Received: 12 March 2018 | Revised: 8 August 2018 | Accepted: 11 October 2018

DOI: 10.1002/rob.21842

WILEY

REGULAR ARTICLE

Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments

Fei Gao* | William Wu | Wenliang Gao | Shaojie Shen

Department of Electronic and Computer Engineering, Robotics Institute, The Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong

Correspondence
Fei Gao, Department of Electronic and Computer Engineering, Robotics Institute, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong.
Email: fgaoaa@connect.ust.hk

Funding information
DJI, Grant/Award Number: Joint PG Program under HDJI Lab; Hong Kong University of Science and Technology, Grant/Award Number: project R9341

Abstract

Micro aerial vehicles (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 paper, we present a framework for online generating safe and dynamically feasible trajectories directly on the point cloud, which is the lowest-level representation of range measurements and is applicable to different sensor types. We develop a quadrotor platform equipped with a three-dimensional (3D) light detection and ranging (LiDAR) and an inertial measurement unit (IMU) for simultaneously estimating states of the vehicle and building point cloud maps of the environment. Based on the incrementally registered point clouds, we online generate and refine a flight corridor, which represents the free space that the trajectory of the quadrotor should lie in. We represent the trajectory as piecewise Bézier curves by using the Bernstein polynomial basis and formulate the trajectory generation problem as a convex program. By using Bézier curves, we can constrain the position and kinodynamics of the trajectory entirely within the flight corridor and given physical limits. The proposed approach is implemented to run onboard in real-time and is integrated into an autonomous quadrotor platform. We demonstrate fully autonomous quadrotor flights in unknown, complex environments to validate the proposed method.

KEYWORDS
aerial robotics, autonomous navigation, motion planning, trajectory generation

1 | INTRODUCTION

Micro aerial vehicles (MAVs), especially quadrotors, are increasingly used in field applications, thanks to their mobility, agility, and flexibility. Autonomous navigation enables aerial vehicles to be applied to tasks that are inaccessible or dangerous for humans or ground vehicles, such as search-and-rescue, inspection and exploration, and monitoring and surveillance. In such missions, the motion planning module plays an essential role in online generating feasible trajectories toward the target of the vehicle.

In unknown cluttered environments, it is critical to online build dense maps for obstacle avoidance. Given estimation of the vehicle state, many approaches have been proposed to convert depth measurements generated by onboard sensors into a global map.

*This paper is based on our previous work originally presented at the 14th International Symposium on Safety, Security, and Rescue Robotics 2016 (SSRR 2016). This extended study includes significant technical improvements in the methodology. Extensive field experiments, as well as a large set of benchmark tests, are also presented.

Representative methods include the voxel grid (Elfes, 1989), Octomap (Wurm, Hornung, Bennewitz, Stachniss, & Burgard, 2010), elevation map (Choi, Park, Lim, & Yu, 2012), and so forth. 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 paper, instead of using the above-mentioned postprocessed maps, we opt to plan trajectories directly on point clouds' raw data. A point cloud, which is an underlying representation of the environment, does not need a discretization of the configuration space or complicated maintenance. It is essentially an unordered set of three-dimensional (3D) raw data, which can be obtained by many sensors, such as by camera in Yang, Gao, and Shen (2017) and by light detection and ranging (LiDAR) in Zhang and Singh (2018). 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 introduced in our previous research published in F. Gao and Shen (2016) and is significantly improved and extended in this paper. We summarize the contributions of this study as follows:

1. A rapidly exploring random tree (RRT)*-based, anytime path-finding method, named safe-region RRT*, for incrementally finding and refining a collision-free flight corridor directly on point clouds. The proposed approach 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.

We list our improvements and extensions compared with our previous conference paper (F. Gao and Shen, 2016):

1. We advance the path-finding module to an incremental version. In this paper, the flight corridor is continuously refined and modified to adapt to newly discovered obstacles during flights, whereas in our previous work, when the vehicle replanned all previous samples were discarded and the entire roadmap was regenerated from scratch.
2. We improve the performance of the trajectory optimization module. Using the Bernstein polynomial basis, we now generate a safe and kinodynamically feasible trajectory by solving a single convex program. In our previous work, in contrast, the safety and the dynamical feasibility were enforced by iteratively solving the optimization and adding extra constraints. Also, by introducing a conic reformulation, the stability and the efficiency of the trajectory optimizer are further improved.
3. We do a whole new set of field experiments in diverse cluttered environments, including more complex and larger scenes than in our previous work, to validate the robustness of our proposed method. We also present extensive benchmarked comparisons, which are not included in our previous paper.

In what follows, we discuss the relevant literature in Section 2. The autonomous quadrotor system is briefly introduced in Section 3. In Sections 4 and 5, we thoroughly present our flight corridor generation and trajectory optimization methods. The results are presented in Section 6. Finally, the paper is concluded in Section 7.

2 | RELATED WORKS

There has been much research on developing autonomous quadrotor systems. Among these works, some focus on the perception functionality (Bry, Richter, Bachrach, & Roy, 2015; Faessler et al., 2016; S. Shen, Michael, & Kumar, 2015) and some present complete system results (Fang et al., 2017; Lin et al., 2018; Mohta et al., 2018). In this study, we propose a motion planning framework for the safe navigation of an autonomous quadrotor. Here, we briefly go through recent representative motion planning algorithms.

2.1 | Path finding

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 (LaValle, 2001) to searching based (Likhachev, Gordon, & Thrun, 2004) have been proposed and applied. In the representative sampling-based method (RRT; LaValle, 2001), samples are drawn randomly from the configuration space to guide a tree to grow toward 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 (Karaman & Frazzoli, 2011). Asymptotically optimal sampling-based methods, including probabilistic roadmap* (PRM*), rapidly exploring random graph (RRG), and RRT*, in which the solution converges to the global optimality as the samples increase, have also been proposed in Karaman and Frazzoli (2011). 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 Webb and van den Berg (2013) combines a fixed final state and frees the 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, which combines RRG and the belief roadmap, was proposed by Bry and Roy (2011). 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 2D, 3D, or higher-order state space. Typical methods include Anytime Repairing A* (ARA*; Likhachev et al., 2004), Jump Point Search (JPS; Harabor & Grastien, 2011), and hybrid A* (Dolgov, Thrun, Montemerlo, & Diebel, 2010). 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 an order of magnitude compared with 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 (Likhachev & Ferguson, 2009) 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 anytime dynamic A* (AD*) (Likhachev, Ferguson, Gordon, Stentz, & Thrun 2005). Similarly, in Liu, Atanasov, Mohta, and Kumar (2017), 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.

2.2 | Trajectory optimization

Many methods have been proposed to optimize the discrete path to generate a continuous trajectory for smooth movements of vehicles. Gradient-based methods, like covariant hamiltonian optimization for motion planning (CHOMP) (Ratliff, Zucker, Bagnell, & Srinivasa, 2009), formulate the trajectory optimization problem as a nonlinear optimization over the penalty of the safety and the smoothness. In Lin et al. (2018) and Oleynikova et al. (2016), gradient-based methods are applied to piecewise polynomial trajectories and local planning for quadrotor navigation. Control-based methods consider the dynamic model of the quadrotor and directly solves the control input, which drives the quadrotor safely to targets. In van den Berg, Wilkie, Guy, Niethammer, and Manocha (2012), the authors consider a linear quadrotor model and a linear quadratic Gaussian (LQG) controller. They propose to use the LQG obstacle set to represent all target states that lead to collisions. Safe control commands are generated as long as the farthest target point is selected out of the obstacle set. Sequential convex programming (SCP), or a so-called concave–convex procedure (CCP), is utilized in Schulman et al. (2013) to convexify and solve the nonconvex trajectory generation problem iteratively. In iterations, the relaxed convex problem approaches the objective and constraints of the original nonconvex problem. Similarly, the incremental SCP described in Y. Chen, Cutler, and How (2015) is applied on quadrotor flights and shows improvements in time efficiency and success rate.

Mellinger and Kumar (2011) pioneered a minimum-snap trajectory generation algorithm, where the trajectory is represented by piecewise polynomial functions, and the trajectory generation problem is formulated as a quadratic programming (QP) problem. A closed-form solution for minimum-snap trajectory generation is obtained in Richter, Bry, and Roy (2013), where the safety of the trajectory is achieved by iteratively adding intermediate waypoints in a discrete path found by RRT*. Mixed integer quadratic programming (MIQP), which uses combinational optimization with QP, is introduced in Mellinger, Kushleyev, and Kumar (2012) to enforce the avoidance of collisions for quadrotors. In Deits and Tedrake (2015), iterative regional inflation by semidefinite (IRIS) programming, a recently developed convex segmentation method, is used to approximate the free space with convex regions. Then, a sum-of-squares (SOS) program is used to assign the smooth trajectories to these convex regions. J. Chen, Liu, and Shen (2016), Liu, Watterson, et al. (2017), and our previous work (F. Gao & Shen, 2016) all online carve a flight corridor consisting of simple convex regions against obstacles and confine the trajectory within the corridor. The safety and the dynamical feasibility are achieved by iteratively adding constraints on violated extrema and re-solving the convex program in J. Chen et al. (2016) and F. Gao and Shen, 2016). However, in this way, a feasible solution can only be obtained after many iterations of the original program. Another way would be to sample the time instance along the trajectory and add extra constraints (Liu, Watterson, et al., 2017) on it. However, this would make the complexity of the problem scale poorly, and there are no criteria for choosing the resolution of discretization. In Campos-Macías, Gómez-Gutiérrez, Aldana-López, de la Guardia, and Parra-Vilchis (2017), trajectories with piecewise constant acceleration are generated. The authors derived the velocity bound as well as the maximum deviation between the trajectory and the path concerning the limit of acceleration and minimum path clearance. The generated trajectory is safely bounded regarding acceleration, velocity, and trajectory deviation. However, the bound is very conservative, making the generated trajectory's speed always deficient.

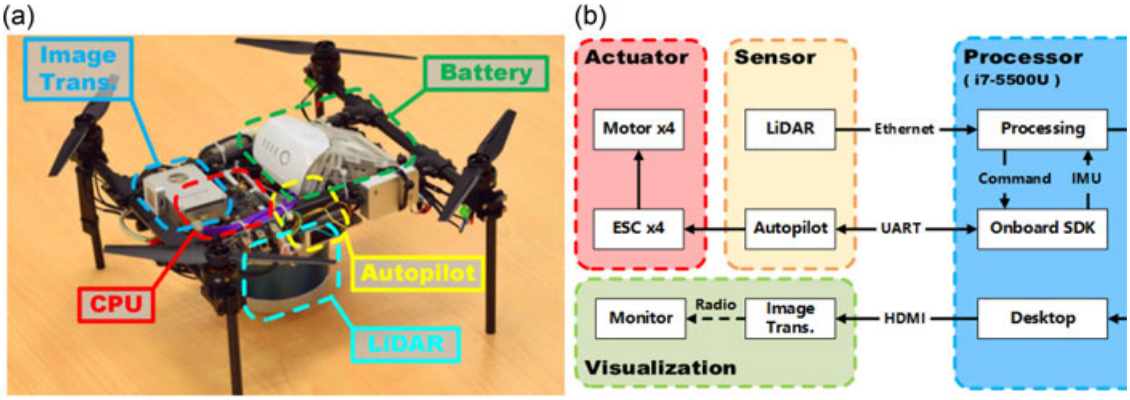

FIGURE 1 Our developed quadrotor platform. Some key devices, including the onboard CPU, Velodyne LiDAR, DJI A3 autopilot, image transmission, and intelligent battery, are labeled in (a). The overall hardware and communication architecture is shown in (b). (a) Self-developed quadrotor platform and (b) hardware architecture. CPU: central processing unit; ESC: electronic speed controller; HDMI: high-definition multimedia interface; IMU: inertial measurement unit; LiDAR: light detection and ranging; UART: universal asynchronous receiver-transmitter [Color figure can be viewed at wileyonlinelibrary.com]

3 | SYSTEM OVERVIEW

3.1 | Hardware architecture

Compared with our previous platform in F. Gao and Shen (2016), in this paper, we develop a more compact and modular aerial robot (Figure 1). We use a minicomputer, which 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 solid-state drive (SSD). All processing is carried out onboard on this minicomputer. Our quadrotor is equipped with a Velodyne VLP-16 Puck LITE[2] (Velodyne LiDAR, San Jose, CA) LiDAR for perception and a DJI A3 autopilot[3] (DJI, Shenzhen, China) for feeding inertial measurement unit (IMU) data and stabilizing the attitude. The autopilot communicates with the onboard computer by the intermediate layer, DJI onboard software development kit (SDK).[4] We disable the Global Positioning System (GPS) and magnetometer functionalities in 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 (DJI, Shenzhen, China) image transmission system to connect with the onboard computer. The quadrotor dynamic system consists of a DJI Snail motor (DJI, Shenzhen, China) with a 7-in. 7027s propeller[5] and DJI Takyon Z425-M (DJI, Shenzhen, China) electronic speed controller (ESC).[6] The quadrotor is powered by the intelligent battery used in the DJI Phantom 4 Pro.[7] (DJI, Shenzhen, China).

[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

[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

3.2 | Software architecture

The complete software pipeline of our quadrotor system, from perception to planning and control, is shown in Figure 2. Scan measurements from the 3D 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 (Zhang & Singh, 2014) is first 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 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 k dimensional-tree (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 special euclidean group (3) [SE(3)] (Lee, Leoky, & McClamroch, 2010) 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 minicomputer.

3.3 | Localization and mapping

We now present our implementation of the laser-based state estimation and simultaneous mapping method. Our approach is adopted from Zhang and Singh (2014), and it constitutes the

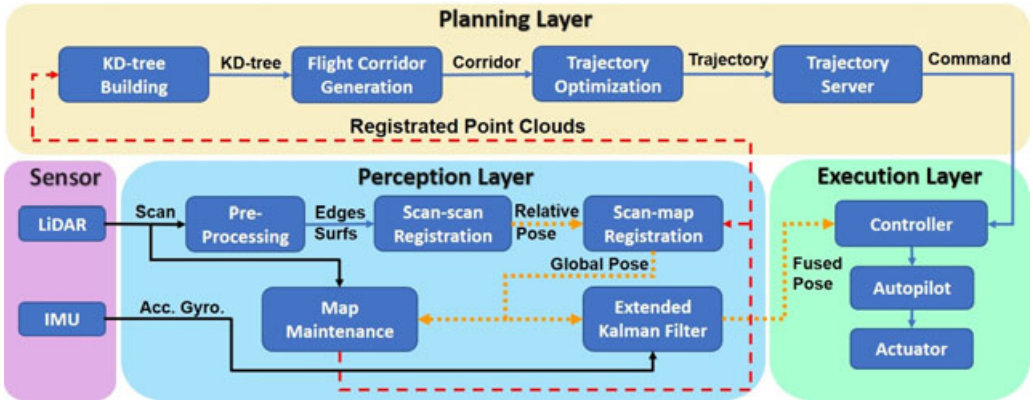

FIGURE 2 The software architecture (Section 3.2) of the quadrotor platform. In the perception layer, scan-to-scan registration and scan-to-map registration are performed to get the 6-DOF pose estimation and simultaneously generate registered point clouds. High-rate odometry is obtained by fusing the 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 carried out onboard. DOF: degrees of freedom; IMU: inertial measurement unit; LiDAR: light detection and ranging [Color figure can be viewed at wileyonlinelibrary.com]

perceptual foundation for a complete autonomous navigation system. The state estimation is initialized, and the origin of world frame is fixed at the quadrotor take-off point. The 6-DOF relative motions between the LiDAR frame and the world frame are 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 (Figure 2). In the preprocessing submodule, points are classified as edges and surfaces based on curvature and are extracted from each 360° 3D 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 alignments 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 F. Gao and Shen (2016) and Zhang and Singh (2014). Additionally, we mention two features in our maintenance of the globally registered point clouds:

1. To bound the computational complexity for our path-finding method (presented in Section 4), 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, which have very near coordinates.
2. For each point first 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 ith scan as follows:

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

where α and β are the parameters that control the rate of decrease and increase of the weight, respectively, 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, respectively, at which the point is considered as permanently added or removed from the map. Points that are repeatedly observed will eventually be fixed, whereas points that have received few observations, such as points on moving objects or outliers, will be removed after some updates. The estimation and mapping module in our system is able to onboard run at more than 10 Hz with a satisfactory density and accuracy.

To achieve the high-rate state estimation required for feedback control, we use a loosely coupled (Lynen, Achtelik, Weiss, Chli, & Siegwart, 2013) 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, whereas 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.

4 | ANYTIME FLIGHT CORRIDOR GENERATION ON POINT CLOUDS

4.1 | KD-tree-based point cloud representation

Our laser-based state estimation algorithm is performed utilizing the KD-tree (Bentley, 1975) data structure for fast nearest neighboring points search. A balanced KD-tree with N points can be constructed in $O(N \log N)$ time with a space complexity of $O(kN)$, and the M

nearest neighbor search has a time complexity of $O(M \log N)$. 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 3D space to its nearest neighboring point (obstacle) in 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 100 m × 100 m × 5 m map is on average 8–10 ms for our onboard computer, which is far less than the mapping and replanning frequency (10 Hz). 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 Section 3.3, the number of points in the point cloud is therefore upper bounded.

4.2 | Sampling-based random safe-region tree generation

As presented in Section 4.1, the safe radius of an arbitrary point in the map can be found by 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 rewired. 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 (Gammell, Srinivasa, & Barfoot, 2014) 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, which 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 reinitialize it as in our previous work (F. Gao & Shen, 2016). 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.

4.2.1 | Random tree expansion

The expansion of the random tree is the basis of our proposed algorithm and is shown in Algorithm 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 $n{\cdot}\mathbf{c}$ of its center, the length $n \cdot r$ of its radius, and the total path cost $n{\cdot}f$ 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 Algorithm 2, the **Intersect**(c_r, n_n) function generates a ray from $\mathbf{c}_r$ to $n_n{\cdot}\mathbf{c}$ and returns the intersection point of the ray with n_n's sphere. We set this coordinate as the center of the new node $n_r \cdot \mathbf{c}$. We then perform a radius search **RadiusSearch**$(n_r{\cdot}\mathbf{c}, \mathcal{P})$ to find the radius of the largest sphere centered at $n_r{\cdot}\mathbf{c}$ against the point cloud raw data $\mathcal{P}$ and set it as $n_r{\cdot}r$. 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 (Karaman & Frazzoli, 2011). 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 Gammell et al. (2014) to improve the convergence rate toward 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 Figure 3. 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
$C \leftarrow$ min(**PathExtract**$(\mathcal{T})$)
return 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**$(n_r.\mathbf{c}, \mathcal{P})$
$n_r \cdot \mathbf{c} \leftarrow \mathbf{c}$
$n_r \cdot r \leftarrow r$
return n_r

4.2.2 | Trajectory commitment and branch pruning

After the initial flight corridor is obtained and the trajectory within it is generated (to be discussed in Section 5), the quadrotor starts to track the trajectory toward 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, the 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* (Karaman, Walter, Perez, Frazzoli, & Teller, 2011) 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 Figure 4. 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. Because 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

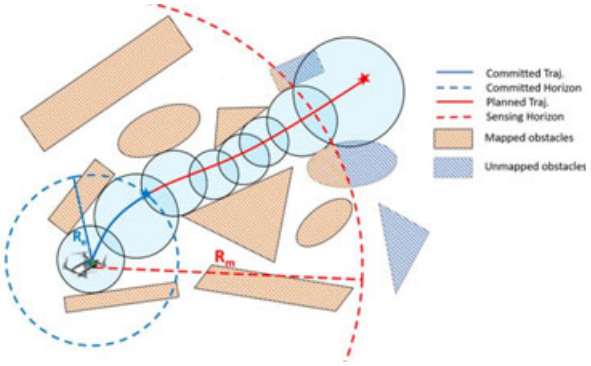

FIGURE 4 Trajectory commitment mechanism in a receding horizon. In this figure, the flight corridor is visualized as blue circles. The red curve is the generated trajectory to the global planning target, whereas the blue curve is the committed trajectory, which is reserved in a horizon for execution. The red-dashed circle and the blue-dashed arc indicate the sensing range and execution horizon, respectively. Mapped and unknown obstacles are in orange and blue hatching, respectively [Color figure can be viewed at wileyonlinelibrary.com]

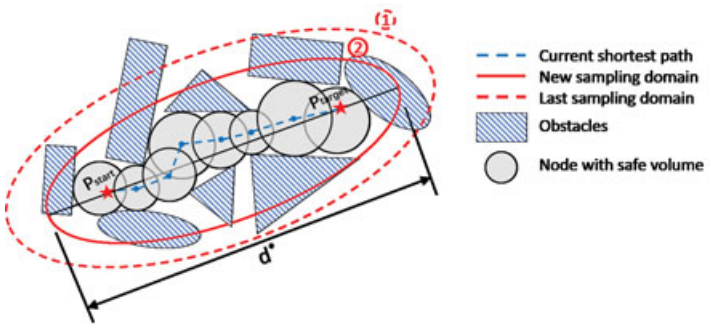

FIGURE 3 Heuristic-sampling domain update. Red-dashed ellipsoid with marker 1 is the last hyper-ellipsoid domain for generating samples. After a new better solution d^* has been found, the sampling domain shrinks to the red solid ellipsoid with marker 2. Red stars indicate the start point and the target point for the path-finding mission. Gray spheres are safe regions found by sampling and nearest neighbor query, and the blue-dashed poly-line is the current best path discovered by a new sample [Color figure can be viewed at wileyonlinelibrary.com]

accordingly. During the execution of the committed trajectory, the random tree is continuously expanded and rewired 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 carried out 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 in A^* graph search, an underestimate of the optimal cost for a node n to reach the target can be used as an admissible heuristic $n \cdot h$. 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 $n \cdot f + n \cdot h$ larger than the current best solution f_{best} is not considered promising and is deleted from the tree permanently.

4.2.3 | 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 the 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 Section 3.3, 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 because most of the nodes will not contribute to the optimal path. In this paper, we use the lazy-validity-checking strategy, similar to Hauser (2015). 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 Algorithm 3 can be divided into two stages. First is the exploitation stages. Under a time limit $t^{\text{r}}_{\text{max}}$, new samples are continuously drawn, and the random tree is expanded, the same as in Algorithm 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 Algorithm 4. For each node n_{c} in $\mathcal{C}$, 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 predefined time $t^{\text{e}}_{\text{max}}$ 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 reinitialize the safe-region RRT* only if the quadrotor has finished the committed trajectory. Otherwise, Algorithm 3 proceeds continuously, trying to find a feasible path by adding more and more samples.

Algorithm 3 Safe-region RRT* refinement

```
t ← 0
while t ≤ t^r_max do
    T ← TreeExpand(T)
    t ← Time()
end while
t ← 0
{P} ← PathExtract(T)
while t ≤ t^e_max AND {P} ≠ ∅ do
    C ← min({P})
    if PathCheck(C) then
      return C
    end if
    {P} ← {P}/C
    t ← Time()
end while
return ∅
```

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

```
Require P, T
for each n_c ∈ C do
    n_c · r ← RadiusSearch(n_c·c, P)
    if VolumeCheck(n_c) then
      {n_child} ← Successor(n_c)
      for each n_z ∈ {n_child} do
        if EdgeBreak(n_c, n_z) then
          T ← BranchCut(T, n_z)
          return False
        end if
      end for
    else
      T ← BranchCut(T, n_c)
      return False
    end if
end for
return True
```

5 | SAFE AND DYNAMICALLY FEASIBLE TRAJECTORY GENERATION

As presented by Mellinger and Kumar (2011), a quadrotor system enjoys the differential flatness property. The full state space of a quadrotor system can be reduced to independent 3D positions, the yaw angle, and

their derivatives. Trajectories of 3D positions with derivatives bounded within the kinodynamic limits can, therefore, be followed by a properly designed geometric controller (Lee et al., 2010). In this paper, because 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.

5.1 | Piecewise Bernstein basis trajectory formulation

In this paper, instead of using a monomial basis polynomial, we use the Bernstein polynomial basis and represent the trajectory as a piecewise Bézier curve, which is a special case of a B-spline curve. We start by comparing the formulations of these two bases. The ith-order monomial polynomial basis of variable t is t^i. The monomial polynomial function for each segment of the piecewise trajectory is written as

$$P_j(t) = p_j^0 + p_j^1 t + p_j^2 t^2 + \cdots + p_j^n t^n, \tag{2}$$

where n is the degree of the monomial polynomial and $[p_j^0, p_j^1, ..., p_j^n]$ denoted as $\mathbf{p_j}$ is the set of coefficients of the jth piece of the trajectory.

The ith-order Bernstein polynomial basis $b_n^i(t)$ is defined as

$$b_n^i(t) = \binom{n}{i} \cdot t^i \cdot (1-t)^{n-i}, \tag{3}$$

where n is the degree of the Bernstein polynomial and $\binom{n}{i}$ is the binomial coefficient. A polynomial function constituted by the Bernstein basis is called a Bézier curve and is written as

$$B_j(t) = c_j^0 b_n^0(t) + c_j^1 b_n^1(t) + \cdots + c_j^n b_n^n(t) = \sum_{i=0}^{n} c_j^i b_n^i(t), \tag{4}$$

where $[c_j^0, c_j^1, ..., c_j^n]$ denoted as $\mathbf{c_j}$ is the set of control points of the jth piece of the Bézier curve. Actually, for a Bézier curve, the control points can be viewed as weights of the Bernstein basis and the Bernstein basis can also be viewed as weights of the control points. Compared with a monomial basis polynomial, a Bézier curve has several special properties, which are useful in our algorithm:

1. *Endpoint interpolation property*: A Bézier curve always starts at its first control point and ends at its last control point, and never passes any other control points.
2. *Fixed interval property*: A Bézier curve parameterized to variable t is defined on $t \in [0, 1]$.
3. *Convex hull property*: A Bézier curve $B(t)$ consists of a set of control points $\mathbf{c}$ that 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 a hodograph and is also a Bézier curve with control points defined by $c_i^{(1)} = n \cdot (c_{i+1} - c_i)$, where n is the degree.

According to the above-mentioned Property 2 of the Bézier curve, we need to scale the parameter t to an arbitrary allocated time for each piece of the curve. Consequently, an m-segment piecewise Bézier curve in one dimension μ out of x, y, z can be written as follows:

$$f_\mu(t) = \begin{cases} s_1 \cdot \sum_{i=0}^{n} c_{\mu 1}^i b_n^i\left(\frac{t-T_0}{s_1}\right), & t \in [T_0, T_1], \\ s_2 \cdot \sum_{i=0}^{n} c_{\mu 2}^i b_n^i\left(\frac{t-T_1}{s_2}\right), & t \in [T_1, T_2], \\ \vdots & \vdots \\ s_m \cdot \sum_{i=0}^{n} c_{\mu m}^i b_n^i\left(\frac{t-T_{m-1}}{s_m}\right), & t \in [T_{m-1}, T_m], \end{cases} \tag{5}$$

where c_{ji} is the ith control point of the jth 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 because, 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 Section 4. We assign each piece of the Bézier curve to one sphere of the flight corridor.

Following the minimum-snap formulation (Mellinger & Kumar, 2011), the objective function we use for trajectory generation is the integral of the squared kth derivative of the trajectory, which is written as

$$J = \sum_{\mu \in \{x,y,z\}} \int_0^T \left(\frac{d^k f_\mu(t)}{dt^k}\right)^2 dt. \tag{6}$$

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 $\mathbf{Q}_o$ 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 jth segment of the trajectory is denoted as $f_{\mu j}(t)$. Then, the pure Bézier curve (without scaling, as in Equation (4)) 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 jth segment is derived as

$$\begin{aligned} J_{\mu j} &= \int_0^{s_j} \left(\frac{d^k f_{\mu j}(t)}{dt^k}\right)^2 dt = \int_0^1 s_j \left(\frac{s_j \cdot d^k (g_{\mu j}(\tau)}{d(s_j \cdot \tau)^k}\right)^2 d\tau \\ &= \frac{1}{s_j^{2k-3}} \cdot \int_0^1 \left(\frac{d^k g_{\mu j}(t)}{d\tau^k}\right)^2 d\tau. \end{aligned} \tag{7}$$

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 $\mathbf{M}$, that is, $\mathbf{p} = \mathbf{M} \cdot \mathbf{c}$. For instance, $n = 6$, we can derive

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -6 & 6 & 0 & 0 & 0 & 0 & 0 \\ 15 & -30 & 15 & 0 & 0 & 0 & 0 \\ -20 & 60 & -60 & 20 & 0 & 0 & 0 \\ 15 & -60 & 90 & -60 & 15 & 0 & 0 \\ -6 & 30 & -60 & 60 & -30 & 6 & 0 \\ 1 & -6 & 15 & -20 & 15 & -6 & 1 \end{bmatrix}. \tag{8}$$

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 (Mellinger & Kumar, 2011). And the overall objective $\mathbf{Q}_o$ is the sum of $\mathbf{Q}_\mu$ in each axis. We leave out the details of $\mathbf{Q}_o$ for brevity. Note that at a given order n, the transformations 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.

5.2 | 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

$$a_{\mu j}^{0,i} = c_{\mu j}^{i},\ a_{\mu j}^{l,i} = \frac{n!}{(n-l)!} \cdot \left(a_{\mu j}^{l-1,i+1} - a_{\mu j}^{l-1,i}\right),\ l \geq 1, \tag{9}$$

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

5.2.1 | 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 nth 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 lth ($l \leq n$)-order derivative (waypoints) $d_{\mu j}^{(l)}$ that exists at the beginning[8] of the μ dimension, the jth piece of the trajectory, we have

$$a_{\mu j}^{l,0} \cdot s_j^{(1-l)} = d_{\mu j}^{(l)}. \tag{10}$$

5.2.2 | Continuity constraints

The trajectory must be continuous at all the ϕth ($0 \leq \phi \leq k-1$) 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 jth and (j+1)th pieces of the curve, to set a continuity constraint at the ϕth order, we can write the equation as

$$a_{\mu j}^{\phi,n} \cdot s_j^{(1-\phi)} = a_{\mu,j+1}^{\phi,0} \cdot s_{j+1}^{(1-\phi)}. \tag{11}$$

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

5.2.3 | 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 Section 4. For one segment of the trajectory, because 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 jth segment, constraints are

$$\sum_{\mu \in \{x,y,z\}} (c_{ji}^{\mu} - p_j^{\mu})^2 \leq r_j^2, \tag{12}$$

where p_j and r_j are the center and radius of the corresponding sphere in the jth piece of the trajectory.

5.2.4 | Dynamical feasibility constraints

Similar to the safety constraints, utilizing Properties 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 the 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 jth segment, we have

$$\begin{aligned} &v_m^- \leq n \cdot \left(c_{\mu j}^{i} - c_{\mu j}^{i-1}\right) \leq v_m^+, \\ &a_m^- \leq n \cdot (n-1) \cdot \left(c_{\mu j}^{i} - 2c_{\mu j}^{i-1} + c_{\mu j}^{i-2}\right) \Big/ s_j \leq a_m^+. \end{aligned} \tag{13}$$

5.3 | Trajectory optimization formulation

The waypoint constraints and the 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, $\mathbf{c} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_m]$. To sum up, the trajectory generation problem is written as

$$\begin{aligned} \min\ & \tfrac{1}{2}\mathbf{c}^T\mathbf{Q}_0\mathbf{c} \\ \text{s.t.}\ & \tfrac{1}{2}\mathbf{c}^T\mathbf{Q}_i\mathbf{c} + \mathbf{a}_i\mathbf{c} \leq r_i, \quad i = 1, \ldots, n, \\ & \mathbf{A}_{eq}\mathbf{c} = \mathbf{b}_{eq}, \\ & \mathbf{A}_{ie}\mathbf{c} \leq \mathbf{b}_{ie}, \end{aligned} \tag{14}$$

where all quadratic matrices $\mathbf{Q}_i$ are positive semidefinite. 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 Figure 5 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, because 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 because 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 | Conic optimization reformulation

To address the numerical issue of the program and improve efficiency, we convert the above QCQP to an SOCP. Compared with the QCQP, modeling the optimization as an 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 an 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 Andersen (2013).

For our QCQP formulation in Equation (14), given a quadratical constraint in the canonical form

$$\frac{1}{2}\mathbf{c}^T\mathbf{Q}_i\mathbf{c} + \mathbf{a}_i\mathbf{c} \le r_j, \tag{15}$$

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

$$w_i = r_i - \mathbf{a}_i\mathbf{c}. \tag{16}$$

Then, an equivalent form of the quadratical constraint is derived as

$$\begin{aligned} \tfrac{1}{2}\mathbf{c}^T\mathbf{Q}_i\mathbf{c} &\le w_i, \\ r_i - \mathbf{a}_i\mathbf{c} &= w_i. \end{aligned} \tag{17}$$

Because all $\mathbf{Q}_i$ in Equation (14) are positive semidefinite, they can always be factorized as

$$\mathbf{Q}_i = \mathbf{F}_i^T \cdot \mathbf{F}_i \tag{18}$$

by the Cholesky decomposition or other methods. Then, by combining Equations (17) and (18), we can derive

$$\begin{aligned} (\mathbf{F}_i \cdot \mathbf{c})^2 &\le 2w_i, \\ r_i - \mathbf{a}_i\mathbf{c} &= w_i, \end{aligned} \tag{19}$$

which can be further rewritten in an equivalent conic formulation:

$$(w_i, t_i, \mathbf{y}_i) \in \mathbf{K}_r^{k_y+2}, \tag{20}$$

$$\begin{cases} w_i = r_i - \mathbf{a}_i\mathbf{c}, \\ t_i = 1, \\ \mathbf{y}_i = \mathbf{F}_i \cdot \mathbf{c}, \end{cases} \tag{21}$$

where k_y is the dimensionality of $\mathbf{y}_i$ and $\mathbf{K}_r^{k_y+2}$ a rotated second-order cone with the dimensionality of $k_y + 2$. An n-dimensional rotated

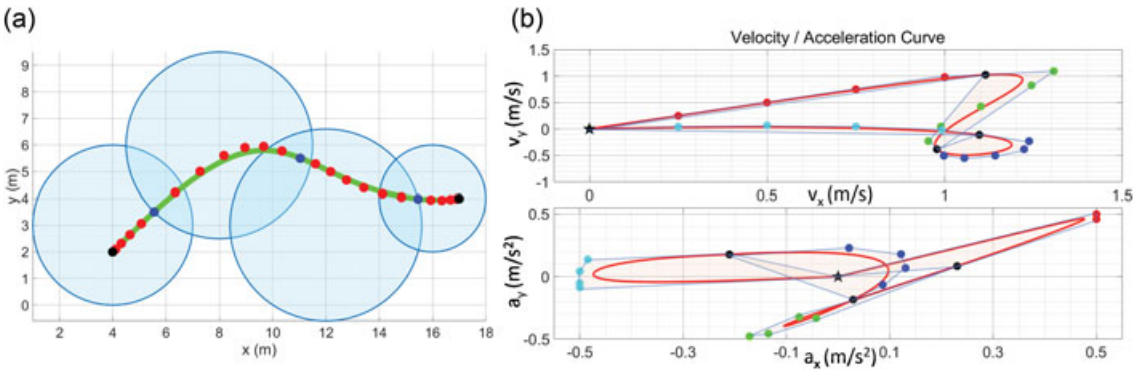

FIGURE 5 An illustration of a piecewise Bézier curve generated in the flight corridor. In (a), the curve is in green. The control points, which are enforced by waypoint constraints and continuity constraints, are shown in black and blue, and other intermediate control points are red. The corridor is 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. The convex hulls formed by control points of each piece of the curve are plotted in white transparent polygons [Color figure can be viewed at wileyonlinelibrary.com]

quadratic cone $\mathbf{K}_r^n$ is mathematically defined as $\{\mathbf{x} \in \mathcal{R}^n | 2x_1x_2 \geq x_3^2 + \cdots + x_n^2,\ x_1, x_2 \geq 0\}$. 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:

$$\begin{aligned} \min\ & w_0 \\ \text{s.t.}\ & (w_i, t_i, \mathbf{y}_i) \in \mathbf{K}_i^{r,k_y+2}, \quad i = 0, \ldots, n, \\ & w_i = r_i - \mathbf{a}_i\mathbf{c}, \qquad\quad i = 1, \ldots, n, \\ & \mathbf{A}_{\text{eq}}\mathbf{c} = \mathbf{b}_{\text{eq}}, \\ & \mathbf{A}_{\text{ie}}\mathbf{c} \leq \mathbf{b}_{\text{ie}}, \end{aligned} \tag{22}$$

where $t_i = 1$ and $\mathbf{y}_i = \mathbf{F}_i \cdot \mathbf{c}$, $i = 0, \ldots, n$. 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 with the original QCQP formulation, although more variables are introduced as slackness variables, the better numerical stability and the 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 Section 6.2.

6 | RESULTS

6.1 | 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 (Campos-Macías et al., 2017; Richter et al., 2013). 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 carried out onboard. The flights of the quadrotor in all trials are fully autonomous, without any human operations or interventions. In our onboard tests, the Velodyne 3D LiDAR runs at 20 Hz and outputs 15,000 points in each scan. The state estimation module runs at the same rate as the sensor, whereas the registered point clouds for motion planning are updated at 10 Hz. The replanning frequency is also 10 Hz. Although the Velodyne LiDAR can detect obstacles as far as 100 m, the sensing range in our system is set as 30 m 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 6 s, considering the sensing range and the maximum allowed speed of the quadrotor. The time limit for initializing our path-finding module is 80 ms. In each replanning loop, the time budgets for refining and re-evaluating the path are 60 and 40 ms, respectively. The maximum radius of the safe region in the flight corridor is 5 m.

6.2 | 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 (Section 5) 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 trajectory generated in

1. the monomial polynomial basis with the iterative constraining strategy (F. Gao & Shen, 2016);
2. the Bernstein polynomial basis with the problem in the QCQP formulation (Section 5.3);
3. the Bernstein polynomial basis with the problem in the equivalent SOCP formulation (Section 5.4).

We construct numerous instances to apply the above three solvers and compare the solutions. We randomly generate 10 maps, with the fixed obstacle number as 400. The limit of iterations in the monomial polynomial QCQP solver is set as 5. The maximum allowed velocity and acceleration are fixed as 2 m/s and 2 m/s^2, respectively. 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 Figure 6 and demonstrate two important conclusions. First, formulating the trajectory generation problem in a noniterative 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, an SOCP is much more numerically stable than a QCQP, and more easily converges to the global optimal solution. Second, 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.

6.3 | Benchmark results

We present benchmarked comparisons against several other works. First, we compare the proposed trajectory optimization method against another optimization-based method (Campos-Macías et al., 2017), which

[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

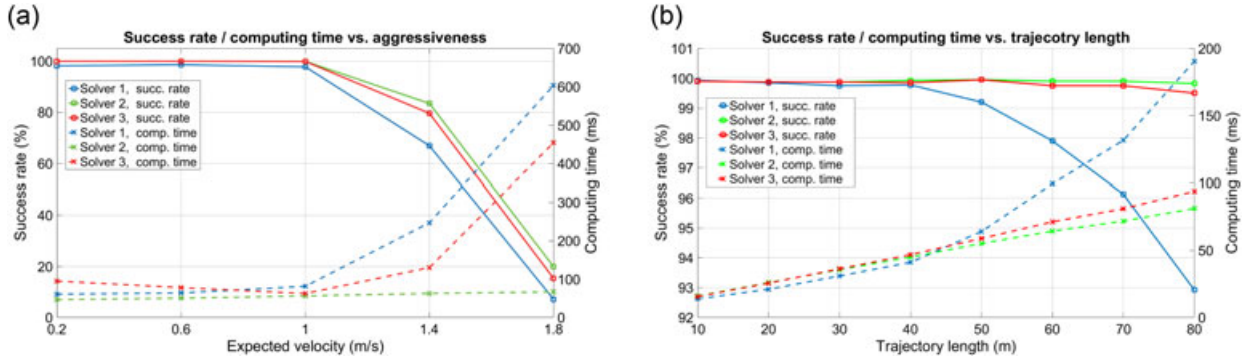

FIGURE 6 The results of generating trajectories against different expected average velocities in (a) and against different expected trajectory lengths in (b). Success rate and computing time are compared. Solver 1 refers to the monomial basis iterative formulation used in our previous work (F. Gao & Shen, 2016), solver 2 is the Bernstein basis QCQP formulation given in Section 5.3, and solver 3 refers to the conic solver we finally derived in Section 5.4. The success rate for each method is shown by cube markers, and the computing time is represented by cross markers. (a) Trajectories generated against varying aggressiveness and (b) trajectories generated against varying length [Color figure can be viewed at wileyonlinelibrary.com]

also guarantees the safety and kinodynamical feasibility by solving a single convex optimization program. Second, 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 (Richter et al., 2013) and our previous work (F. Gao & Shen, 2016). Trajectories generated in comparisons are shown in Figure 7.

6.3.1 | Comparison of the trajectory optimization

We compare the back-end trajectory optimization in our proposed motion planning framework against the benchmark method from Campos-Macías et al. (2017), which is denoted as **Benchmark1**. In the comparison, we set the velocity limit of the quadrotor as 2 m/s and the acceleration limit as 2 m/s^2. Unlike our method, in the benchmark method, the acceleration and the 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 as 0.5 m.

We first fix the obstacle density as 500 trees/map to conduct an overall comparison. We randomly generate 10 maps and do 400 trials on each one. As shown in Table 1, **Benchmark1** is overconservative compared with our method. Because 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 (0.5 m) compared with the value (0.035 m) used in the original paper (Campos-Macías et al., 2017), 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 overconservative formulation used to derive the safety guarantee. For the same reason, the lower resulting objective in **Benchmark1** is obtained by taking a much longer time to finish the trajectory.

We also present the comparisons of the success rate versus obstacle density and the computational time versus trajectory length. We generate trajectories in maps with obstacle densities ranging from 100 trees to 1,100 trees/map. Then, we fix the obstacle density

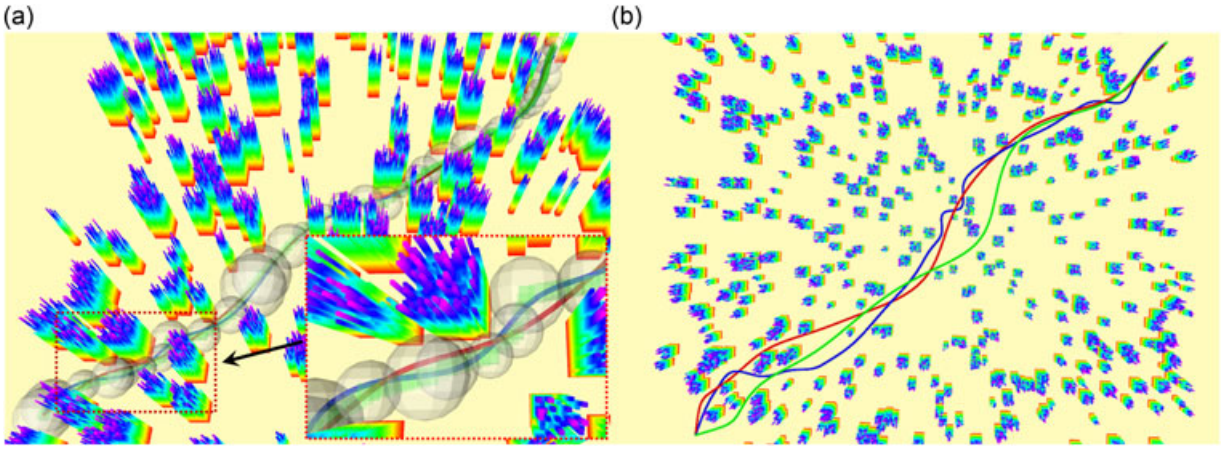

FIGURE 7 The comparison of the trajectory optimization in (a) and of the complete planning framework in (b). In (a), red and blue curves are generated by our proposed method and **Benchmark1** (Campos-Macías et al., 2017), respectively. White transparent spheres indicate our flight corridor and green transparent cubes are the path constraints in **Benchmark1**. In (b), red, green, and blue curves are, respectively, generated by our previous method (F. Gao & Shen, 2016), our proposed method, and **Benchmark2** (Richter et al., 2013). Comparison of (a) trajectory optimization and (b) planning framework [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 1 Comparison of trajectory optimization

Method	Length (m)	Objective	Time (ms)	Ave. Vel.	Ave. Acc.	Max. Vel.	Max. Acc.	Succ. Rate (%)
Proposed	80.0225	3.3064	88.60	0.8874	0.0995	1.7568	0.9355	99.88
Benchmark1	82.0235	0.9206	333.21	0.3815	0.0282	0.9804	0.7452	99.65

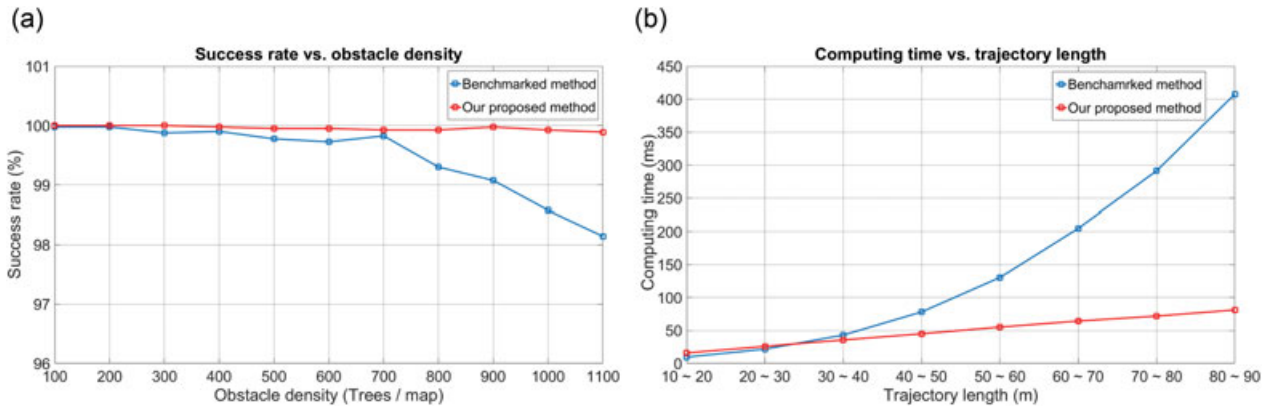

FIGURE 8 The comparison of the trajectory optimization against varying obstacle density (a) and trajectory length (b). The results of our proposed method and benchmark method (Campos-Macías et al., 2017) are, respectively, in red and blue. Four hundred trials are conducted for each case. (a) Successful rate against varying obstacle density and (b) computing time against varying length [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 2 Comparison of motion planning framework

Method	Length (m)	Objective	Path time (ms)	Traj. Time (ms)	Path Succ. Rate (%)	Traj. Succ. Rate (%)	Kino. Feasi. Rate (%)
Proposed	79.4201	3.4550	50.91	84.28	99.35	99.08	99.08
Previous	79.5530	3.5927	50.44	102.75	99.23	94.50	94.50
Benchmark2	81.8709	25.3661	97.43	73.47	100.00	98.35	76.35

as 600 trees/map and generate trajectories with different lengths by choosing the location of the target coordinates. The results are given in Figure 8a,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 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.

6.3.2 | Comparisons of the planning framework

We also conduct system-level comparisons with both the front-end path-finding and back-end trajectory generation submodules. The comparisons are made against our previous method and the state-of-the-art quadrotor planning framework (Richter et al., 2013), which is often called the waypoints-based method or unconstrained QP method and is denoted as **Benchmark2** here. In **Benchmark2**, 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 a closed form. Safety of the trajectory is achieved by iteratively adding new waypoints in the middle of the segment of the trajectory where a collision occurs. A snapshot of the comparison is given in Figure 7b. 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 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 **Benchmark2**. 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 **Benchmark2** is set as 20, which is decided by balancing the success rate and computation time.

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

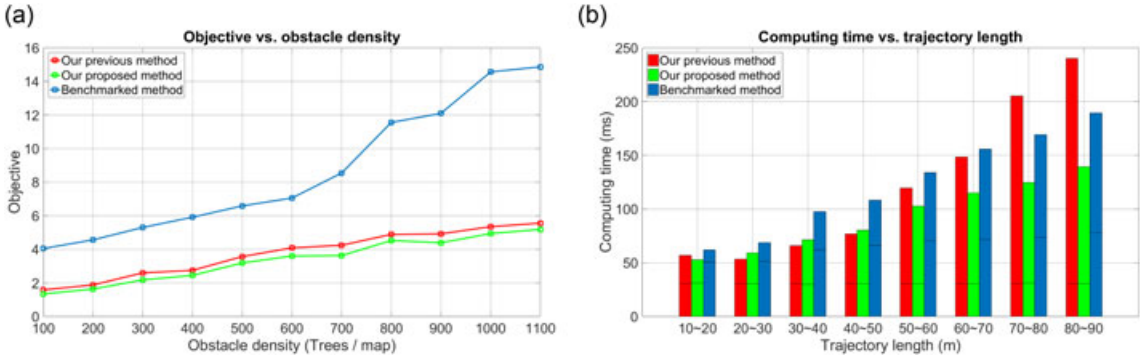

FIGURE 9 The comparison of the generated trajectories against varying obstacle densities (a) and trajectory length (b). The results of our proposed method are shown in green. The results of our previous method are plotted in red. And blue lines and bars indicate the results of the benchmark method. In (b), times below and above the horizon black line in each bar indicate time consumed in path finding and trajectory generation, respectively. (a) Objective against varying obstacle density and (b) computing time against varying trajectory length [Color figure can be viewed at wileyonlinelibrary.com]

We first conduct an overall comparison to show detailed performances of each method. The results are presented in Table 2. We randomly generate 10 different maps, which have 500 trees/map. In each map, 400 tests are performed with a random target. Maximum velocity and acceleration are 2 m/s and 2 m/s^2, respectively. The time for each piece of the trajectory is allocated using an average velocity of 1 m/s, and the time budget of our front-end modules for finding the path is 50 ms. The most noticeable difference is that compared with our corridor-based methods, the waypoint-based method generates trajectories with much higher objective values because 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

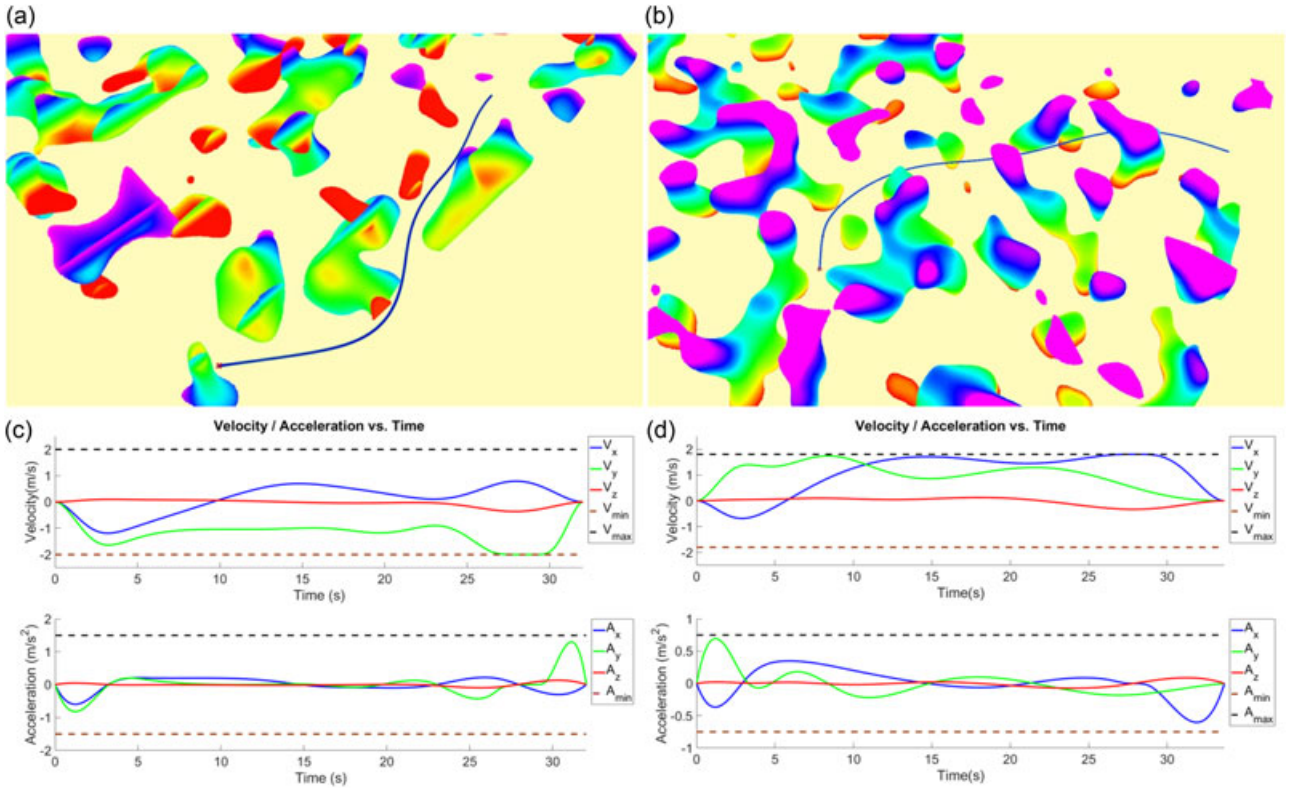

FIGURE 10 Two instances of trajectory generation. Velocities and accelerations in x, y, z axes are plotted against time in the bottom row of the figure and are shown to be entirely constrained within limits. Time is allocated using low aggressiveness in (a) and (c) and high aggressiveness in (b) and (d). (a) Trajectory generated with low aggressiveness, (b) trajectory generated with high aggressiveness, (c) velocity/acceleration with low aggressiveness, and (d) trajectory generated with low aggressiveness [Color figure can be viewed at wileyonlinelibrary.com]

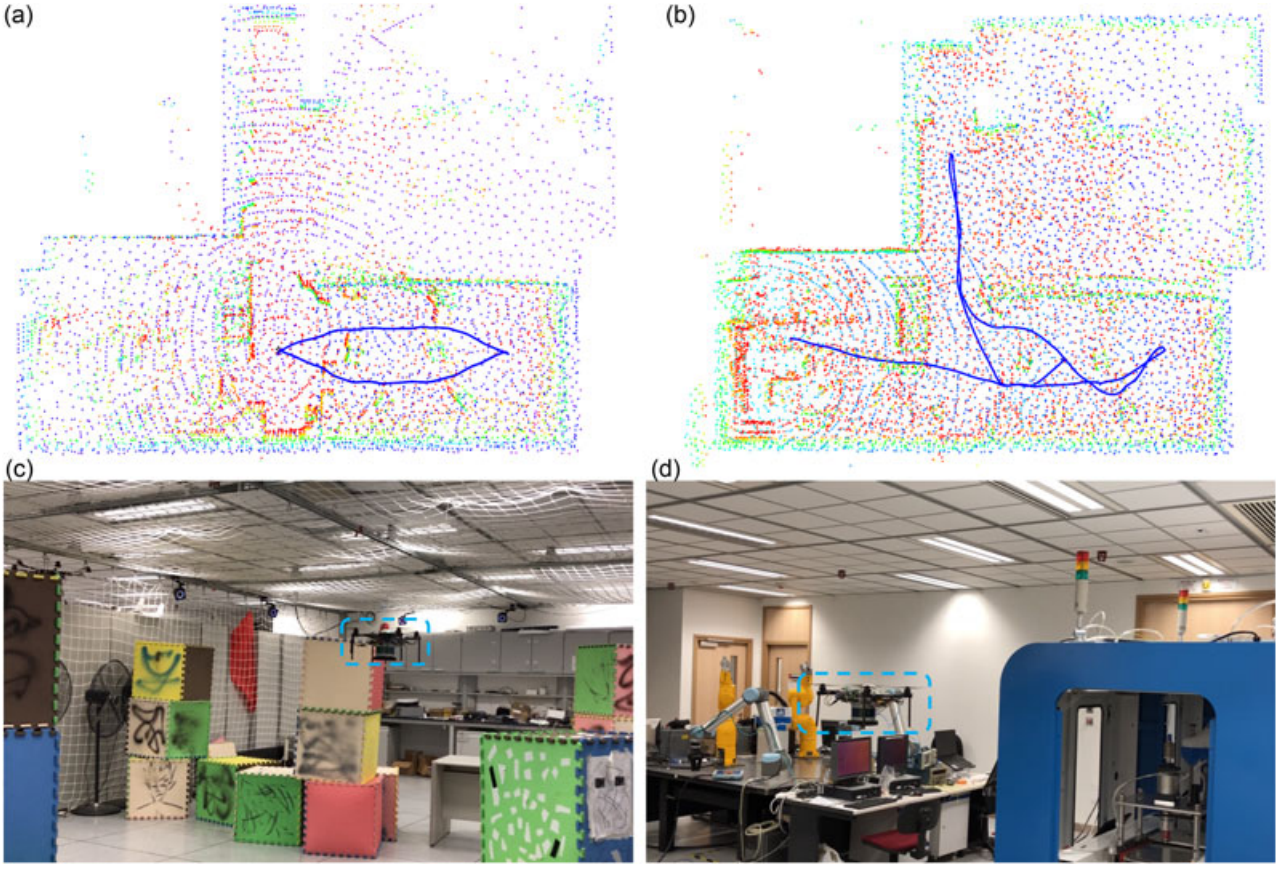

FIGURE 11 Indoor autonomous flight tests. Color code of obstacles indicates height. The generated trajectory and the tracking path of the quadrotor are shown as the red and the blue curves, respectively. (a) Tracking trajectory of the simple case, (b) tracking trajectory of the complex case, (c) snapshot of the indoor simple case, and (d) snapshot of the indoor complex case [Color figure can be viewed at wileyonlinelibrary.com]

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 2, the trajectory success rate means the collision-free rate, a 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 objective of **Benchmark2** is much higher than those of both our previous and proposed methods, as shown in Figure 9a. 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 Figure 9b, 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 Section 4, 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. Because 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, whereas our proposed method only needs to solve the program once, no matter how complicated the corridor is or whether a feasible solution exists.

6.4 | Simulated flights

We also validate the hard kinodynamic constraints enforced on trajectories in a simulation. As is shown in Figure 10, two instances are presented in a random Perlin-noised-map. In the first test (Figure 10a), time is allocated based on a low average

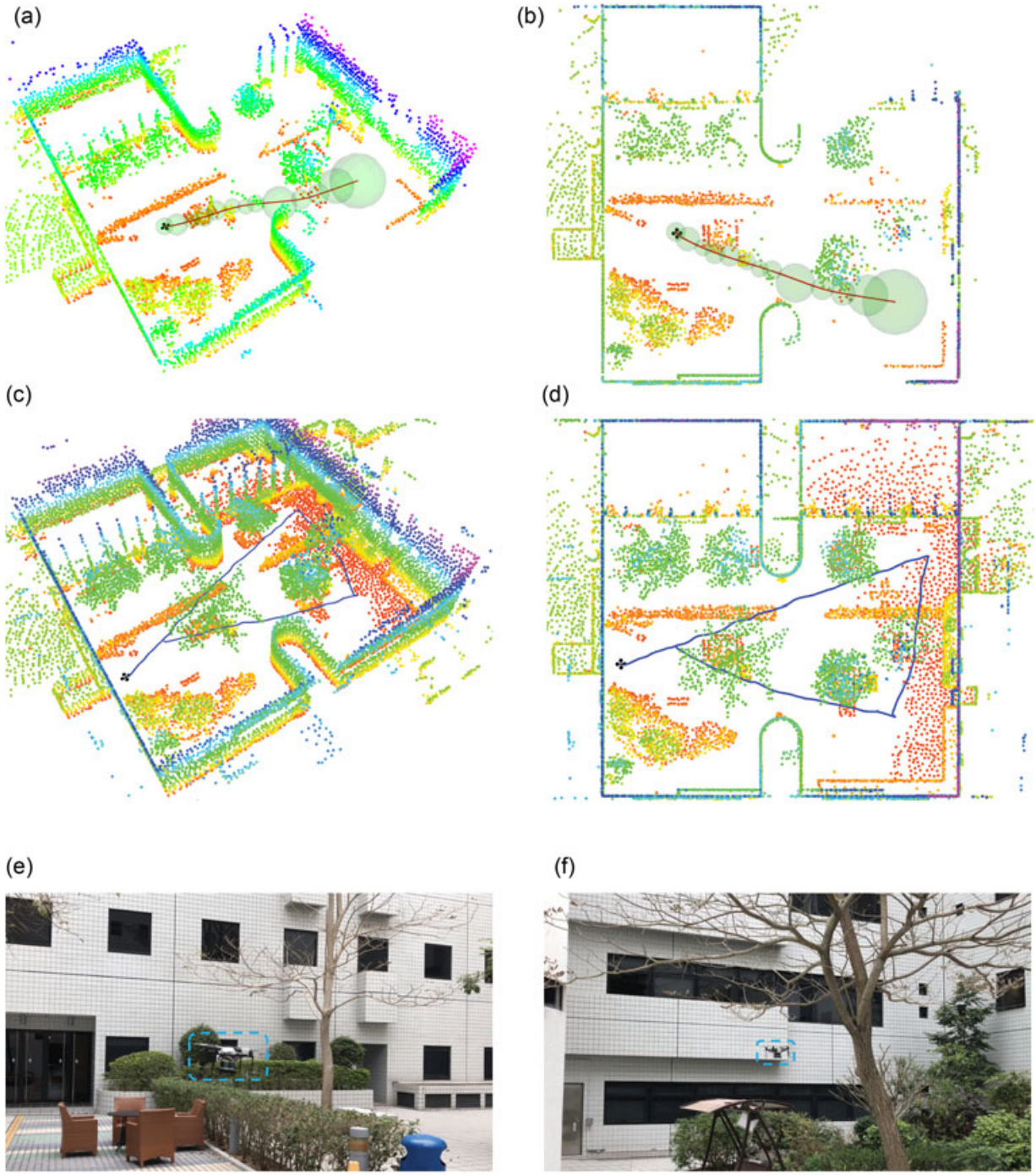

FIGURE 12 Outdoor autonomous flight tests in a garden courtyard. Markers are interpreted the same as in Figure 11, plus the flight corridor shown as a series of transparent green spheres. (a) Trajectory generated for the first target, (b) downward view, (c) overview of the autonomous flight, (d) downward view, (e) snapshot of the quadrotor in the test, and (f) snapshot of the quadrotor in the test [Color figure can be viewed at wileyonlinelibrary.com]

velocity (1 m/s) and the kinodynamic bounds are $\pm$ 2 m/s and $\pm$ 2 m/s^2. The second test (Figure 10b) is relatively difficult with an aggressive time allocation (1.5 m/s) and very tight kinodynamic bounds ($\pm$ 1.8 m/s and $\pm$ 0.75 m/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.

6.5 | Indoor experiments

Indoor experiments are carried out 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 $\pm$ 2 m/s and

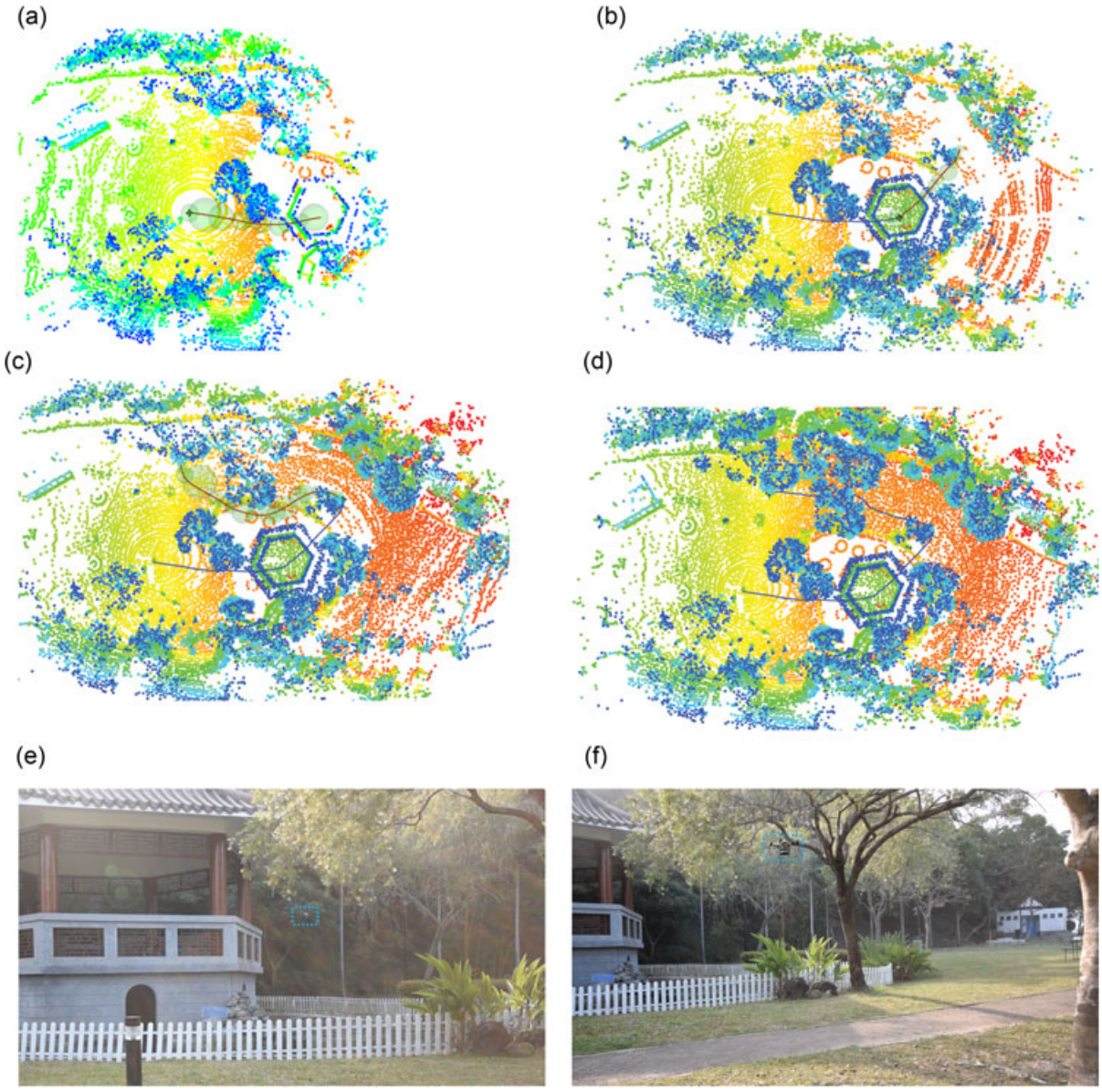

FIGURE 13 Outdoor autonomous flight tests in an unstructured environment. Markers are interpreted the same as in Figure 12. Trajectories generated after receiving targets are visualized. (a) Trajectory generated for the first target, (b) trajectory generated for the second target, (c) trajectory generated for the third target, (d) overview of the autonomous flight, (e) snapshot of the quadrotor in the test, and (f) snapshot of the quadrotor in the test [Color figure can be viewed at wileyonlinelibrary.com]

$\pm$ 1 m/s^2. And we reserve 0.15 m, considering the size of the quadrotor, and 0.2 m, 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 Figure 11. For online visualization of all generated trajectories and the replanning mechanism in the flight, we refer readers to the video attachment.

6.6 | 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 100 ms, 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.

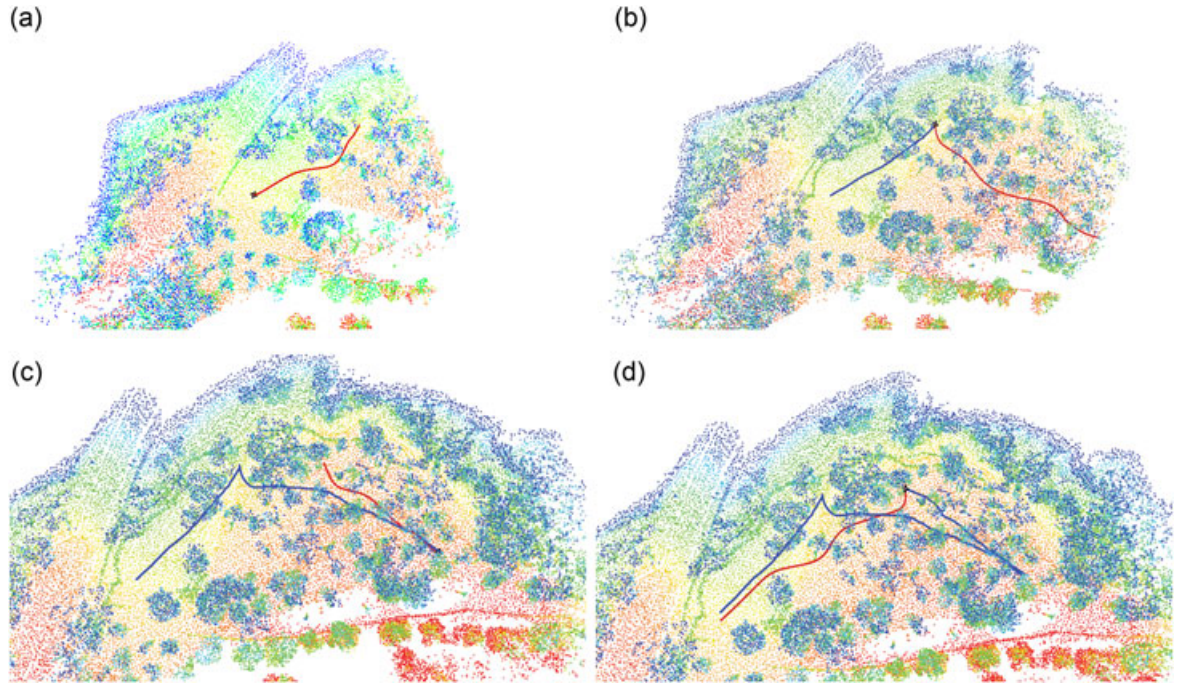

FIGURE 14 Outdoor autonomous flight tests in a complex forest. Markers are interpreted the same as in Figure 12. Four navigation targets are given sequentially in this experiment. Trajectories generated after receiving targets are visualized in (a–d). For clear visualization, the flight corridor is not shown in these figures, and points above 5 m of the point clouds are removed. Trajectory generated for (a) the first target, (b) the second target, (c) the third target, and (d) the fourth target [Color figure can be viewed at wileyonlinelibrary.com]

6.6.1 | 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 carried out 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 34.76 ms. The total traversal distance of the quadrotor is 73.71 m. Photos of the garden courtyard and the quadrotor in flight are shown in Figure 12e,f. The overview of the flight is shown in Figure 12c,d. Note that point clouds on the ground are removed for clear visualization.

6.6.2 | 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 29.71 ms. The total traversal distance of the quadrotor is 64.82 m. The results of the experiment are shown in Figure 13. Here, we only show trajectories generated after receiving targets.

6.6.3 | 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 37.79 ms. The total traversal distance of the quadrotor is 131.74 m. Photos of the environment and the quadrotor in flight are shown in Figure 15. The four trajectories generated after receiving targets are shown in Figure 14. 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.

6.7 | 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 (W. Gao & Shen, 2017), as shown in Figure 16. The quadrotor is equipped with dual fisheye cameras with one upward facing and another downward facing. The localization is carried out by visual-inertial fusion (Qin, Li, & Shen, 2018) in the onboard CPU, and the depths of each pixel in images are estimated using stereo matching in an additional graphics processing unit (GPU). Compared with the above-mentioned

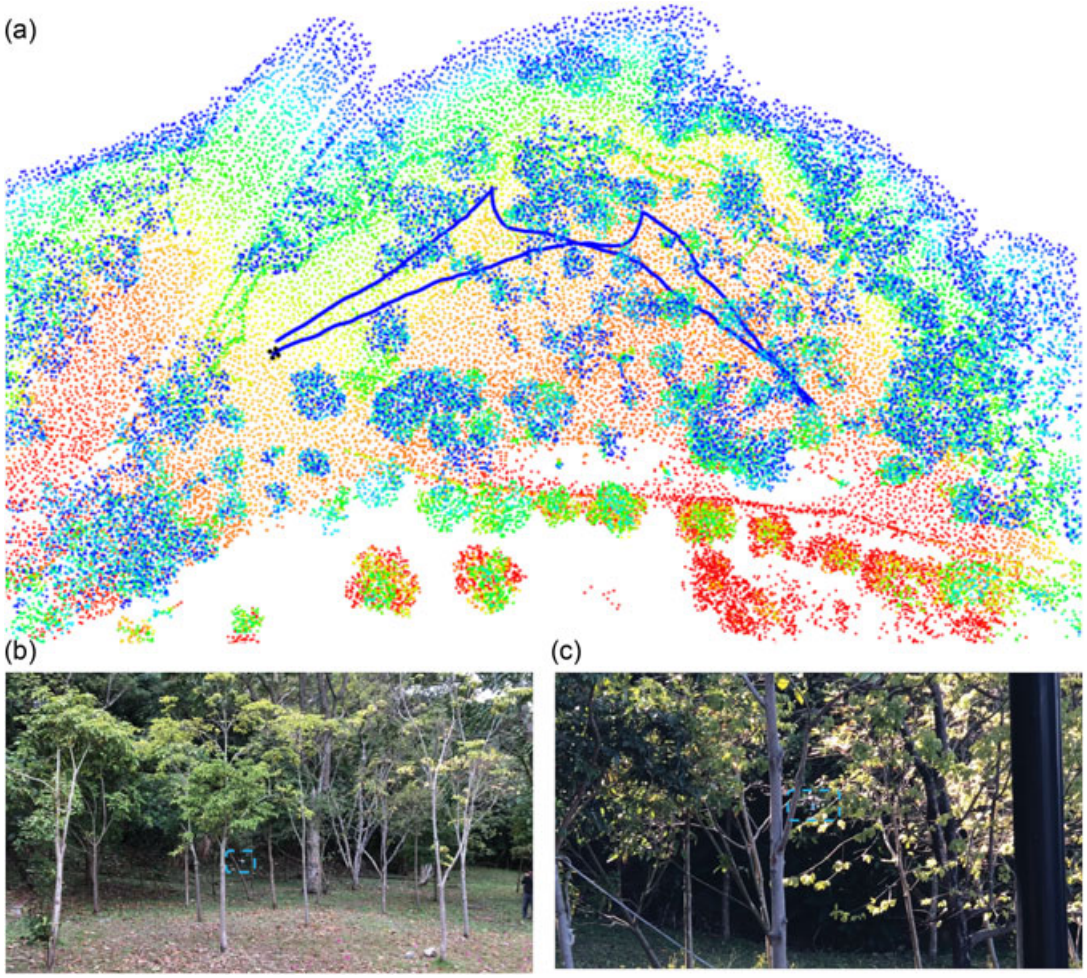

FIGURE 15 Overview of the quadrotor flights in the dense forest in (a). The total traversal distance of the quadrotor is 131.74 m in this test. Snapshots of the autonomous flights in the unknown cluttered forest in (b) and (c), corresponding to Figure 14. Markers are interpreted the same as in Figure 14. (a) Overview of the autonomous flight in the dense forest, (b) snapshot of the quadrotor in the test, and (c) snapshot of the quadrotor in the test [Color figure can be viewed at wileyonlinelibrary.com]

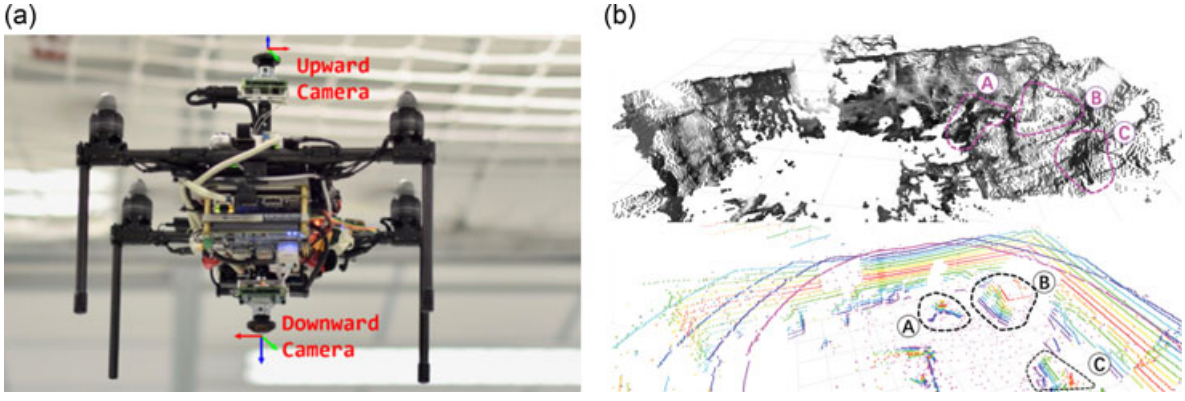

FIGURE 16 The dual fisheye quadrotor system in (a) and the comparison of the depth measurements against LiDAR in (b). The quadrotor is equipped with two fisheye cameras. Depth estimation is carried out by a Nvidia TX2 (NVIDIA Corporation, Santa Clara, CA) GPU (https://developer.nvidia.com/embedded/buy/jetson-tx2), and other processing is carried out by a mini i7 CPU. In (b), A, B, and C show corresponding objects. (a) The dual fisheye quadrotor platform and (b) comparing the depth with LiDAR measurements [Color figure can be viewed at wileyonlinelibrary.com]

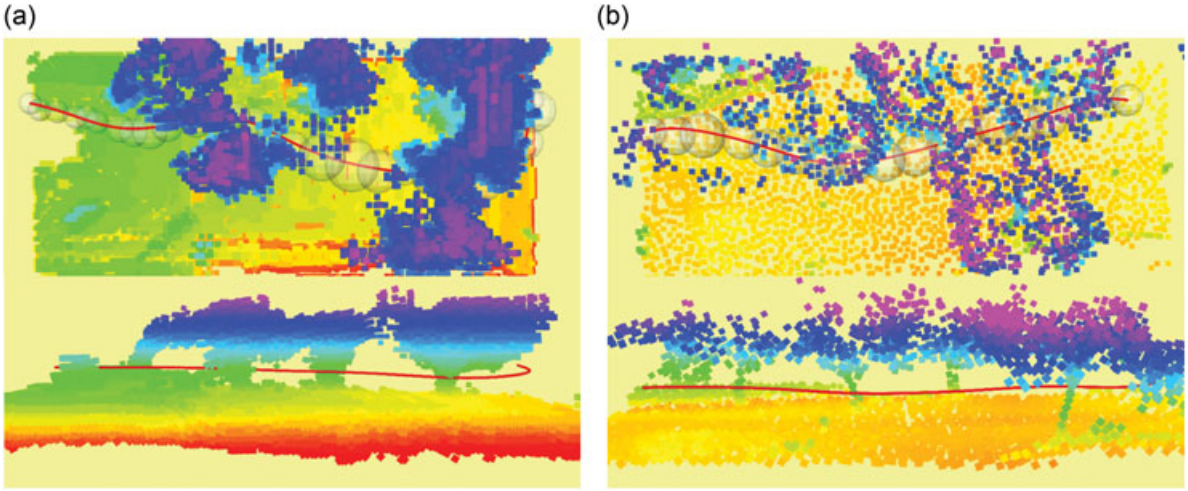

FIGURE 17 Comparison of the flight corridor and the trajectory generated using the dual fisheye cameras in (a) and using the Velodyne LiDAR in (b). Outliers in the vision-based mapping system occupy much free space and result in a longer flight corridor and trajectory. (a) Planning with the dual fisheye cameras and (b) planning with the LiDAR [Color figure can be viewed at wileyonlinelibrary.com]

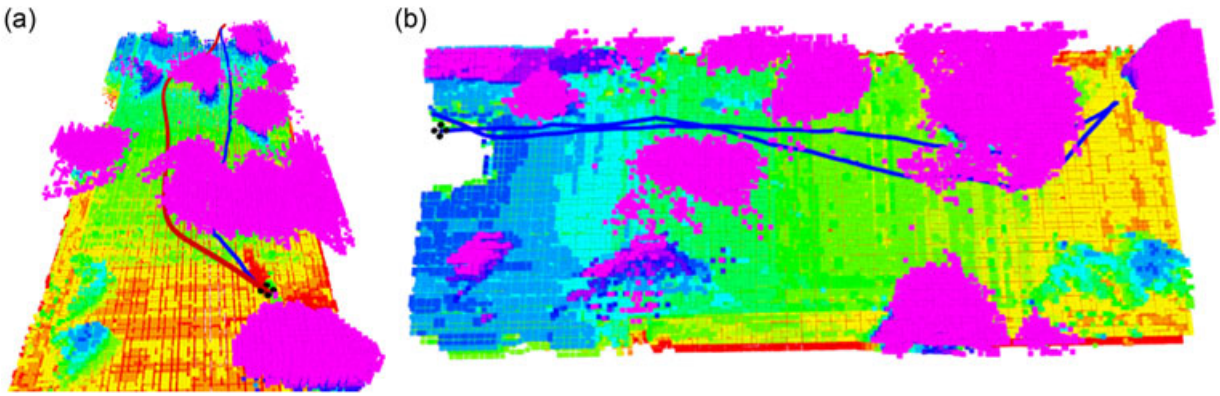

FIGURE 18 Autonomous flight test with a vision-based quadrotor. The trajectory generated for the second navigation is shown in (a) and the overview of the experiment is shown in (b). Markers are interpreted the same as in Figure 14. Online visualization is included in the video. (a) Trajectory generated in flight and (b) overview of the quadrotor path [Color figure can be viewed at wileyonlinelibrary.com]

LiDAR-based mapping system, the pixelwise 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 (Klingensmith, Dryanovski, Srinivasa, & Xiao,). Similar to point clouds, the spatially hashed voxels outputs 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 postprocessed map structure, such as the grid map. The overall latency from the images input to the spatially hashed voxels output is around 300–400 ms, whereas our LiDAR-based system only requires less than 100 ms to update the globally registered point cloud. Also, the sensing range of the vision-based navigation system is very limited at around 4–5 m, because of the onboard computational limitation and the resolution of images. Therefore, in the experiment, we set a relatively low speed (0.8 m/s) for the quadrotor. Another issue is about the accuracy. The dual fisheye system builds a map denser than by the LiDAR, as shown in Figure 17, because for every pixel the depth is calculated. However, it generates a much more noisy result with many false positive depth estimations. In Figure 17, 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 (35.4 m vs. 29.7 m). The result of the autonomous flight is presented in Figure 18 and more details are included in the video.

7 | CONCLUSION AND FUTURE WORK

7.1 | 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 autonomous quadrotor platform equipped with a 3D 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 postprocessed 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, which 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.

7.2 | Lessons learned

During the construction of our quadrotor platform and the field tests, we learned a major lesson: The control accuracy significantly affects the overall performance of the quadrotor system. If the control error is too significant, the quadrotor may deviate obviously from the planned trajectory, or even not pass the committed target. To address this issue from the planning perspective, one can reserve a large safety margin in the flight corridor or inflate obstacles. However, both these countermeasures sacrifice free space and therefore may result in no feasible trajectory existing in a cluttered environment. In contrast to compensating for the control error, we have opted to upgrade the motors and propellers of the quadrotor to provide a higher thrust-to-weight ratio (TWR). After testing, we finally chose the Snail motor based on the balance of energy consumption and thrust. As can be seen in the third outdoor flight test in the experimental video, our quadrotor tracks the generated trajectory accurately despite the strong wind in the testing forest.

7.3 | 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 prespecified 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.

Because 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 F. Gao and Shen (2016) to a vision-based quadrotor (Yang et al., 2017) and ground vehicle (C. Shen, Zhang, Li, Gao, & Shen, 2017) 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.

The laser-based localization and point clouds registration method in this paper is based on the assumption that the environment is static. Although we add an outlier and dynamic points removal mechanism in Section 3.3, we cannot guarantee all moving objects will be removed from registered point clouds. Highly dynamic moving objects not only affect the accuracy of the localization and mapping but also block free space in the configuration space of the quadrotor. We would like to adopt a more sophisticated method, such as that in Pomerleau, Krüsi, Colas, Furgale, and Siegwart (2014), to deal with dynamic environments in our future work.

ORCID

Fei Gao http://orcid.org/0000-0002-6513-374X
William Wu http://orcid.org/0000-0003-4508-0326
Shaojie Shen http://orcid.org/0000-0002-5573-2909

REFERENCES

Andersen, E. D. (2013). *On formulating quadratic functions in optimization models* (Technical Report TR-1-2013, MOSEK ApS). Retrieved from URL: http://docs.mosek.com/whitepapers/qmodel.pdf

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, *18*(9), 509–517.

Bry, A., Richter, C., Bachrach, A., & Roy, N. (2015). Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments. *The International Journal of Robotics Research*, *34*(7), 969–1002.

Bry, A., & Roy, N. (2011, May). *Rapidly-exploring random belief trees for motion planning under uncertainty*. In Robotics and Automation (ICRA), 2011 IEEE International Conference (pp. 723–730). IEEE.

Campos-Macías, L., Gómez-Gutiérrez, D., Aldana-López, R., de la Guardia, R., & Parra-Vilchis, J. I. (2017). A hybrid method for online trajectory planning of mobile robots in cluttered environments. *IEEE Robotics and Automation Letters (RA-L)*, *2*(2), 935–942.

Chen, J., Liu, T., & Shen, S. (2016). Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Stockholm, Sweden, 1476-1483.

Chen, Y., Cutler, M., & How, J. P. (2015). Decoupled multiagent path planning via incremental sequential convex programming. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 5954–5961.

Choi, S., Park, J., Lim, E., & Yu, W. (2012). Global path planning on uneven elevation maps. *Proceedings of the IEEE International Conference on Ubiquitous Robots and Ambient Intelligence, Daejeon, Korea*, 49–54.

Deits, R., & Tedrake, R. (2015). Efficient mixed-integer planning for UAVs in cluttered environments. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, USA, 42–49.

Dolgov, D., Thrun, S., Montemerlo, M., & Diebel, J. (2010). Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, *29*(5), 485–501.

Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer*, *22*(6), 46–57.

Faessler, M., Fontana, F., Forster, C., Mueggler, E., Pizzoli, M., & Scaramuzza, D. (2016). Autonomous, vision-based flight and live dense 3D mapping with a quadrotor micro aerial vehicle. *Journal of Field Robotics*, *33*(4), 431–450.

Fang, Z., Yang, S., Jain, S., Dubey, G., Roth, S., Maeta, S., & Scherer, S. (2017). Robust autonomous flight in constrained and visually degraded shipboard environments. *Journal of Field Robotics (JFR)*, *34*(1), 25–52.

Gammell, J. D., Srinivasa, S. S., & Barfoot, T. D. (2014). Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Chicago, IL, 2997–3004.

Gao, F., & Shen, S. (2016). Online quadrotor trajectory generation and autonomous navigation on point clouds. *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Lausanne, Switzerland, 139–146. Best Conference Paper Award.

Gao, W., & Shen, S. (2017). Dual-fisheye omnidirectional stereo. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 6715–6722.

Harabor, D. D., & Grastien, A. (2011). Online graph pruning for pathfinding on grid maps. *AAAI'11 Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 1114–1119. San Francisco, CA: AAAI Press.

Hauser, K. (2015). Lazy collision checking in asymptotically-optimal motion planning. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2951–2957.

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, *30*, 846–894.

Karaman, S., Walter, M. R., Perez, A., Frazzoli, E., & Teller, S. (2011, May). *Anytime Motion Planning using the RRT*. In Robotics and Automation (ICRA), 2011 IEEE International Conference (pp. 1478–1483). IEEE.

Klingensmith, M., Dryanovski, I., Srinivasa, S., & Xiao, J. (2015). Chisel: Real time large scale 3D reconstruction onboard a mobile device using spatially hashed signed distance fields. *Proceedings of the Robotics: Science and Systems (RSS)*.

LaValle, S. M., & Kuffner Jr, J. J. (2001). Randomized kinodynamic planning. *The International Journal of Robotics Research*, *20*(5), 378–400.

Lee, T., Leoky, M., & McClamroch, N. H. (2010). Geometric tracking control of a quadrotor UAV on SE(3). *Proceedings of the IEEE Control and Decision Conference (CDC)*, Atlanta, GA, 5420–5425.

Likhachev, M., & Ferguson, D. (2009). Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, *28*(8), 933–945.

Likhachev, M., Ferguson, D. I., Gordon, G. J., Stentz, A., & Thrun, S. (2005). Anytime dynamic a*: An anytime, replanning algorithm. *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 262–271.

Likhachev, M., Gordon, G. J., & Thrun, S. (2004). ARA*: Anytime A* with provable bounds on sub-optimality. *Advances in Neural Information Processing Systems* (pp. 767–774).

Lin, Y., Gao, F., Qin, T., Gao, W., Liu, T., Wu, W., … Shen, S. (2018). Autonomous aerial navigation using monocular visual-inertial fusion. *Journal of Field Robotics*, *35*(1), 23–51.

Liu, S., Atanasov, N., Mohta, K., & Kumar, V. (2017). Search-based motion planning for quadrotors using linear quadratic minimum time control. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2872–2879.

Liu, S., Watterson, M., Mohta, K., Sun, K., Bhattacharya, S., Taylor, C. J., & Kumar, V. (2017). Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments. *IEEE Robotics and Automation Letters (RA-L)*, *2*(3), 1688–1695.

Lynen, S., Achtelik, M. W., Weiss, S., Chli, M., & Siegwart, R. (2013). A robust and modular multi-sensor fusion approach applied to MAV navigation. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, 3923–3929.

Mellinger, D., & Kumar, V. (2011). Minimum snap trajectory generation and control for quadrotors. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, 2520–2525.

Mellinger, D., Kushleyev, A., & Kumar, V. (2012). Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Saint Paul, MN, 477–483.

Mohta, K., Watterson, M., Mulgaonkar, Y., Liu, S., Qu, C., Makineni, A. … Kumar, V. (2018). Fast, autonomous flight in GPS-denied and cluttered environments. *Journal of Field Robotics*, *35*(1), 101–120.

Oleynikova, H., Burri, M., Taylor, Z., Nieto, J., Siegwart, R., & Galceran, E. (2016). Continuous-time trajectory optimization for online UAV replanning. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, Korea, 5332–5339.

Pomerleau, F., Krüsi, P., Colas, F., Furgale, P., & Siegwart, R. (2014). Long-term 3D map maintenance in dynamic environments. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 3712–3719.

Qin, T., Li, P., & Shen, S. (2018). Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, *34*(4), 1004–1020.

Ratliff, N., Zucker, M., Bagnell, J. A., & Srinivasa, S. (2009). Chomp: Gradient optimization techniques for efficient motion planning. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 489–494.

Richter, C., Bry, A., & Roy, N. (2013). Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. Proceedings of the International Symposium of Robot. Research (ISRR), Singapore, 649-666.

Schulman, J., Ho, J., Lee, A., Awwal, I., Bradlow, H., & Abbeel, P. (2013). Finding locally optimal, collision-free trajectories with sequential convex optimization. Proceedings of the Robotics: Science and Systems, Vol. 9, 1-10.

Shen, C., Zhang, Y., Li, Z., Gao, F., & Shen, S. (2017). Collaborative air-ground target searching in complex environments. *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Shanghai, China, 230.

Shen, S., Michael, N., & Kumar, V. (2015). Tightly-coupled monocular visual-inertial fusion for autonomous flight of rotorcraft MAVs. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, Washington, USA.

van den Berg, J., Wilkie, D., Guy, S. J., Niethammer, M., & Manocha, D. (2012). LQG-obstacles: Feedback control with collision avoidance for mobile robots with motion and sensing uncertainty. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Saint Paul, MN, 346–353.

Webb, D. J., & van den Berg, J. (2013). Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Germany, 5054–5061.

Wurm, K. M., Hornung, A., Bennewitz, M., Stachniss, C., & Burgard, W. (2010). Octomap: A probabilistic, flexible, and compact 3D map representation for robotic systems. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, AK, USA, Vol. 2.

Yang, Z., Gao, F., & Shen, S. (2017). Real-time monocular dense mapping on aerial robots using visual-inertial fusion. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 4552–4559.

Zhang, J., & Singh, S. (2014). Loam: Lidar odometry and mapping in real-time. *Proceedings of the Robotics: Science and Systems (RSS)*, UCB, USA, 109–111.

Zhang, J., & Singh, S. (2018). Aerial and Ground-Based Collaborative Mapping: An Experimental Study. In *Field and Service Robotics* (pp. 397–412). Springer, Cham.

How to cite this article: Gao F, Wu W, Gao W, Shen S. Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments. *J Field Robotics*. 2018;1–24. https://doi.org/10.1002/rob.21842

APPENDIX A

TABLE A1 Index to multimedia extensions

Extension	Media type	Description
1	Video	It shows the experiments presented in this paper