ GET_NEIGHBORS($\mathcal{C}$)
    while $\mathcal{C}^+ \neq \varnothing$ **do**
        for $p^+ \in \mathcal{C}^+$ **do**
            if CHECK_CONVEXITY($p^+, \mathcal{C}, \mathcal{M}$) **then**
                $\mathcal{C} \leftarrow \mathcal{C} \cup p^+$
                $\mathcal{C}^* \leftarrow \mathcal{C}^* \cup p^+$
            end if
        end for
        $\mathcal{C}^+ \leftarrow \varnothing$
        $\mathcal{C}^+ \leftarrow \mathcal{C}^+ \cup$ GET_NEIGHBORS($\mathcal{C}^*$)
        $\mathcal{C}^* \leftarrow \varnothing$
    end while
    Output: $\mathcal{C}$
end function

Definition: A set $\mathcal{S}$ in a vector space over $\mathcal{R}$ is called a convex set if the line segment joining any pair of points of $\mathcal{S}$ lies entirely in $\mathcal{S}$. [45].

The pipeline for iteratively inflating such a cluster while preserving convexity is stated in Alg. 5. Our method operates on a 3D occupancy map $\mathcal{M}$ where voxels are labeled as *obstacle* or *free*. Three voxel sets are maintained in the algorithm. $\mathcal{C}$ stands for the targeting convex voxel cluster. $\mathcal{C}^+$ is the set of voxels that are tentative to be added to $\mathcal{C}$ in this iteration. And $\mathcal{C}^*$ contains newly added voxels which preserve the convexity. The cluster inflation starts by adding the seed voxel p to $\mathcal{C}$, and adding all neighboring voxels of p to $\mathcal{C}^+$. In an iteration, each voxel p^+ in $\mathcal{C}^+$ is checked whether it can preserve convexity using the function CHECK_CONVEXITY($p^+, \mathcal{C}, \mathcal{M}$). This function, as is shown in Alg. 6, casts rays from p^+ to each existing voxels in $\mathcal{C}$. According to the definition of the convex set, $\mathcal{M}$ with p^+ is convex if and only if all rays are collision-free. Based on this criteria, qualified voxels are considered as *active* voxels and are added into $\mathcal{C}$ and $\mathcal{C}^*$. Neighboring voxels of all *active* voxels p^* are traversed and collected by the function GET_NEIGHBORS($\mathcal{C}^*$) for next iteration. The inflation ends when $\mathcal{C}^+$ becomes empty, which implies no additional voxels can be added into $\mathcal{C}$. Fig. 7.5 also illustrates the procedure of the *convex cluster inflation*.

Having a *convex cluster* consists of a number of voxels, we convert it to the algebraic representation of a polyhedron for following spatial trajectory optimization. Quick hull algorithm [3]

Algorithm 6 Convexity Checking

```
1: Notation: Ray Cast: l
2: function CHECK_CONVEXITY(p^+, C, M)
3:     for p ∈ C do
4:         l ← CAST_RAY(p^+, p)
5:         if PASS_OBS(l, M) then
6:             return False
7:         end if
8:     end for
9:     return True
10: end function
```

is adopted here for quickly finding the convex hull of all clustered voxels. The convex hull is in vertex representation (V-representation) $\{V_0, V_1, ..., V_m\}$, and is then converted to its equivalent hyperplane representation (H-representation) by using double description method [24]. The H-representation of a 3-D polyhedron is a set of affine functions:

$$\begin{aligned} a_0^x \cdot \mathbf{x} + a_0^y \cdot \mathbf{y} + a_0^z \cdot \mathbf{z} &\leq k_0, \\ a_1^x \cdot \mathbf{x} + a_1^y \cdot \mathbf{y} + a_1^z \cdot \mathbf{z} &\leq k_1, \\ &\vdots \\ a_n^x \cdot \mathbf{x} + a_n^y \cdot \mathbf{y} + a_n^z \cdot \mathbf{z} &\leq k_n, \end{aligned} \tag{7.1}$$

where $\{a_n^x, a_n^y, a_n^z\}$ is the normal vector of the 3-D hyperplane and k_n is a constant.

7.3.2 CPU Acceleration

As is shown in Alg. 5, determining whether a voxel preserves the convexity needs to conduct ray-casting to all existing voxels in the *convex cluster*. Iterating with all voxels and rays makes this algorithm impossible to run in real-time, especially when the occupancy grid map has a fine resolution. To make the polyhedron generated in real-time, we take careful engineering considerations on the implementations and propose some critical techniques to increase the efficiency significantly.

Polyhedron Initialization

We initialize each convex cluster as an axis-aligned cube using our previous method [32], which can be done very fast since only index query ($\mathcal{O}(1)$) operations are necessary. After inflating the

cube to its maximum volume, as in Fig. 7.3, we switch to the convex clustering to further group convex free space around the cube.

The proposed polyhedron initialization may result in a final polyhedron different from the one which is clustered from scratch. This is because an axis-aligned cube only inflates in x, y, z directions while a *convex cluster* grows in all possible directions (26-connections in a 3D grid map). However, this initialization process is reasonable. Our purpose is not making each polyhedron optimal but capturing as much as possible free space than a simple cube cannot. In practice, the initialization provides a fast discovery of nearby space which is easy to group, and does not prevent the following *convex cluster inflation* to refine the polyhedron and find sizeable free space. In Sect. 7.5.3, we show that the initialization process significantly improves the computing efficiency with only a neglectable sacrifice on the volume of the final polyhedron.

Early Termination

We label all voxels in the cluster as *inner* voxels which inside the *convex cluster*, and *outer* voxels which on the boundary of the *convex cluster*. When traversing a ray from a candidate voxel to a voxel in the *convex cluster*, we early terminate the ray casting when it arrives at a voxel labeled as *inner*.

Theorem 1: The early termination at *inner* voxels is sufficient for checking convexity.

Proof: According to the definition of convex set, a ray connecting an *inner* voxel to any other voxel in the *convex cluster* lies entirely in the *convex cluster*. Hence, the extension line of an *inner* voxel must lie inside the *convex cluster* and therefore it must pass the convexity check.

Voxel Selection

To further reduce the number of voxels that need to be cast rays, given a candidate voxel, only *outer* voxels are used to check its convexity.

Theorem 2: Using *outer* voxels of a *convex cluster* is sufficient for checking convexity.

Proof: Obviously, the *convex cluster* is a closed set with *outer* voxels at its boundary. The candidate voxel is outside this set. Therefore, casting a ray from any *inner* voxel to the candidate voxel must pass one of the *outer* voxels. According to *Theorem 1*, checking convexity of this ray

can terminate after the ray passes an *outer* voxels, which means for a candidate voxel, checking rays cast to *outer* voxels is sufficient.

By introducing the above techniques, the proposed *convex cluster inflation* can work in real-time for a mediate grid resolution ($0.2m$) on CPUs. The efficacy of these techniques is numerically validated in Sect. 7.5.3.

7.3.3 GPU Acceleration

We propose a parallel computing scheme that significantly speeds up the inflation by one order of magnitude where a GPU is available. As is shown in Sec. 7.3.1, when the *convex cluster* discovers a new neighboring voxel, sequentially traversing and checking all rays is naturally parallelizable. With the help of many-core GPUs, we can cast rays and check collisions parallelly. Moreover, to fully utilize the massively parallel capability of a GPU, reduce the serialize operations, and minimize the data transferring between CPU and GPU, we examine all potential voxels of the cluster parallelly in one iteration. Instead of discovering a new voxel and checking its rays, we find all neighbors of the active set $\mathcal{C}^*$ and check their rays all in parallel. The detailed procedure is presented in Alg. 7, where GET_NEIGHBORS($\mathcal{C}$) collects all neighbors of a set of voxels, and PARA_CHECK_CONVEXITY($\mathcal{C}^+, \mathcal{C}$, $\mathcal{M}$) checks the convexity of all candidate voxels parallelly in GPUs. Note that in the serialized version of the proposed method, the voxel discovered earlier may prevent later ones from being clustered, as is illustrated in Fig. 7.5. However, in the parallel clustering, all voxels examined at the same time may add conflicting voxels to the cluster. Therefore, we introduce an additional variable, r to record sequential information of voxels. As shown in Alg. 8, the kernel function is running on the GPU per block. It checks the ray cast from every candidate voxel in $\mathcal{C}^+$ to a cluster voxel in $\mathcal{C}$ and to each other candidate voxel which has a prior index. After that, the function CHECK_RESULTS(r) selects all qualified voxels and adds them into $\mathcal{C}$. Firstly, candidate voxels that have collisions with $\mathcal{C}$ are directly excluded. Then, candidate voxels having collisions with other candidates that have already been added into $\mathcal{C}$ are excluded. In this way, we finally get the same results as in the serialized version of the clustering. The efficacy of parallel computing is shown in Sect. 7.5.3.

Algorithm 7 Parallel Convex Cluster Inflation

Notation:
Parallel Raycasting Result: r
function PARA_CONVEX_INFLATION(p^s, $\mathcal{M}$)
    $\mathcal{C} \leftarrow \{p^s\}$
    $\mathcal{C}^* \leftarrow \varnothing$
    $\mathcal{C}^+ \leftarrow$ GET_NEIGHBORS($\mathcal{C}$)
    while $\mathcal{C}^+ \neq \varnothing$ **do**
        /* *GPU data uploads* */
        $r \leftarrow$ PARA_CHECK_CONVEXITY($\mathcal{C}^+, \mathcal{C}$, $\mathcal{M}$)
        /* *GPU data downloads* */
        $\mathcal{C}^* \leftarrow$ CHECK_RESULTS(r)
        $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}^*$
        $\mathcal{C}^+ \leftarrow$ GET_NEIGHBORS($\mathcal{C}^*$)
    end while
    Output: $\mathcal{C}$
end function

Algorithm 8 Parallel Convexity Checking

function PARA_CHECK_CONVEXITY($\mathcal{C}^+, \mathcal{C}$, $\mathcal{M}$)
    for $p_i^+ \in \mathcal{C}^+$ **do**
        $r[i].status \leftarrow True$
        for $p \in \mathcal{C}$ **do**
            /* *Kernel function starts* */
            $l \leftarrow$ CAST_RAY(p_i^+, p)
            if PASS_OBS($l, \mathcal{M}$) **then**
                $r[i].status \leftarrow False$
            end if
            /* *Kernel function ends* */
        end for
        for $p_j^+ \in \mathcal{C}^+$ AND $j < i$ **do**
            /* *Kernel function starts* */
            $l \leftarrow$ CAST_RAY(p_i^+, p_j^+)
            if PASS_OBS($l, \mathcal{M}$) **then**
                $r[i].status \leftarrow Pending$
                $r[i].list.push\_back(j)$
            end if
            /* *Kernel function ends* */
        end for
    end for
    return r
end function

Algorithm 9 Parallel Check Results

```
1:
2: function CHECK_RESULTS(r, C+)
3:     for p_i+ ∈ C+ do
4:         if r[i].status == True then
5:             C* ← C* ∪ p+
6:         else if r[i].status == Pending then
7:             for j ∈ r[i].list do
8:                 if p_j+ ∈ C* then
9:                     go to 3
10:                end if
11:            end for
12:            C* ← C* ∪ p+
13:        end if
14:    end for
15:    return C*
16: end function
```

7.3.4 Corridor Generation and Loop Elimination

Algorithm 10 Flight Corridor Generation

```
1: Notation: Flight Corridor G, Drone Position p, Convex Polyhedron P
2: Initialize :
3: P ← CONVEX_INFLATION(p, M)
4: G.push_back(P)
5: while Teaching do
6:     p ← UPDATE_POSE()
7:     if OUTSIDE(p, G[-1]) then
8:         if INSIDE(p, G[-2]) then
9:             G.pop_back()
10:        else
11:            P = CONVEX_INFLATION(p, M)
12:            G.push_back(P)
13:        end if
14:    end if
15: end while
16: return G
```

Since the trajectory provided by a user may behave arbitrarily jerky and contain local loops, we introduce a specially designed mechanism to eliminate unnecessary loops, i.e., repeatable polyhedrons. The exclusion of repeatable polyhedrons is essential since in following trajectory optimization (Sect. 7.4), each polyhedron is assigned with one piece of the trajectory. Repeatable polyhedrons would result in an optimized trajectory looping as the user does, which is obviously

not efficient. The pipeline of the corridor generation is shown in Alg. 10 and Fig. 7.6. At the beginning of the teaching, the flight corridor is initialized by finding the maximum polyhedron around the position of the drone. Then as the human pilots the drone to move, we keep checking the drone's position. If it goes outside the last polyhedron ($\mathcal{G}$[-1]), we further check whether the drone discovers new free space or not. If the drone is contained within the second last polyhedron ($\mathcal{G}$[-2]), we can determine that the teaching trajectory has a loop, as shown in Fig. 7.6(c). Then, the last polyhedron in the corridor is regarded as repeatable and is therefore popped out from the corridor. Otherwise, as shown in Fig. 7.6(d), the drone is piloted to discover new space. Then a new polyhedron $\mathcal{P}$ is inflated and added to the tail of the corridor. The corridor generation is terminated when the teaching finish. The final flight corridor shares the same topological structure with the teaching trajectory since no obstacles are included in the corridor. And it has no unnecessary loops.

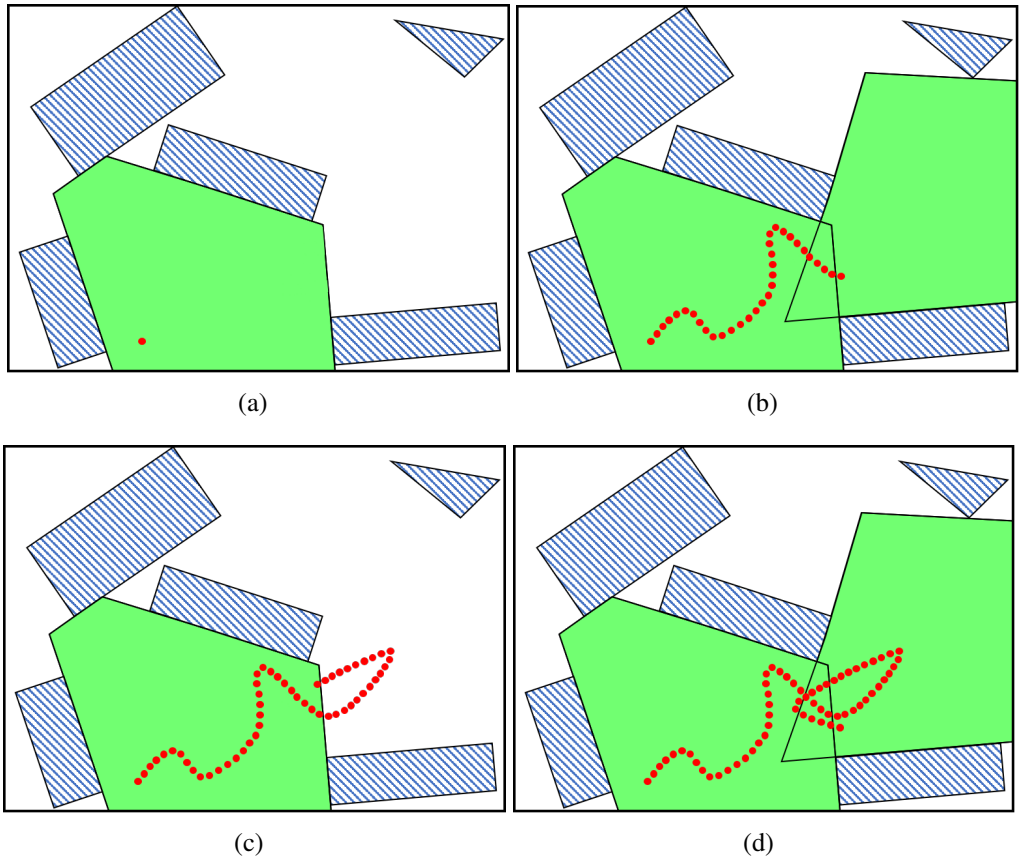


Figure 7.6. The flight corridor generation process. Red dots are coordinates along the teaching trajectory. (b), a new convex polyhedron is generated and added to the flight corridor when the drone leaves the corridor enters undiscovered space. (c), the drone leaves the last polyhedron and returns back to the second to last one, so the last polyhedron is deleted from the corridor.

7.4 Spatial-Temporal Global Trajecotry Optimization

7.4.1 Spatial Trajecotry Optimization

For the spatial optimization, we use the Bernstein basis to represent the trajectory as a piecewise Bézier curve, since it can be easily constrained in the flight corridor by enforcing constraints on control points. The Bernstein-based trajectory optimization is stated before in Chap. 4. However, to complete the derivation of spatial-temporal joint optimization, we briefly go through the mathematical formulation.

An i^{th}-order Bernstein basis is:

$$b_n^i(t) = \binom{n}{i} \cdot t^i \cdot (1-t)^{n-i}, \tag{7.2}$$

where n is the degree of the basis, $\binom{n}{i}$ is the binomial coefficient and t is the variable parameterizing the trajectory. An N-piece piecewise Bézier curve is written as:

$$f_\mu(t) = \begin{cases} \sum_{i=0}^{n} c_{\mu,1}^i b_n^i(t/T_1), & t \in [0, T_1], \\ \sum_{i=0}^{n} c_{\mu,2}^i b_n^i(t/T_2), & t \in [0, T_2], \\ \vdots & \vdots \\ \sum_{i=0}^{n} c_{\mu,N}^i b_n^i(t/T_N), & t \in [0, T_N]. \end{cases} \tag{7.3}$$

For the m^{th} piece of the curve, $c_{\mu.m}^i$ is the i^{th} control point, and T_m is the time duration. The spatial trajectory is generated in x, y, z dimensions, and $\mu \in x, y, z$. μ is omitted in the following derivation for brevity. In this equation, t is scaled by T_m since a standard Bézier curve is defined on $[0, 1]$.

Follow the formulation in minimum-snap [55], the squared jerk is minimized in this paper. Since the 3^{rd} order derivative of a curve corresponds to the angular velocity, the minimization of jerks alleviates the rotation and therefore facilitates visual tracking. The objective of the piecewise curve is:

$$J = \sum_{\mu}^{x,y,z} \sum_{m=1}^{N} \int_0^{T_m} \left(\frac{d^3 f_{\mu,m}(t)}{dt^3} \right)^2 dt. \tag{7.4}$$

which is in a quadratic form denoted as $\mathbf{c}^T \mathbf{Q} \mathbf{c}$. Here $\mathbf{c}$ is composited by all control points in *x, y, z* dimensions. $\mathbf{Q}$ is a semi-definite Hessian matrix.

For a Bézier curve, its higher order derivatives can be represented by linear combinations of corresponding lower-order control points. For the 1^{st} and 2^{nd} order derivatives of the m^{th} piece of the curve in Eq. 7.3, we have:

$$f'_m(t) = \sum_{i=0}^{n-1} n(c_m^{i+1} - c_m^i) b_{n-1}^i(\frac{t}{T_m}), \tag{7.5}$$
$$f''_m(t) = \sum_{i=0}^{n-2} n(n-1)(c_m^{i+2} - 2c_m^{i+1} + c_m^i) b_{n-2}^i(\frac{t}{T_m}).$$

Boundary Constraints

The trajectory has the boundary constraints on the initial state (p^0, v^0, a^0) and the final state (p^f, v^f, a^f) of the quadrotor. Since a Bézier curve always passes the first and last control points, we enforce the boundary constraints by directly setting equality constraints on corresponding control points in each dimension:

$$\begin{aligned} &c_0^0 = p^0, \\ &c_N^n = p^f, \\ &n(c_0^1 - c_0^0) = v^0, \\ &n(c_N^n - c_N^{n-1}) = v^f, \\ &n(n-1)(c_0^2 - 2c_0^1 + c_0^0) = a^0, \\ &n(n-1)(c_N^n - 2c_N^{n-1} + c_N^{n-2}) = a^f. \end{aligned} \tag{7.6}$$

Continuity Constraints

For ensuring smoothness, the minimum-jerk trajectory must be continuous for derivatives up to 2^{nd}-order at all connecting points on the piecewise trajectory. The continuity constraints are enforced by setting equality constraints between corresponding control points of two consecutive curves. For the j^{th} and $(j+1)^{th}$ pieces of the curve, we can write the equation in each dimension as:

$$\begin{aligned} &c_j^n = c_{j+1}^0, \\ &(c_j^n - c_j^{n-1})/T_j = (c_{j+1}^1 - c_{j+1}^0)/T_{j+1}, \\ &(c_j^n - 2c_j^{n-1} + c_j^{n-2})/T_j^2 = (c_{j+1}^2 - 2c_{j+1}^1 + c_{j+1}^0)/T_{j+1}^2, \end{aligned} \tag{7.7}$$

Safety Constraints

The safety of the trajectory is guaranteed by enforcing each piece of the curve to be inside the corresponding polyhedron. Thanks to the convex hull property, an entire Bézier curve is confined within the convex hull formed by all its control points. Therefore we constrain control points using hyperplane functions obtained in Eq. 7.1. For the i^{th} control point $c^i_{j,x}, c^i_{j,y}, c^i_{j,z}$ of the j^{th} piece of the trajectory in x, y, z dimensions, constraints are:

$$\begin{aligned} a^x_0 \cdot c^i_{j,x} + a^y_0 \cdot c^i_{j,y} + a^z_0 \cdot c^i_{j,z} &\leq k_0, \\ a^x_1 \cdot c^i_{j,x} + a^y_1 \cdot c^i_{j,y} + a^z_1 \cdot c^i_{j,z} &\leq k_1, \\ &\vdots \\ a^x_n \cdot c^i_{j,x} + a^y_n \cdot c^i_{j,y} + a^z_n \cdot c^i_{j,z} &\leq k_n, \end{aligned} \tag{7.8}$$

Constraints in Equs. 7.6 and 7.7 are affine equality constraints ($\mathbf{A}_{eq}\mathbf{c} = \mathbf{b}_{eq}$) and Eq. 7.8 is in affine in-equality formulation ($\mathbf{A}_{ie}\mathbf{c} \leq \mathbf{b}_{ie}$). Finally, the spatial trajectory optimization problem is formulated as a QP as follows:

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{Q}\mathbf{c} \\ \text{s.t.} \quad & \mathbf{A}_{eq}\mathbf{c} = \mathbf{b}_{eq}, \\ & \mathbf{A}_{ie}\mathbf{c} \leq \mathbf{b}_{ie}. \end{aligned} \tag{7.9}$$

Unlike our previous works on corridor constrained trajectory [32, 31], here the kinodynamic feasibility (velocity and acceleration) is not guaranteed by adding higher-order constraints into this program, but by temporal optimization (Sect. 7.4.2). For a rest-to-rest trajectory, the program in Eq. 7.9 is always mathematically feasible.

7.4.2 Temporal Trajectory Optimization

In spatial optimization, a corridor-constrained spatial trajectory is generated given a fixed time allocation. To optimize the trajectory temporally, we design a re-timing function $\{t(\tau) : \tau \rightarrow t\}$ to map the original time variable t to a variable τ. The relation between τ and t is shown in Fig. 7.7. In this paper, the re-timing function $t(\tau)$ is named as the temporal trajectory, and finding the optimal $t(\tau)$ is called the temporal optimization. For the N-piece spatial curve defined in Equ. 7.3, we write

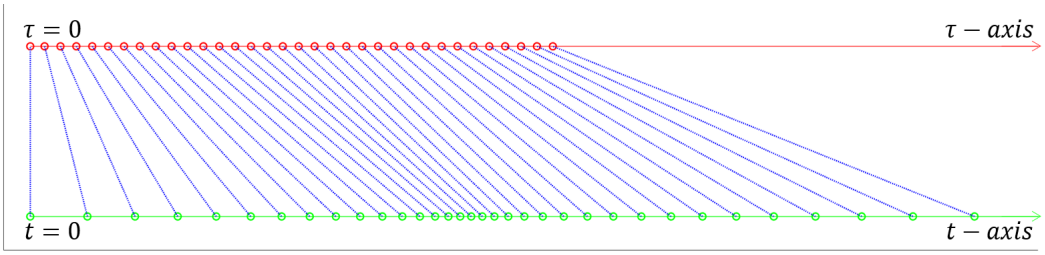


Figure 7.7. The effect of the temporal optimization. t and τ are the time profile of the spatial trajectory before and after optimization.

$t(\tau)$ as a corresponding N-piece formulation:

$$t(\tau) = \begin{cases} t_1(\tau), & t_1(0) = 0, t_1(\mathcal{T}_1^*) = T_1, t_1 \in [0, T_1] \\ t_2(\tau), & t_2(0) = 0, t_2(\mathcal{T}_2^*) = T_2, t_2 \in [0, T_2] \\ \vdots & \vdots \\ t_N(\tau), & t_N(0) = 0, t_N(\mathcal{T}_N^*) = T_N, t_N \in [0, T_N] \end{cases} \tag{7.10}$$

where $T_1, T_2, ...T_N$ are original time durations of the spatial curve $f_\mu(t)$, and $\mathcal{T}_1^*, \mathcal{T}_2^*, ...\mathcal{T}_N^*$ are time durations after temporal optimization. Since physically time only increases, $t(\tau)$ is a monotonically increasing function. Therefore we have $\dot{t}(\tau) \geq 0$. Note what's different from Chap. 6 is here we re-map the original time parameter t to the best time parameter τ, instead of parametrizing the spatial trajectory first to a virtual parameter.

The total time $\mathcal{T}$ of the temporal trajectory can be written as:

$$\mathcal{T} = \int_0^{\mathcal{T}} 1 d\tau = \sum_{m=1}^{N} \int_0^{T_m} \frac{1}{\dot{t}_m} dt, \tag{7.11}$$

considering $\dot{t} = dt/d\tau$. We can introduce a regularization term that penalizes the changing rate of t, to trade-off the minimization of time and control extremeness, or so-called motion aggressiveness, in our final temporal trajectory. The objective function is then written as:

$$\mathcal{J} = \sum_{m=1}^{N} \int_1^{T_m} \left(\frac{1}{\dot{t}_m} + \rho \cdot \ddot{t}_m^{\,2} \right) dt, \tag{7.12}$$

where ρ is a weight of the aggressiveness. By setting a larger ρ we can obtain more gentle motions in the temporal trajectory. If $\rho = 0$, the temporal optimization is solved for generating motions as fast as possible. The motions generated with a large ρ can be viewed in our previous work [30].

The derivation of the complete time optimization program is the same as in Chap. 6, and is therefore omitted here. Finally, the optimization problem is formulated as a standard Second Order Cone Program (SOCP):

$$\begin{aligned} \min \quad & \mathbf{h}^T\mathbf{d} + \rho \cdot t, \\ \text{s.t.} \quad & \mathbf{A}_{eq} \cdot \mathbf{x} = \mathbf{b}_{eq}, \\ & \mathbf{A}_{ie} \cdot \mathbf{x} \leq \mathbf{b}_{ie}, \\ & (t, 1, \mathbf{a}) \in Q_r^{2+K\cdot(m+1)}, \\ & (d_i^k, c_i^{k+1} + c_i^k, \sqrt{2}) \in Q_r^3, k = 0, 1, ..., K-1, \\ & (b_i^k + 1, b_i^k - 1, 2c_i^k) \in Q^3, k = 0, 1, ..., K, \end{aligned} \tag{7.13}$$

In our *teach-repeat-replan* system, since the global repeating trajectory always has static initial and final states, Equ. 7.13 is always mathematically feasible regardless of the solution of spatial optimization. Because a feasible solution of the optimization program can always be found by infinitely enlarging the time. Combined with the fact that spatial optimization also always has a solution (Sect. 7.4.1), once a flight corridor is given, a spatial-temporal trajectory must exist.

7.4.3 Local Re-planning Framework

To address the above issues fundamentally, we propose a local re-planning framework which reactively wraps the global trajectory to avoid unmodeled obstacles. A sliding local map is maintained onboard, where obstacles are fused, and an ESDF (Euclidean Signed Distance Field) [22] is updated accordingly. Note that the dense global map is attached to the global pose graph but the local map introduced here is associated with the local VIO frame and sliding with the drone.

ESDF Mapping

We adopt our previous work FIESTA [34], which is an advanced incremental ESDF [73] mapping framework, to build the local map for online re-planning. FIESTA fuses the depth information into a voxel-hashed occupancy map [41] and updates the distance value of voxels as few as possible using a breadth-first search (BFS) framework. It is lightweight, efficient, and produces near-optimal results. Details can be checked in [34]. The ESDF is necessary for the following gradient-based

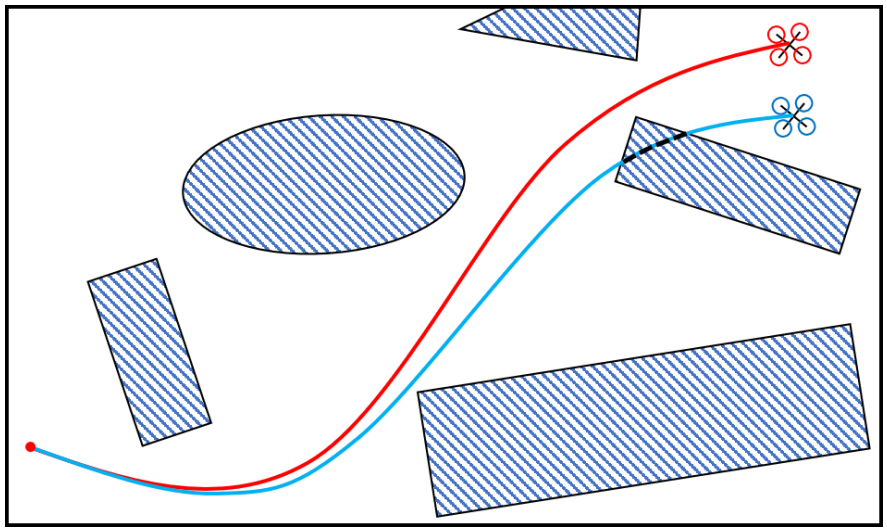


Figure 7.8. An illustration of colliding with obstacles when there are significant pose drifts but no timely loop closure corrections. Obstacles are depicted in the global frame. The flight path of the drone in the VIO frame is shown in the red curve. But the actual trajectory in the global frame is the blue curve, which collides with obstacles on the global map.

trajectory wrapping. An example of a local occupancy map and its corresponding ESDF map are shown in Fig. 7.9. Note, in our system, the range of the local map is decided by the range of current depth observation.

Sliding Window Re-planning

Due to the limited onboard sensing range and computing resources, it is impossible and unnecessary to conduct global re-planning. In this article, we maintain a temporal sliding window over the global trajectory and conduct local re-planning within it. As is shown in Fig. 7.10, when obstacles are observed to block the trajectory in the sliding window, a re-planed trajectory is generated to avoid obstacles, and rejoin the global trajectory afterward.

7.4.4 Gradient-Based B-spline Optimization

B-spline Trajectory Formulation

A B-spline is a piecewise polynomial function defined by a series of control points $\{\mathbf{Q}_0, \mathbf{Q}_1, \cdots, \mathbf{Q}_N\}$ and knot vector $[t_0, t_1, \cdots, t_m]$. For a p-degree B-spline, we have $m = N + p + 1$. Following the matrix representation of the De BoorâĂŞCox formula [13], the value of a B-spline can be evaluated

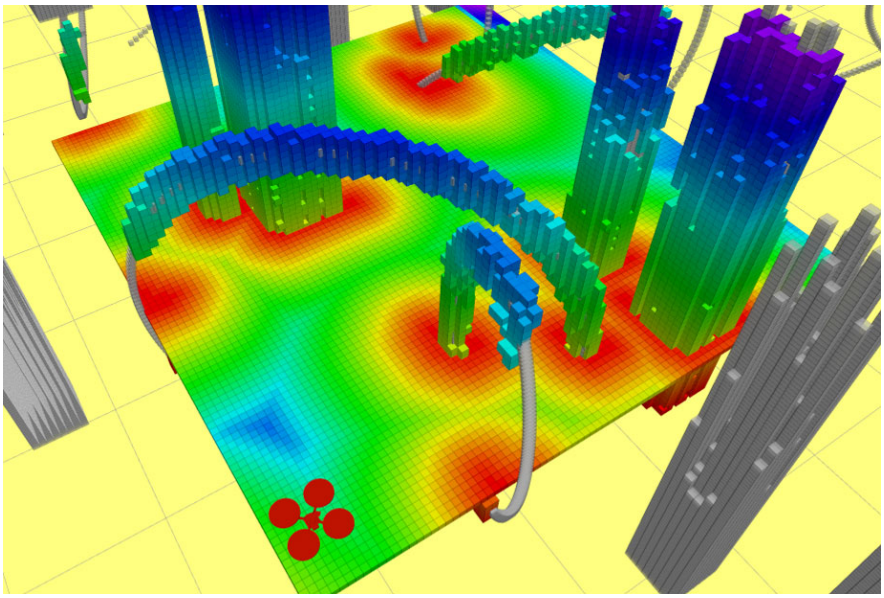


Figure 7.9. The local occupancy map its corresponding ESDF map visualized at a height of $0.6m$.

as:

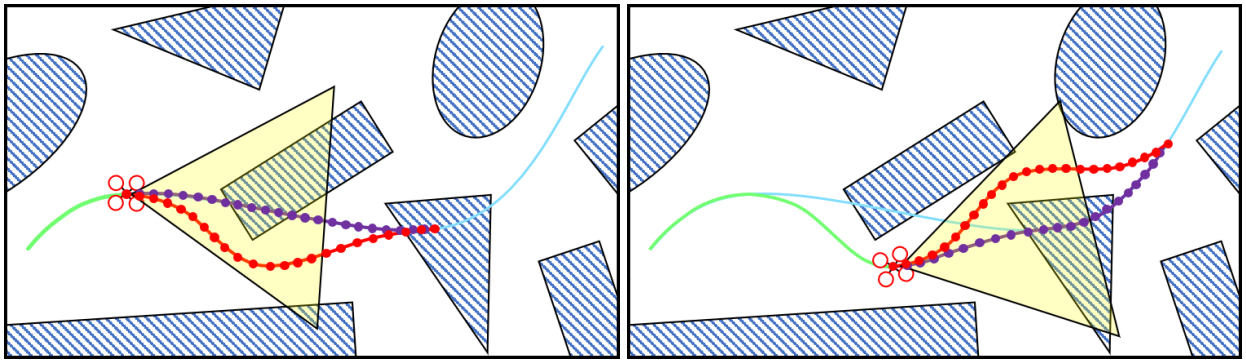
$$P(u) = [1, u, \ldots, u^p] \cdot \mathbf{M}_{p+1} \cdot [\mathbf{Q}_{i-p}, \mathbf{Q}_{i-p+1}, \ldots, \mathbf{Q}_i]^T \tag{7.14}$$

here $\mathbf{M}_{p+1}$ is a constant matrix depends only on p. And $u = (t - t_i)/(t_{i+1} - t_i)$, for $t \in [t_i, t_{i+1})$.

B-spline Initialization

We initialize the local trajectory optimization by re-parameterizing the trajectory in the re-planning horizon as a uniform B-spline. The reason we use uniform B-spline is that it has a simple mathematical formula that is easy to evaluate in the following optimization. For a uniform B-splines, each knot span $\Delta t_i = t_{i+1} - t_i$ has identical value Δt. The local trajectory is first discretized to a set of points according to a given Δt. Then these points are fitted to a uniform B-spline by solving a min-least square problem.

Note that, a p degree uniform B-spline is naturally $p - 1$ order continuous between consecutive spans. Therefore, there is no need to enforce continuity constraints in the following optimization explicitly. Besides, for a p degree B-spline trajectory defined by $N + 1$ control points, the first and last p control points are fixed due to the continuous requirement of the starting and ending states of the local trajectory.



(a) (b)

Figure 7.10. An illustration of the online re-planning mechanism. The blue and green curves are the global trajectory and the actual flight path of the drone, respectively. The purple curve and dots are the global trajectory in the sliding window and its corresponding control points. The red curve and dots are the re-planned local trajectory and its control points. The yellow frustum shows the sensing horizon of the drone.

Elastic Band Optimization

The basic requirements of the re-planed B-spline are three-folds: smoothness, safety, and dynamical feasibility. We define the smoothness cost J_s using a jerk-penalized elastic band cost function[69, 96]:

$$
\begin{aligned}
J_s = & \\
& \sum_{i=1}^{N-1} \| \underbrace{(\mathbf{Q}_{i+2} - 2\mathbf{Q}_{i+1} + \mathbf{Q}_i)}_{\mathbf{F}_{i+1,i}} - \underbrace{(\mathbf{Q}_{i+1} - 2\mathbf{Q}_i + \mathbf{Q}_{i-1})}_{\mathbf{F}_{i-1,i}} \|^2 \\
& = \sum_{i=1}^{N-1} \|\mathbf{Q}_{i+2} - 3\mathbf{Q}_{i+1} + 3\mathbf{Q}_i - \mathbf{Q}_{i-1}\|^2, \qquad (7.15)
\end{aligned}
$$

which can be viewed as a sum of the squared jerk of control points on the B-spline. Note here we use this formulation which is independent of the time parametrization of the trajectory instead of the traditional time-integrated cost function [55]. Because the time duration in each span of the B-spline may be adjusted after the optimization (Sec. 7.4.4), Eq. 7.15 captures the geometric shape of the B-spline regardless of the time parametrization. Besides, Eq. 7.15 bypasses the costly evaluation of the integration and is, therefore, more numerically robust and computationally efficient in optimization.

The safety and dynamical feasibility requirements of the B-spline are enforced as soft con-

straints and added to the cost function. Also, the collision cost J_c, dynamical feasibility cost J_v, and J_a are evaluated at only control points. The collision cost J_c is formulated as the accumulated L2-penalized closest distance to obstacles along the trajectory, which is written as

$$J_c = \sum_{i=p}^{N-p} F_c(d(\mathbf{Q}_i)), \tag{7.16}$$

where $d(\mathbf{Q}_i)$ is the distance between $\mathbf{Q}_i$ to its closet obstacle and is recorded in the ESDF. F_c is defined as

$$F_c(d) = \begin{cases} (d - d_0)^2 & d \leq d_0 \\ 0 & d > d_0 \end{cases} \tag{7.17}$$

where d_0 is the expected path clearance. J_v and J_a are applied to velocities and accelerations, which exceed the physical limits. The formulations of J_v and J_a are similar to Eq. 7.16 and are omitted here. The overall cost function is:

$$J_{total} = \lambda_1 J_s + \lambda_2 J_c + \lambda_3 (J_v + J_a), \tag{7.18}$$

where $\lambda_1, \lambda_2, \lambda_3$ are weighting coefficients. J_{total} can be minimized for a local optimal solution by general optimization methods such as Gauss-Newton or Levenberg-Marquardt.

Iterative Refinement

In the above-unconstrained optimization problem, although collisions and dynamical infeasibilities are penalized, there is no hard guarantee on generating a strictly feasible solution. To improve the success rate in practice, we add a post-process to refine the trajectory iteratively. In each iteration, we check collisions and feasibilities of all optimized control points. If collisions are detected, we increase the collision term J_c by increasing λ_2 and solve the optimization problem (Eq. 7.18) again.

Since we wrap the local trajectory to go around obstacles, the trajectory is always lengthened after the optimization. Consequently, using the original time parametrization will unavoidably result in a higher aggressiveness, which means the quadrotor tends to fly faster. Then its velocity and acceleration would easily exceed the predefined limits. Therefore, we adjust the time parameterization of the local trajectory to squeeze out dynamical infeasibilities. We slightly enlarge infeasible

knots spans of the B-spline by the following heuristic.

$$\Delta t_i^{'} = \min\{\alpha, \max\{\frac{v_m}{v_{max}}, (\frac{a_m}{a_{max}})^{\frac{1}{2}}\}\} \cdot \Delta t_i, \tag{7.19}$$

where α is a constant slightly larger than 1. v_m, a_m are infeasible velocity and acceleration and v_{max}, a_{max} are maximum allowed acceleration and velocity of the drone. The time duration is iteratively enlarged until obtaining a feasible solution or exceeding the maximum iteration limit. If no feasible solution exists after the time adjustment, λ_3 is increased, and the trajectory is optimized again.

7.5 Results

7.5.1 Implementation Details

The global planning method proposed in this paper is implemented with a QP solver OOQP[1] and a SOCP solver Mosek[2]. The local re-planning depends on a nonlinear optimization solver NLopt[3]. The source code of all modules in our quadrotor system, including local/global localization, mapping, and planning, are released as ros-packages[4] for the reference of community. Readers of this paper can easily replicate all the presented results. The state estimation, pose graph optimization, local mapping, local re-planning, and the controller is running onboard on a Manifold-2C[5] mini-computer. Other modules are running on an off-board laptop which has a GTX 1080[6] graphics card.

Our global map is built to attach to a global pose graph. Both the map and the pose graph are saved for repeating. Before the repeating, the drone is handheld to close the loop of the current VIO frame with the saved global pose graph. The relative transformation of these two frames is used to project the control commands to the VIO frame. During the repeating, pose graph optimization is also activated to calculate the pose drift and compensate for the control command.

[1] http://pages.cs.wisc.edu/~swright/ooqp/

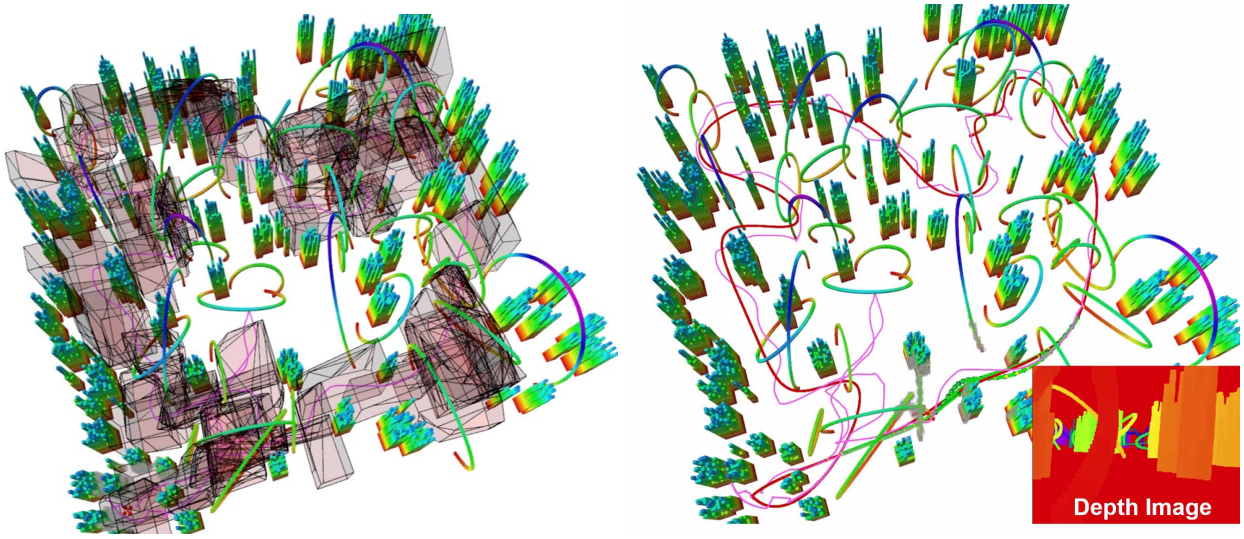
[2] https://www.mosek.com

[3] https://nlopt.readthedocs.io

[4] https://github.com/HKUST-Aerial-Robotics/Teach-Repeat-Replan

[5] https://store.dji.com/product/manifold-2?vid=80932

[6] https://www.nvidia.com/en-us/geforce/20-series/



(a) The flight corridor catpures local free space.

(b) The local re-planning trajectory and current depth image.

Figure 7.11. The trajectory generated in a complex simulated environment. The flight corridor consists of large free convex polyhedrons are shown in (a), and the optimized space-time trajectory is shown in (b).

7.5.2 Simulated Flight Test

We first test our global and local planning methods in simulations. The simulated environments are randomly deployed with various types of obstacles and circles for drone racing, as shown in Fig. 7.11. The simulating tool we use is a light-weight simulator MockaFly[7], which contains quadrotor dynamics model, controller, and map generator. And the simulator is also released as an open-source package with this paper. In the simulation, a drone is controlled by a joystick to demonstrate the teaching trajectory. The simulated drone is equipped with a depth camera whose depth measurements are real-time rendered in GPU by back-projecting the drone's surrounding obstacles. We randomly add noise on the depth measurements to mimic a real sensor. The re-planning module is activated in the simulation and is triggered by the noise added on the depth. The teaching trajectory and the flight corridor is shown in Fig. 7.11(a). The global trajectory, local re-planned trajectory, and depth measurement are shown in Fig. 7.11(b). More details about the simulation are presented in the attached video.

[7] https://github.com/HKUST-Aerial-Robotics/mockasimulator

Table 7.1. Comparison of Computing Time of Corridor Generation

Computing Time (s)	GPU	CPU++	CPU+	CPU_raw
Res. = $0.25m$	0.031	0.111	0.162	0.359
Res. = $0.20m$	0.055	0.310	0.503	1.309
Res. = $0.15m$	0.169	1.423	2.803	9.583
Res. = $0.10m$	0.942	13.940	30.747	141.659
Res. = $0.075m$	3.660	71.862	157.181	927.131

7.5.3 Benchmark Comparisons

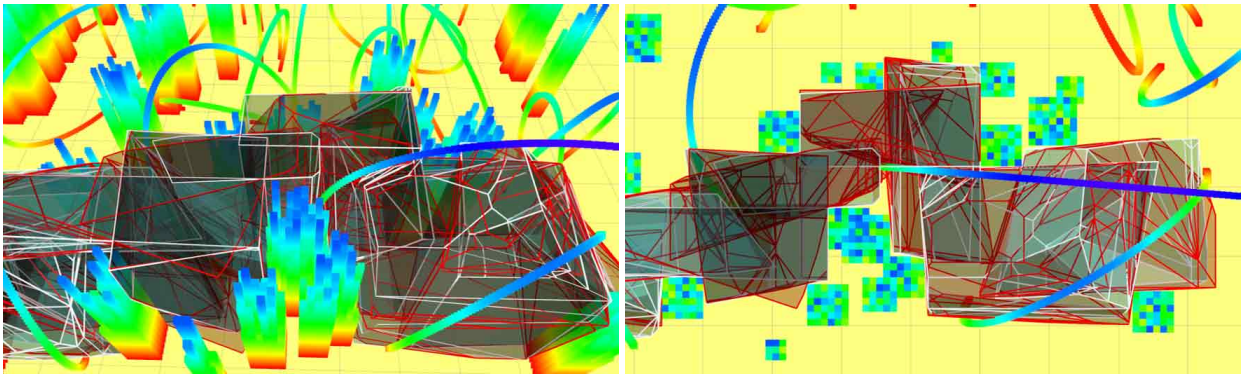
Corridor Generation

We test the performance of the flight corridor generation methods (Sect. 7.3), to show the efficacy of the proposed techniques for CPU (Sect. 7.3.2) and GPU (Sect. 7.3.3) accelerations. For convenience, we denote the basic process for doing *convex cluster inflation* as CPU_raw; CPU_raw added cube initialization as CPU+; the one with cube initialization, vertex selection and early termination as CPU++; and the parallel version of the *convex cluster inflation* as GPU. We first compare the time consumed for finding the largest flight corridor with these methods, to validate the improvements of efficiency by using our proposed CPU and GPU acceleration techniques. Then, we compare the ratio of space capturing by methods with and without the polyhedron initialization, and by our previous method [30]. The motivation of the latter comparison is two-fold:

1. It serves to show superior performance by replacing cubes with polyhedrons.
2. As discussed in Sect. 7.3.2, the initialization process would result in different final clustering results compared to the pure *convex cluster inflation*. This comparison also validates that the initialization process only does neglectable harm to free space capturing.

We generate ten random maps, with $10 \sim 20$ random teaching trajectories given in each map. The average length of teaching trajectories is $20m$. Results are given in Tabs. 7.1 and 7.2.

As shown in Tab. 7.1, as the resolution of the map being finer, the computing time of the simple *convex cluster inflation* quickly becomes unacceptable huge. In CPU, with the help of *polyhedron initialization*, the computational efficiency is improved several times. Moreover, according to Tab. 7.1, introducing the *voxel selection* and *early termination* can increase the speed more than



(a) (b)

Figure 7.12. The flight corridor generated with and without the initialization process. Polyhedrons with bounding edges in white and red are found by methods with and without the initialization, respectively.

Table 7.2. Comparison of Space Captured of Corridor Generation

Space Ratio (%)	w/ Initialization	w/o Initialization	Previous [30]
Res. = $0.25m$	99.22	100.00	82.28
Res. = $0.20m$	99.56	100.00	82.92
Res. = $0.15m$	98.93	100.00	81.82
Res. = $0.10m$	97.06	100.00	82.78
Res. = $0.075m$	97.14	100.00	83.03

one order of magnitude in a fine resolution. The efficacy of the GPU acceleration is even more significant. As shown in Tab. 7.1, the GPU version improves the computing speed 30 times at a fine resolution ($0.075m$), and 10 times at a coarse resolution ($0.25m$). For a finer resolution, more candidate voxels are discovered in one iteration of Alg. 7, thus more computations are conducted parallelly to save time.

For the second comparison, we count the number of free voxels included in the flight corridor found by each method. At each resolution, we take the result of the method without initialization as 100% and compare others against it. Tab. 7.2 indicates two conclusions:

1. Using polyhedrons instead of axis-aligned cubes can significantly increase the volume of the flight corridor.

2. Using initialization only slightly sacrifices the volume of the flight corridor. And the sacrifice is neglectable in a medium or coarse resolution ($0.15 \sim 0.25m$).

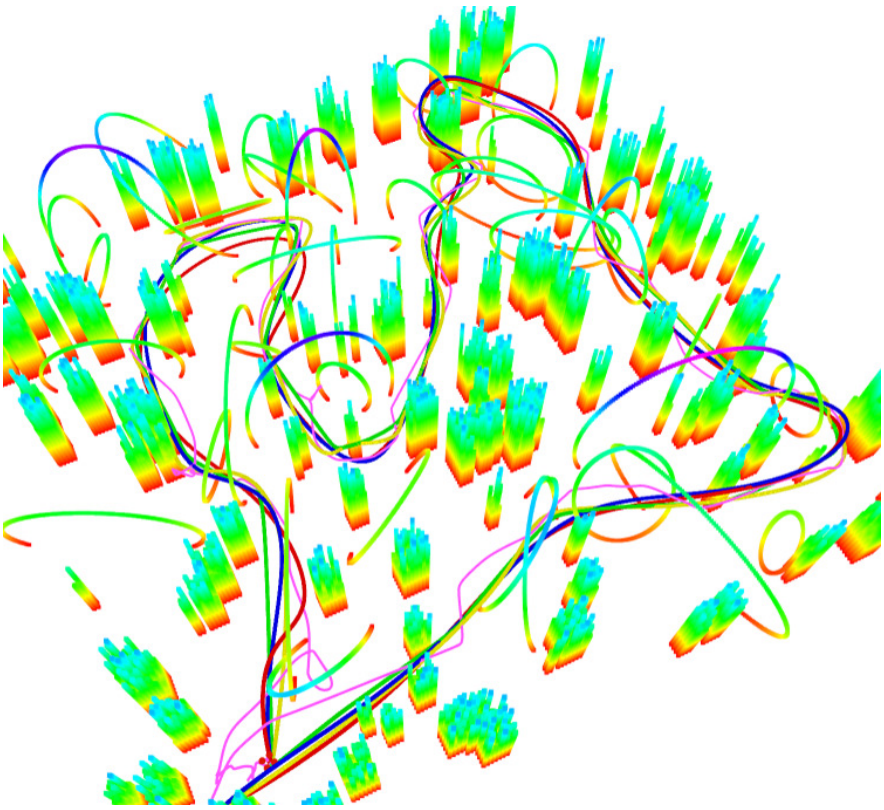


Figure 7.13. The comparison of trajectories optimized by different methods. The manual flight trajectory is shown as the purple curve. Blue, red, green, and yellow trajectories are generated by our proposed method, our previous method [30], gradient-based method [28] and waypoints-based method [71].

The first conclusion holds because a simple cube only discovers free space in x, y, z directions and sacrifices much space in a highly non-convex environment, as in Fig. 7.3. The second conclusion comes from the fact that in a highly non-convex environment, a regular shaped polyhedron (a cube) does not prevent the following voxel clustering in its nearby space. It shows that the initialization plus the clustering refinement does not harm the volume of the final polyhedron, and is acceptable in practice, especially for a resolution not very fine.

Global Planning

We compare the proposed global planning method against our previous work [30] and another representative optimization-based trajectory generation methods, such as the waypoint-based method [71] and the gradient-based method [28]. For the latter two benchmarked methods, there is no explicit way to capture the topological structure of the teaching trajectory. Therefore, we convert the teaching trajectory to a piecewise path by recursively finding a collision-free straight line path along

Table 7.3. Comparison of Trajectory Optimization

Method	Length (m)	Time (s)	Energy $((m/s^3)^2)$
Proposed Method	**84.607**	**55.154**	**83.350**
Previous Method	86.723	57.736	89.883
Gradient-based [28]	89.622	111.398	109.575
Waypoint-based [71]	97.045	94.895	204.267

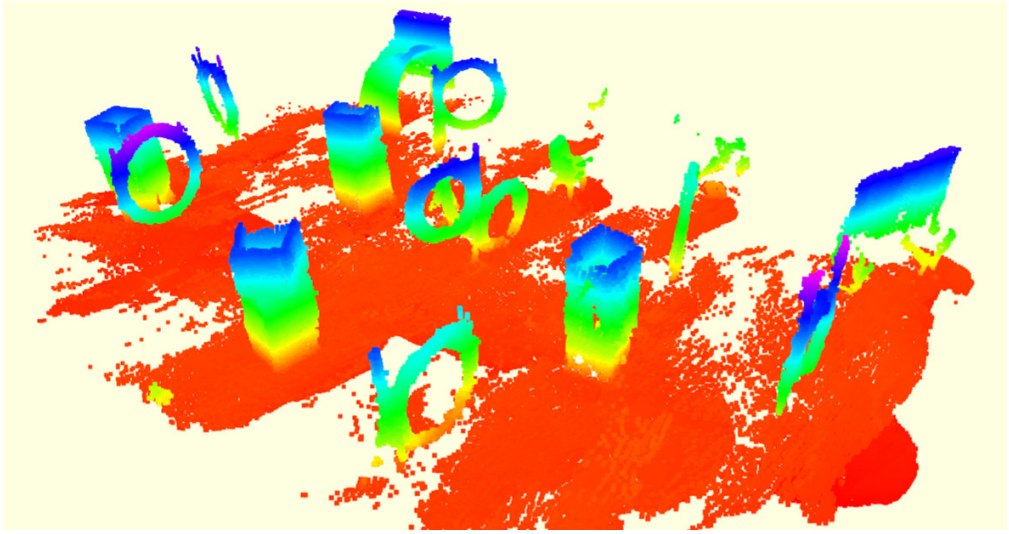
with it. Then we use this path to initialize the waypoint-based [71] method and the gradient-based method [28]. Benchmarked methods are also integrated into the coordinate descent framework with temporal optimization. Some parameters dominate the performance of these benchmarked methods, especially for the gradient-based method [28] where the trade-off between collision and smoothness is essential. For a fair comparison, parameters are tuned to achieve the best performances before the test. We randomly generate ten simulated environments with dense obstacles, as in Sect. 7.5.2, and conduct ten teach-and-repeat trials in each map. A sample result of generated trajectories is shown in Fig. 7.13.

As shown in Tab. 7.3, our proposed method outperforms in all length, time, and energy aspects. The waypoint-based [71] method and the gradient-based [28] method both require a piecewise linear path as initialization. The waypoint-based [71] method can only add intermediate waypoints on the initial path. Therefore, it is mostly dominated by its initialization and tends to output a solution with low quality. The gradient-based [28] method has no such restriction and can adjust the path automatically by utilizing gradient information. However, its optimization formulation is underlying non-convex, since the collision cost is defined on a non-convex ESDF. Therefore, the gradient-based [28] method always finds a locally optimal solution around its initial guess. Compared to these two methods, our method is initialization-free. Both the spatial and temporal optimization of our proposed method enjoys the convexity in its formulation. They are guaranteed to find the global energy-optimal and time-optimal solutions in the flight corridor. Naturally, a smoother trajectory also tends to generate a faster time profile. So finally, under the same coordinate descent framework, our method always outperforms [28] and [71]. Compared to to [30], the advanced corridor generation proposed in this paper (Sect. 7.3) can always capture more free space than using our previous axis-aligned corridor. Naturally, it provides much more freedom for global planning and results in much better solutions.

7.5.4 Indoor Flight Test

Fast Flight in a Static Environment

(a) The indoor testing environment.

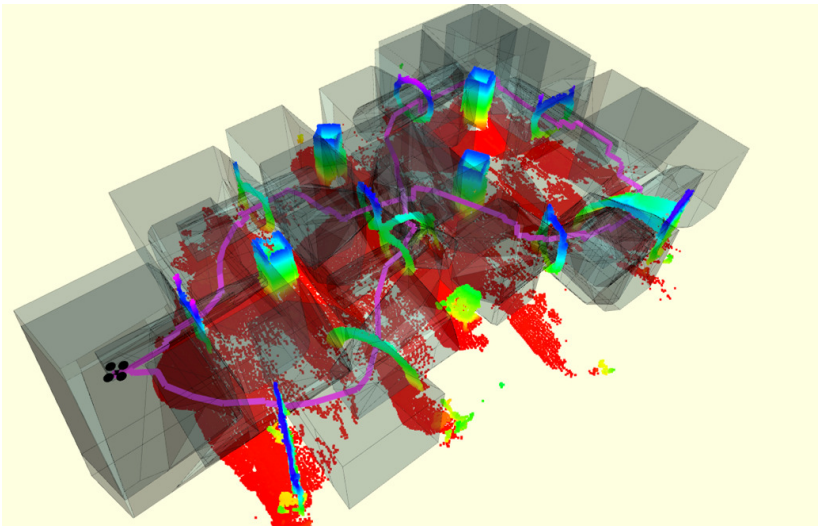


(b) The global consistent dense map.

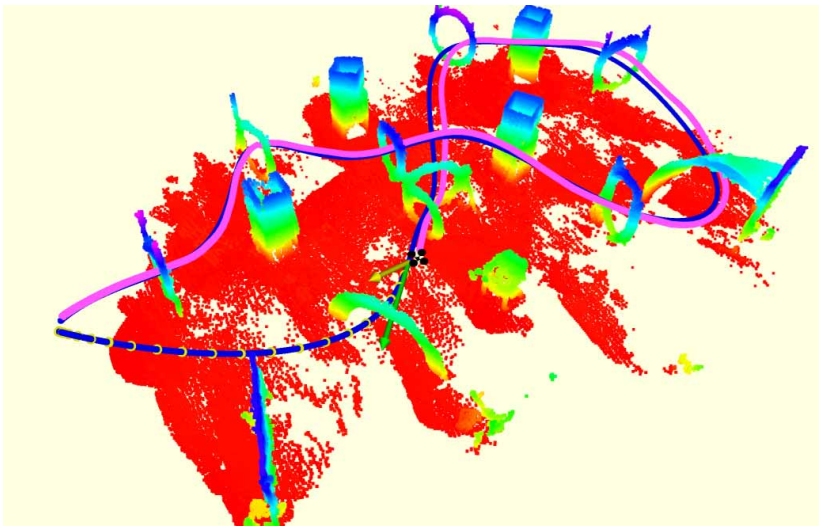
Figure 7.14. The experimental set-up of the fast indoor drone racing flights. (a), the obstacles deployment. (b), the pre-built globally consistent map.

Firstly, we conduct experiments in a cluttered drone racing scenario. This experiment validates the robustness of our proposed system, and also pushes the boundary of aggressive flight of quadrotors. Several different types of obstacles, including circles, arches, and tunnels, are deployed randomly to composite a complex environment, as shown in Fig. 7.14(a). The smallest circle only has a diameter of $0.6m$, which is very narrow compared to the $0.3m \times 0.3m$ tip-to-tip size of our drone.

The maximum velocity and acceleration of the drone are set as $3m/s$ and $3m/s^2$, respectively. And the parameter ρ in Equ. 7.12 is set as 0, which means the quadrotor is expected to fly as fast as possible as long as it respects the kinodynamic limits.



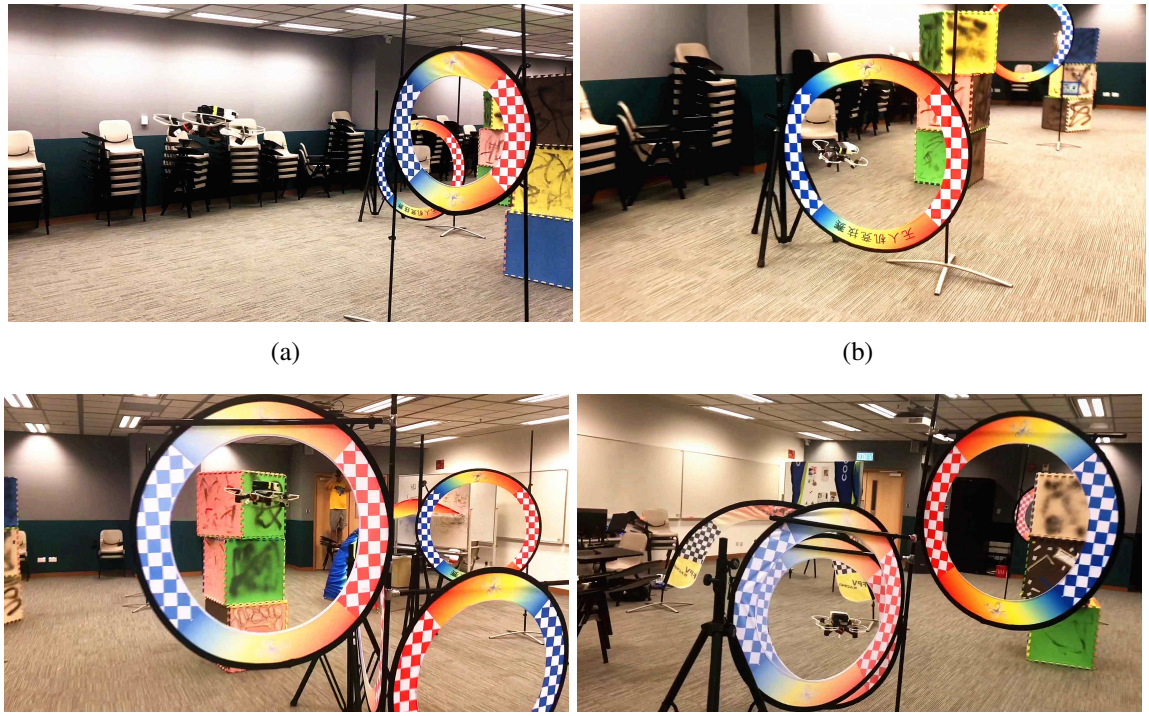
(a) The teaching trajectory and the flight corridor.



(b) The spatial-temporal optimal repeating trajectory.

Figure 7.15. An overview of the teaching trajectory, flight corridor, and repeating trajectory in a pre-built dense map. The colored code indicates the height of obstacles. The flight corridor is represented by transparent blue polyhedrons in (a). The global trajectory, local trajectory, quadrotor flight path are shown in blue, green, and purple curves, respectively.

A dense global consistent map is pre-built using the method stated in Sect. 7.2.2. During the teaching phase, the quadrotor is virtually piloted by a human to move amid obstacles. Then the quadrotor autonomously converts this teaching trajectory to a global repeating trajectory and starts



(a) (b)

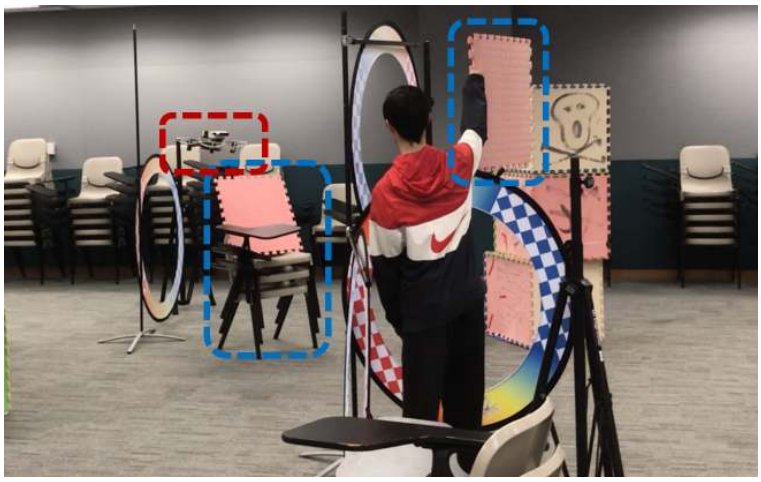
(c) (d)

Figure 7.16. Snapshots of the fast autonomous flight in a static indoor environment. The maximum velocity and acceleration for the quadrotor are set as $3m/s$ and $3m/s^2$.

to track it. Snapshots of the drone in the flight are shown in Fig. 7.16. The teaching trajectory and the convex safe flight corridor are visualized in Fig 7.15(a). And the global repeating trajectory is in Fig. 7.15(b).

Local Re-planning Against Unknown Obstacles

Our system can deal with changing environments and moving obstacles. In this experiment, we test our system also in the drone racing site to validate our local re-planning module. Several obstacles are moved or added to change the drone racing environment significantly, and some others are dynamically added during the repeating flight, as shown in Fig. 7.17.



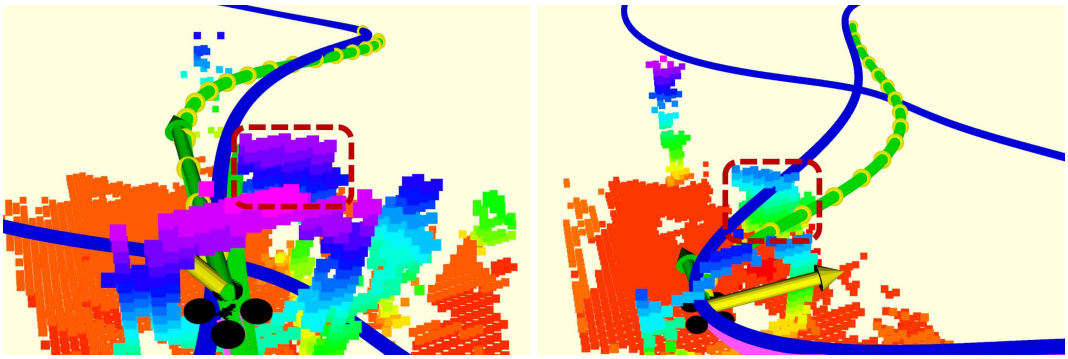
(a) Side-view.

(b) Onboard first-person view.

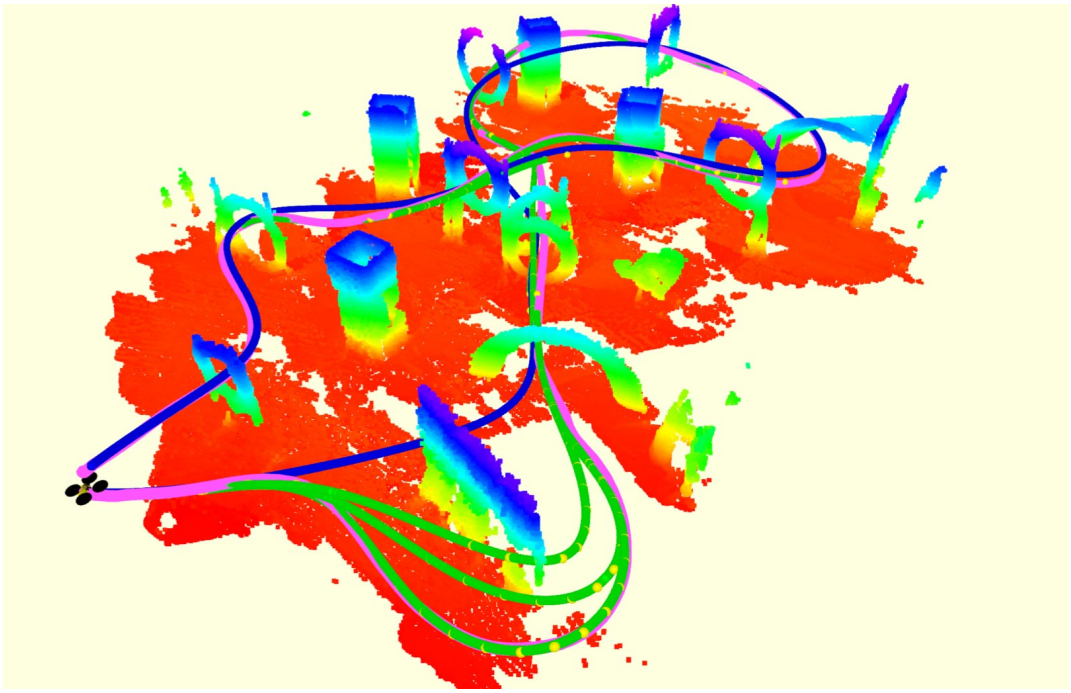
Figure 7.17. The local re-planning experiment against unmapped and moving obstacles. The drone and unmapped obstacles are labeled by the red and blue dashed rectangles, respectively, for clear visualization.

In this experiment, the maximum velocity and acceleration for the quadrotor are set as $2m/s$ and $2m/s^2$. The local ESDF map is sliding with the drone using a ring-buffered updating mechanism [83]. The resolution of the local perception is $0.075m$. The size of the map is decided by points observed spreading in the current frame. The horizon and frequency of the local re-planning are $3.5s$ and $15Hz$, respectively. Re-planning is triggered eight times during the flight in this experiment, and local safe and dynamical feasible splines are generated on time accordingly. Local trajectories, local maps, and the overview of this experiment are shown in Fig. 7.18. We refer

readers to the attached video for more details.



(a) Re-planning, unmapped obstacle. (b) Re-planning, moving obstacle.



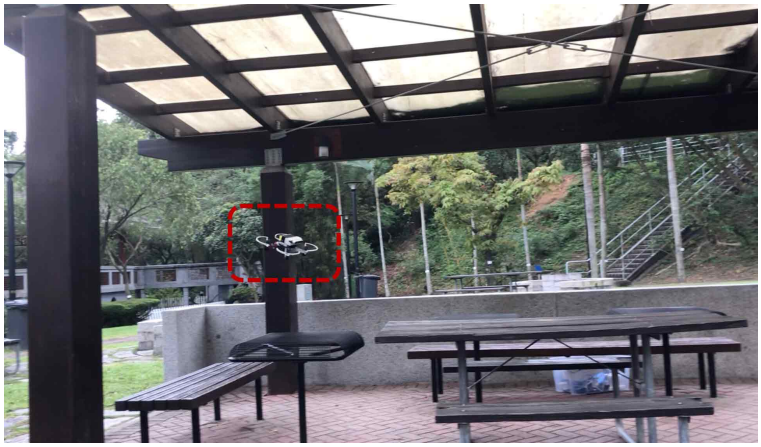
(c) Overview of all re-planning trajectories.

Figure 7.18. Indoor flight in a dynamic environment. In (a) and (b), the unmapped new obstacle and moving obstacles are labeled by red dashed rectangles, and colored voxels represent local obstacle maps. In (c), colored voxels show the global map. Other marks are interpreted as the same as in previous figures.

7.5.5 Outdoor Flight Test

Finally, we conduct quadrotor flight experiments with a much higher aggressiveness in two different outdoor scenes, as in Fig. 7.19, to show the robustness of our system in natural environments.

Although these experiments are conducted outdoor, GPS or other external positioning devices are not used. The teach-repeat-replan pipeline is as the same as before indoor experiments 7.5.4. The maximum allowed velocity and acceleration limits for these two trials are set as $5m/s$, $6m/s^2$ and $7m/s$, $6m/s^2$, respectively. The drone's desired and estimated positions and velocities in the second trial are given in Fig. 7.22, which shows acceptable tracking errors. Since the flight speed is significantly higher than indoor experiments, we set a smaller re-planning horizon as $2.0s$. Results such as the global and local trajectory and the global map are visualized in Figs. 7.20 and 7.21. More clearly visualizations of outdoor experiments are given in the video[8].



(a) Outdoor experimet, trial 1.

(b) Outdoor experimet, trial 2.

Figure 7.19. Snapshots of the experiments in outdoor environments.

[8] https://www.youtube.com/watch?v=urEC2AAGEDs&t=46s

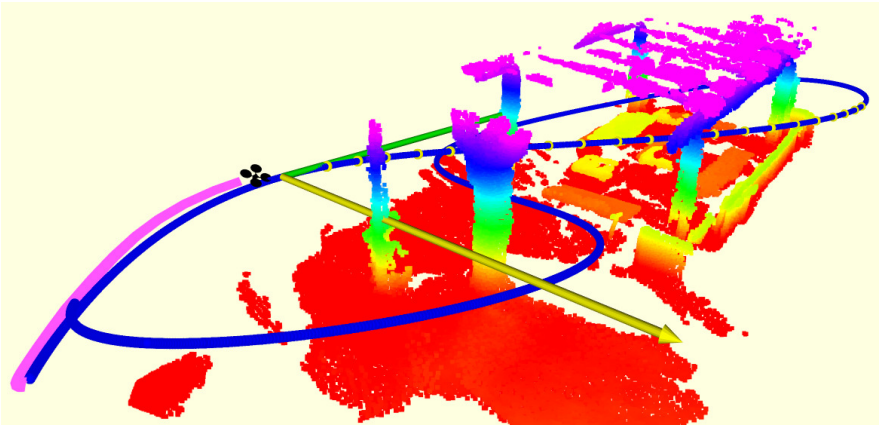
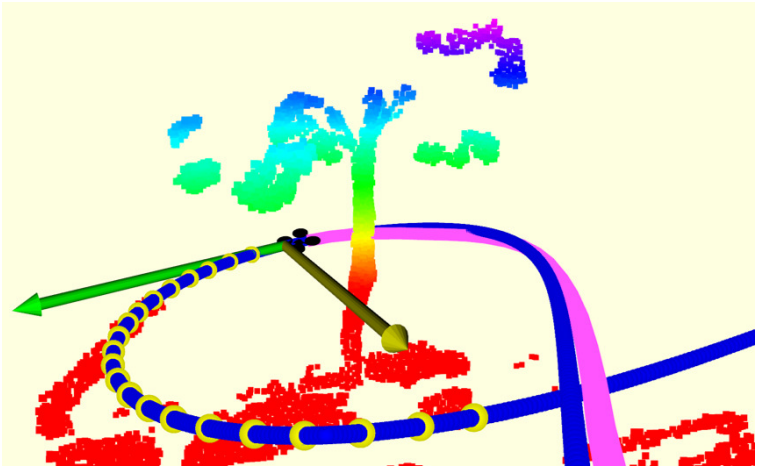
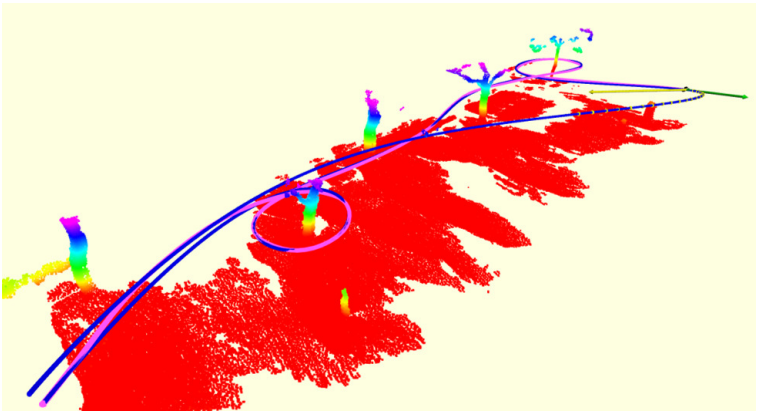


Figure 7.20. The repeating trajectory in outdoor experiments, trial 1.



(a) A close-up of outdoor experiments, trial 2.



(b) An overview of outdoor experiments, trial 2.

Figure 7.21. Outdoor flight, trial 2. Marks are interpreted as the same as in previous figures.

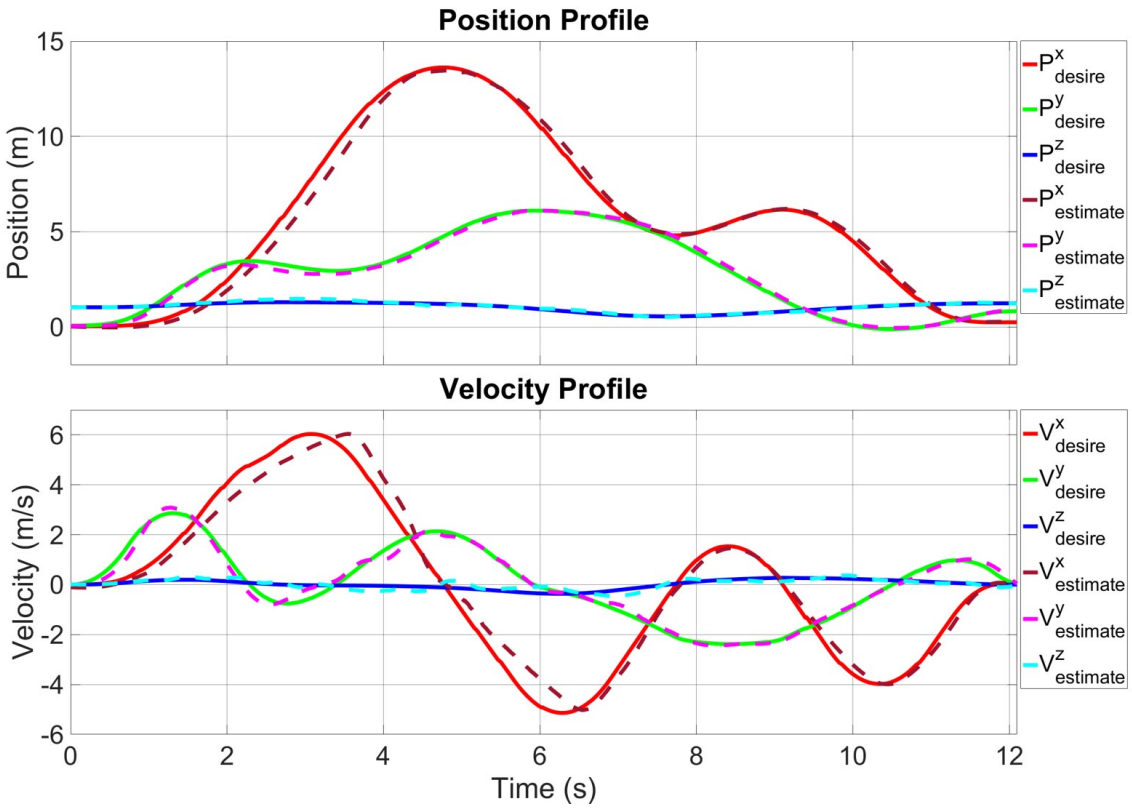


Figure 7.22. Profiles of the desired and estimated position and velocity. The position and velocity are estimated by our localization module VINS [68].

7.6 Conclusion

In this chapter, we propose a framework, *teach-repeat-replan* for quadrotor aggressive flights in complex environments. The main idea of this work is to find the topological equivalent free space of the user's teaching trajectory, use spatial-temporal trajectory optimization to obtain an energy-efficient repeating trajectory, and incorporate online perception and re-planning to ensure the safety against environmental changes and moving obstacles. The teaching process is conducted by virtually controlling the drone in simulation. The generated repeating trajectory captures users' intention and respect an expected flight aggressiveness, which enables autonomous flights much more aggressive than human's piloting in complex environments. The online re-planning guarantees the safety of the flight and also respects the reference of the repeating trajectory.

To group large free space around the teaching trajectory, we propose a GPU-accelerated convex polyhedron clustering method to find a flight corridor. The optimal global trajectory generation problem is decoupled as spatial and temporal sub-problems. Then these two sub-problems are

iteratively optimized under the coordinate descent framework. Moreover, we incorporate the local perception and local trajectory re-planning modules into our framework to deal with environmental changes, dynamic obstacles, and localization drifts.

The proposed system is complete and robust. Users of our system do not have to pilot the drone carefully to give a teaching trajectory. Instead, an arbitrarily jerky/poor trajectory can be converted to an efficient and safe global trajectory. Moreover, when the environment changes or the global localization drifts, the local perception and re-planning modules guarantee the safety of the drone while tracking the global trajectory. Our system is also flexible and easily replicable, as evidenced by various types of experiments presented in this paper, and a third-party application[9]. We release all components of our system for the reference of the community.

[9]Flight demonstration at the Electrical and Mechanical Services Department (EMSD), Hong Kong government. Video: `https://youtu.be/Ut8WT0BURrM`

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Thesis Summary

In this thesis, we present contributions to the state-of-the-art in autonomous navigation of MAVs in complex indoor and outdoor environments, with a focus on motion planning. In Chap. 3, we introduce the local trajectory optimization method by using gradient information in complex environments. Then in Chap. 4, trajectory optimization method with hard constraints on safety and kinodynamic feasibility is presented. In Chap. 5, we present a complete online MAV motion planning framework that directly operates on point clouds and doesn't rely on any post-processed maps. Sampling-based asymptotical path finding method on point clouds is detailed, followed by an optimization-based trajectory generation approach. Then in Chap. 6, we investigate the problem of spatial-temporal decoupled planning and propose a method to optimize the time profile of a given spatial curve. Finally, in this thesis, we integrate all necessary components to build a complete and robust MAV platform, include local and global planning, perception, localization, and control. All methods shown in this thesis are developed based on robotic aerial systems, and target for realistic industrial/field applications. However, the algorithms presented are general motion planning methods and are not limited by the aerial platforms. We have shown the extensions and applications of our methods in ground robots [77, 87], and will distribute these approaches at more scenarios in the future.

8.2 Future Works

In this thesis, several problems related to motion planning and its applications are studied, and primary solutions are given accordingly. More cutting-edge topics that may worth research are also revealed by these studies. In what follows, we briefly state our plan for the next research directions.

8.2.1 Fast Planning in Dynamic Environments

Most research reported in this thesis is conducted in static environments, although the environment can be extremely cluttered, and our method is robust against moving obstacles, these algorithms are not specially designed for dynamic environments. As the development of algorithms and hardware of vision processing, 3D detection, tracking, and prediction, as well as the dense mapping in fully dynamic environments, are quickly becoming possible towards a robust stage. We will extend research in this thesis to dynamic environments.

8.2.2 Distributed Motion Planning for Multi-agents

Also, as the development of onboard computing and multi-agent communication, multi-drone (swarm) is becoming popular in recent years. However, most reported works are conducted in simple indoor scenarios with an external positioning system (Motion capture), or in sparse outdoor environments with GPS. We plan to develop a fully autonomous and robust swarm platform that can be deployed in field applications, such as the forest or other complicated situations, without the help of man-made structures or external supports.

8.2.3 Human-Drone Cooperation

Human-drone interactions are considered in the last work presented in this thesis. However, as the same as most existing human-drone interactive applications, the human's role is appointing the drone with his desirable preference, instead of cooperation with the drone himself. In the future, we plan to develop a complete system; for applications such as indoor exploration and search-and-rescue, drones can actively cooperate with the human with a high-level mission target. And the safety of both the human and the robot should be guaranteed,

8.2.4 Motion Planning for Morphing Drones

Morphing robot, as well as the aerial robot, is a very hot topic at this time. Now, most research focus on developing the robotic prototype, without the functionality of full autonomy. We plan to design motion planning algorithms to adapt the morphing capability of such drones and expand the area for applying advanced robotic designs.

8.2.5 Model Adaptation and Environmental Inference

A straightforward problem always exists throughout the research presented in this thesis. For different platforms and different testing environments, such as a dense forest, or a simple indoor case, different parameter settings are handcrafted by researchers. However, the dynamic limits and the expected flight aggressiveness should not be decided by the researcher/operator, but by the drone's dynamic limitation itself and specific applications. In the future, we expect the drone to automatically adjust its aggressiveness based on the physical limits, and also the inference of the crash uncertainty of the environment.

REFERENCES

[1] P. Aditya A, M. Kevin C, S. Xichen, C. Soon-Jo, and H. Seth. Motion primitives and 3d path planning for fast flight through a forest. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, pages 2940–2947. IEEE, November 2013.

[2] Erling D. Andersen. On formulating quadratic functions in optimization models. *Technical Report TR-1-2013, MOSEK ApS*, 2013. URL: `http://docs.mosek.com/whitepapers/qmodel.pdf`.

[3] C Bradford Barber, David P Dobkin, David P Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software (TOMS)*, 22(4):469–483, 1996.

[4] Andrew J Barry, Peter R Florence, and Russ Tedrake. High-speed autonomous obstacle avoidance with pushbroom stereo. *Journal of Field Robotics*, 35(1):52–68, 2018.

[5] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[6] Laszlo-Peter Berczi and Timothy D Barfoot. It's like déjà vu all over again: Learning place-dependent terrain assessment for visual teach and repeat. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, pages 3973–3980, 2016.

[7] Fabian Blochliger, Marius Fehr, Marcin Dymczyk, Thomas Schneider, and Rol Siegwart. Topomap: Topological mapping and navigation based on visual slam maps. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, pages 1–9, 2018.

[8] Michael Bloesch, Sammy Omari, Marco Hutter, and Roland Siegwart. Robust visual inertial odometry using a direct ekf-based approach. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 298–304. IEEE, 2015.

[9] Adam Bry and Nicholas Roy. Rapidly-exploring random belief trees for motion planning under uncertainty. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, pages 723–730, Shanghai, China, May 2011.

[10] Leobardo Campos-Macías, David Gómez-Gutiérrez, Rodrigo Aldana-López, Rafael de la Guardia, and José I Parra-Vilchis. A hybrid method for online trajectory planning of mobile robots in cluttered environments. *IEEE Robotics and Automation Letters (RA-L)*, 2(2):935–942, 2017.

[11] J. Chen, T. Liu, and S. Shen. Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, pages 1476–1483, Stockholm, Sweden, May 2016.

[12] Sunglok Choi, Jaehyun Park, Eulgyoon Lim, and Wonpil Yu. Global path planning on uneven elevation maps. In *Proc. of the IEEE Intl. Conf. on Ubiquitous Robot. and Ambient Intelligence*, pages 49–54, Daejeon, Korea, Nov. 2012.

[13] Carl de Boor. Subroutine package for calculating with b-splines. Technical report, Los Alamos Scientific Lab., N. Mex., 1971.

[14] R. Deits and R. Tedrake. Computing large convex regions of obstacle-free space through semidefinite programming. In *Algorithmic Foundations of Robotics XI*, volume 107, pages 109–124. Springer, 2015.

[15] R. Deits and R. Tedrake. Efficient mixed-integer planning for UAVs in cluttered environments. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, pages 42–49, Seattle, Washington, USA, May 2015. IEEE.

[16] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501, 2010.

[17] David Droeschel and Sven Behnke. Efficient continuous-time slam for 3d lidar-based online mapping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[18] David Droeschel, Matthias Nieuwenhuisen, Marius Beul, Dirk Holz, Jörg Stückler, and Sven Behnke. Multilayered mapping and navigation for autonomous micro aerial vehicles. *Journal of Field Robotics*, 33(4):451–475, 2016.

[19] Alberto Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[20] Zheng Fang, Shichao Yang, Sezal Jain, Geetesh Dubey, Stephan Roth, Silvio Maeta, Stephen Nuske, Yu Zhang, and Sebastian Scherer. Robust autonomous flight in constrained and visually degraded shipboard environments. *J. Field Robot. (JFR)*, 34(1):25–52, 2017.

[21] Marius Fehr, Thomas Schneider, Marcin Dymczyk, Jürgen Sturm, and Roland Siegwart. Visual-inertial teach and repeat for aerial inspection. *arXiv preprint arXiv:1803.09650*, 2018.

[22] Pedro F Felzenszwalb and Daniel P Huttenlocher. Distance transforms of sampled functions. *Theory of computing*, 8(1):415–428, 2012.

[23] Melvin E Flores. *Real-time trajectory generation for constrained nonlinear dynamical systems using non-uniform rational B-spline basis functions*. California Institute of Technology, 2008.

[24] Komei Fukuda and Alain Prodon. Double description method revisited. In *Franco-Japanese and Franco-Chinese Conference on Combinatorics and Computer Science*, pages 91–111. Springer, 1995.

[25] P Furgale, P Krüsi, F Pomerleau, U Schwesinger, F Colas, and R Siegwart. There and back again-dealing with highly-dynamic scenes and long-term change during topological/metric route following. In *ICRA14 Workshop on Modelling, Estimation, Perception, and Control of All Terrain Mobile Robots*, 2014.

[26] Paul Furgale and Timothy D Barfoot. Visual teach and repeat for long-range rover autonomy. *Journal of Field Robotics*, 27(5):534–560, 2010.

[27] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, pages 2997–3004, Chicago, IL, Sept 2014.

[28] F. Gao, Y. Lin, and S. Shen. Gradient-based online safe trajectory generation for quadrotor flight in complex environments. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, pages 3681–3688, Sept 2017.

[29] Fei Gao and Shaojie Shen. Online quadrotor trajectory generation and autonomous navigation on point clouds. In *Proc. of the IEEE Intl. Sym. on Safety, Security, and Rescue Robotics (SSRR)*, pages 139–146, lausanne, switzerland, 2016.

[30] Fei Gao, Luqi Wang, Kaixuan Wang, William Wu, Boyu Zhou, Luxin Han, and Shaojie Shen. Optimal trajectory generation for quadrotor teach-and-repeat. *IEEE Robotics and Automation Letters (RA-L)*, 2019.

[31] Fei Gao, William Wu, Wenliang Gao, and Shaojie Shen. Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments. *Journal of Field Robotics*, 2018.

[32] Fei Gao, William Wu, Yi Lin, and Shaojie Shen. Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, Brisbane, Australia, May 2018.

[33] Wenliang Gao and Shaojie Shen. Dual-fisheye omnidirectional stereo. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, pages 6715–6722, 2017.

[34] Luxin Han, Fei Gao, Boyu Zhou, and Shaojie Shen. Fiesta: Fast incremental euclidean distance fields for online motion planning of aerial robots. *arXiv preprint arXiv:1903.02144*, 2019.

[35] Daniel Damir Harabor, Alban Grastien, et al. Online graph pruning for pathfinding on grid maps. In *AAAI*, 2011.

[36] Kris Hauser. Lazy collision checking in asymptotically-optimal motion planning. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, pages 2951–2957, 2015.

[37] Jonathan Jamieson and James Biggs. Near minimum-time trajectories for quadrotor uavs in complex environments. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, pages 1550–1555, 2016.

[38] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *2011 IEEE international conference on robotics and automation*, pages 4569–4574. IEEE, 2011.

[39] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30:846–894, 2011.

[40] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the RRT. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, pages 1478–1483, 2011.

[41] Matthew Klingensmith, Ivan Dryanovski, Siddhartha Srinivasa, and Jizhong Xiao. Chisel: Real time large scale 3d reconstruction onboard a mobile device using spatially hashed signed distance fields. In *Proc. of Robot.: Sci. and Syst. (RSS)*, 2015.

[42] Philipp Krüsi, Bastian Bücheler, François Pomerleau, Ulrich Schwesinger, Roland Siegwart, and Paul Furgale. Lighting-invariant adaptive route following using iterative closest point matching. *Journal of Field Robotics*, 32(4):534–564, 2015.

[43] S.M. LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[44] Steven M LaValle. *Planning algorithms*. Cambridge uni. press, 2006.

[45] Steven Lay. *Convex sets and their applications*. Courier Corporation, 2007.

[46] Taeyoung Lee, Melvin Leoky, and N Harris McClamroch. Geometric tracking control of a quadrotor uav on se (3). In *Proc. of the IEEE Control and Decision Conf. (CDC)*, pages 5420–5425, 2010.

[47] Taeyoung Lee, Melvin Leoky, and N Harris McClamroch. Geometric tracking control of a quadrotor uav on se (3). In *Proc. of the IEEE Control and Decision Conf. (CDC)*, pages 5420–5425, Atlanta, GA, Dec. 2010.

[48] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945, 2009.

[49] Maxim Likhachev, David I Ferguson, Geoffrey J Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *ICAPS*, pages 262–271, 2005.

[50] Maxim Likhachev, Geoffrey J Gordon, and Sebastian Thrun. Ara*: Anytime a* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems*, pages 767–774, 2004.

[51] Yi Lin, Fei Gao, Tong Qin, Wenliang Gao, Tianbo Liu, William Wu, and Shaojie Shen. Autonomous aerial navigation using monocular visual-inertial fusion. *Journal of Field Robotics*, 35(1):23–51, 2018.

[52] S. Liu, N. Atanasov, K. Mohta, and V. Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, pages 2872–2879, Sept 2017.

[53] Sikang Liu, Michael Watterson, Kartik Mohta, Ke Sun, Subhrajit Bhattacharya, Camillo Jose Taylor, and Vijay Kumar. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *IEEE Robotics and Automation Letters (RA-L)*, pages 1688–1695, 2017.

[54] Simon Lynen, Markus W Achtelik, Steven Weiss, Maria Chli, and Roland Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, pages 3923–3929, Tokyo, Japan, Nov. 2013.

[55] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, pages 2520–2525, Shanghai, China, May 2011.

[56] Kartik Mohta, Michael Watterson, Yash Mulgaonkar, Sikang Liu, Chao Qu, Anurag Makineni, Kelsey Saulnier, Ke Sun, Alex Zhu, Jeffrey Delmerico, et al. Fast, autonomous flight in gps-denied and cluttered environments. *Journal of Field Robotics*, 35(1):101–120, 2018.

[57] M. Nathan, M. Daniel, L. Quentin, and K. Vijay. The grasp multiple micro-uav testbed. *IEEE Robot. Autom. Mag. (RAM)*, 17(3):56–65, 2010.

[58] Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran. Continuous-time trajectory optimization for online uav replanning. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, pages 5332–5339, Daejeon, Korea, Oct. 2016.

[59] Helen Oleynikova, Christian Lanegger, Zachary Taylor, Michael Pantic, Millane. Alexander, Roland Siegwart, and Juan Nieto. An open-source system for vision-based micro-aerial vehicle mapping, planning, and flight in cluttered environments. *arXiv preprint arXiv:1812.03892*, 2019.

[60] Helen Oleynikova, Zachary Taylor, Roland Siegwart, and Juan Nieto. Safe local exploration for replanning in cluttered unknown environments for microaerial vehicles. *IEEE Robotics and Automation Letters*, 3(3):1474–1481, 2018.

[61] Chris J Ostafew, Angela P Schoellig, and Timothy D Barfoot. Visual teach and repeat, repeat: Iterative learning control to improve mobile robot path tracking in challenging outdoor environments. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, pages 176–181, 2013.

[62] Michael Paton, Kirk MacTavish, Chris J Ostafew, and Timothy D Barfoot. It's not easy seeing green: Lighting-resistant stereo visual teach & repeat using color-constant images. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, pages 1519–1526, 2015.

[63] Michael Paton, Kirk MacTavish, Michael Warren, and Timothy D Barfoot. Bridging the appearance gap: Multi-experience localization for long-term visual teach and repeat. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, pages 1918–1925, 2016.

[64] Clment Petres, Yan Pailhas, Pedro Patron, Yvan Petillot, Jonathan Evans, and David Lane. Path planning for autonomous underwater vehicles. *IEEE Transactions on Robotics*, 23(2):331–341, 2007.

[65] Hung Pham and Quang-Cuong Pham. A new approach to time-optimal path parameterization based on reachability analysis. *IEEE Transactions on Robotics*, 34(3):645–659, 2018.

[66] Quang-Cuong Pham. A general, fast, and robust implementation of the time-optimal path parameterization algorithm. *IEEE Transactions on Robotics*, 30(6):1533–1540, 2014.

[67] James A Preiss, Wolfgang Hönig, Nora Ayanian, and Gaurav S Sukhatme. Downwash-aware trajectory planning for large quadcopter teams. *arXiv preprint arXiv:1704.04852*, 2017.

[68] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[69] Sean Quinlan and Oussama Khatib. Elastic bands: Connecting path planning and control. pages 802–807, 1993.

[70] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, May 2009.

[71] C. Richter, A. Bry, and N. Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Proc. of the Intl. Sym. of Robot. Research (ISRR)*, pages 649–666, Dec. 2013.

[72] Mike Roberts and Pat Hanrahan. Generating dynamically feasible trajectories for quadrotor cameras. *ACM Transactions on Graphics (TOG)*, 35(4):61, 2016.

[73] Theo Schouten and Egon L van den Broek. Incremental distance transforms (idt). In *2010 20th International Conference on Pattern Recognition*, pages 237–240. IEEE, 2010.

[74] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel. Finding locally optimal, collision-free trajectories with sequential convex optimization. In *Robotics: science and systems*, volume 9, pages 1–10, 2013.

[75] James A Sethian. Level set methods and fast marching methods. *Journal of Computing and Information Technology*, 11:1–2, 2003.

[76] Changsheng Shen, Yuanzhao Zhang, Zimo Li, Fei Gao, and Shaojie Shen. Collaborative air-ground target searching in complex environments. In *Proc. of the IEEE Intl. Sym. on Safety, Security, and Rescue Robotics (SSRR)*, page 230, Shanghai, China, 2017.

[77] Changsheng Shen, Yuanzhao Zhang, Zimo Li, Fei Gao, and Shaojie Shen. Collaborative air-ground target searching in complex environments. In *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pages 230–237. IEEE, 2017.

[78] Michael Shomin and Ralph Hollis. Fast, dynamic trajectory planning for a dynamically stable mobile robot. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, pages 3636–3641. IEEE, 2014.

[79] Danny C Sorensen. NewtonâĂŹs method with a model trust region modification. *SIAM Journal on Numerical Analysis*, 19(2):409–426, 1982.

[80] Christoph Sprunk, Gian Diego Tipaldi, Andrea Cherubini, and Wolfram Burgard. Lidar-based teach-and-repeat of mobile robot trajectories. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, pages 3144–3149, 2013.

[81] Ke Sun, Kartik Mohta, Bernd Pfrommer, Michael Watterson, Sikang Liu, Yash Mulgaonkar, Camillo J Taylor, and Vijay Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters*, 3(2):965–972, 2018.

[82] Weidong Sun, Gao Tang, and Kris Hauser. Fast uav trajectory optimization using bilevel optimization with analytical gradients. *arXiv preprint arXiv:1811.10753*, 2018.

[83] Vladyslav Usenko, Lukas von Stumberg, Andrej Pangercic, and Daniel Cremers. Real-time trajectory replanning for mavs using uniform b-splines and a 3d circular buffer. In *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.(IROS)*, pages 215–222, 2017.

[84] EI Verriest and FL Lewis. On the linear quadratic minimum-time problem. *IEEE transactions on automatic control*, 36(7):859–863, 1991.

[85] Diederik Verscheure, Bram Demeulenaere, Jan Swevers, Joris De Schutter, and Moritz Diehl. Time-optimal path tracking for robots: A convex optimization approach. *IEEE Transactions on Automatic Control*, 54(10):2318–2327, 2009.

[86] Kaixuan Wang, Fei Gao, and Shaojie Shen. Real-time scalable dense surfel mapping. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, 2019.

[87] Luqi Wang, Daqian Cheng, Fei Gao, Fengyu Cai, Jixin Guo, Mengxiang Lin, and Shaojie Shen. A collaborative aerial-ground robotic system for fast exploration. *arXiv preprint arXiv:1806.02487*, 2018.

[88] David J Webb and Jan van den Berg. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, pages 5054–5061, Germany, May 2013.

[89] Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger. Elasticfusion: Real-time dense slam and light source estimation. *The International Journal of Robotics Research*, 35(14):1697–1716, 2016.

[90] Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[91] Kai M Wurm, Armin Hornung, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, volume 2, Anchorage, AK, US, May 2010.

[92] Z. Yang, F. Gao, and S. Shen. Real-time monocular dense mapping on aerial robots using visual-inertial fusion. In *Proc. of the IEEE Intl. Conf. on Robot. and Autom. (ICRA)*, pages 4552–4559, 2017.

[93] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Proc. of Robot.: Sci. and Syst. (RSS)*, pages 109–111, UCB, USA, July 2014.

[94] Ji Zhang and Sanjiv Singh. Aerial and ground-based collaborative mapping: An experimental study. In *Field and Service Robotics*, pages 397–412. Springer, 2018.

[95] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters (RA-L)*, 2019.

[96] Zhijie Zhu, Edward Schmerling, and Marco Pavone. A convex optimization approach to smooth trajectories for motion planning with car-like robots. In *Proc. of the IEEE Control and Decision Conf. (CDC)*, pages 835–842, 2015.

[97] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.

APPENDIX A

LIST OF PUBLICATIONS

Journal Papers

1. **Fei Gao**, Luqi Wang, Boyu Zhou, Luxin Han, Jie Pan, Shaojie Shen, “Teach-Repeat-Replan: A Complete and Robust System for Aggressive Flight in Complex Environments." submitted to *IEEE Transactions on Robotics* (TRO), 2019.

2. Boyu Zhou, **Fei Gao***, Luqi Wang, Chuhao Liu, Shaojie Shen, “Robust and Efficient Trajectory Replanning for Fast Autonomous Flight."*IEEE Robotics and Automation Letters* (RA-L), IROS option, 2019. * **corresponding author**.

3. **Fei Gao**, Luqi Wang, Kaixuan Wang, William Wu, Boyu Zhou, Luxin Han, Shaojie Shen, “Optimal Trajectory Generation for Quadrotor Teach-and-Repeat." *IEEE Robotics and Automation Letters* (RA-L), ICRA option, 2019.

4. **Fei Gao**, William Wu, Wenliang Gao, Shaojie Shen, “Flying on Point Clouds: Online Trajectory Generation and Autonomous Navigation for Quadrotor in Cluttered Environments." *Journal of Field Robotics* (JFR), 2018.

5. Yi Lin*, **Fei Gao***, Tong Qin,*, Wenliang Gao*, Tianbo Liu, William Wu, Zhenfei Yang, Shaojie Shen, “Autonomous Aerial Navigation Using Monocular Visual-Inertial Fusion." *Journal of Field Robotics* (JFR), 35(1), 23-51, 2017. * **equally contributed authors**.

Conference Papers

1. **Fei Gao**, Luqi Wang, Kaixuan Wang, William Wu, Boyu Zhou, Luxin Han, Shaojie Shen, “Optimal Trajectory Generation for Quadrotor Teach-and-Repeat." *IEEE International Conference on Robotics and Automation* (ICRA), joint with RA-L, 2019.

2. **Fei Gao**, William Wu, Jie Pan, Boyu Zhou, Shaojie Shen, “Optimal Time Allocation for Quadrotor Trajectory Generation." *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS), 2018.

3. **Fei Gao**, William Wu, Yi Lin, Shaojie Shen, “Online Safe Trajectory Generation For Quadrotors Using Fast Marching Method and Bernstein Basis Polynomial." *IEEE International Conference on Robotics and Automation* (ICRA), 2018.

4. **Fei Gao**, Yi Lin, Shaojie Shen, “Gradient-Based Online Safe Trajectory Generation for Quadrotor Flight in Complex Environments." *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS), 2017.

5. **Fei Gao**, Shaojie Shen, “Quadrotor Trajectory Generation in Dynamic Environments Using Semi-definite Relaxation on Nonconvex QCQP." *IEEE International Conference on Robotics and Automation* (ICRA), 2017.

6. **Fei Gao**, Shaojie Shen, “Online Quadrotor Trajectory Generation and Autonomous Navigation on Point Clouds." *IEEE International Symposium on Safety, Security, and Rescue Robotics* (SSRR), 2016.

7. Luxin Han, **Fei Gao***, Boyu Zhou, Shaojie Shen, “FIESTA: A Fast Incremental Euclidean Distance Fields for Online Quadrotor Motion Planning." *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS), 2019,
   * **corresponding author**.

8. Jiarong Lin, Luqi Wang, **Fei Gao**, Shaojie Shen, Fu Zhang, “Flying Through A Narrow Gap Using Neural Network: An End-to-end Planning And Control Approach." *IEEE/RSJ International Conference on Intelligent Robots and Systems* (IROS), 2019.

9. Kaixuan Wang, **Fei Gao**, Shaojie Shen, “Real-Time Scalable Dense Surfel Mapping." *IEEE International Conference on Robotics and Automation* (ICRA), 2019.

10. Luqi Wang, **Fei Gao**, Fengyu Cai, Shaojie Shen, “CRASH: A Collaborative Aerial-Ground Exploration System Using Hybrid-Frontier Method." *IEEE International Conference on Robotics and Biomimetics* (ROBIO), 2018.

11. Luqi Wang*, Daqian Cheng*, **Fei Gao**, Fengyu Cai, Jixin Guo, Mengxiang Lin, Shaojie Shen, “A Collaborative Aerial-Ground Robotic System for Fast Exploration." *International Symposium on Experimental Robotics* (ISER), 2018.

12. Chong Huang, **Fei Gao**, Jie Pan, Shaojie Shen, Kwang-Ting (Tim) Cheng, et al, “ACT: An Autonomous Drone Cinematography System for Action Scenes." *IEEE International Conference on Robotics and Automation* (ICRA), 2018.

13. Changsheng Shen, Yuanzhao Zhang, Zimo Li, **Fei Gao**, Shaojie Shen, “Collaborative Air-Ground Target Searching in Complex Environments." *IEEE International Symposium on Safety, Security, and Rescue Robotics* (SSRR), 2017.

14. Zhenfei Yang, **Fei Gao**, Shaojie Shen, “Real-time Monocular Dense Mapping on Aerial Robots Using Visual-Inertial Fusion." *IEEE International Conference on Robotics and Automation* (ICRA), 2017.

APPENDIX B

LIST OF OPEN-SOURCE PACKAGES

1. GTOP: Gradient-based Trajectory Optimzation:
   https://github.com/HKUST-Aerial-Robotics/grad_traj_optimization

2. BTraj: Bézier-based Quadrotor Online Planner:
   https://github.com/HKUST-Aerial-Robotics/Btraj

3. pointcloudTraj: Trajectory Generation Directly on Pointclouds:
   https://github.com/HKUST-Aerial-Robotics/pointcloudTraj

4. TimeOptimizer: Optimal Time Allocator for Quadrotor Planning:
   https://github.com/HKUST-Aerial-Robotics/TimeOptimizer

5. Teach-Repeat-Replan: A Complete and Robust System for Aggressive Flight in Complex Environments:
   https://github.com/HKUST-Aerial-Robotics/Teach-Repeat-Replan