Bubble Planner: Planning High-speed Smooth Quadrotor Trajectories using Receding Corridors

Yunfan Ren*, Fangcheng Zhu*, Wenyi Liu, Zhepei Wang, Yi Lin, Fei Gao and Fu Zhang

***Abstract*—Quadrotors are agile platforms. With human experts, they can perform extremely high-speed flights in cluttered environments. However, fully autonomous flight at high speed remains a significant challenge. In this work, we propose a motion planning algorithm based on the corridor-constrained minimum control effort trajectory optimization (MINCO) framework. Specifically, we use a series of overlapping spheres to represent the free space of the environment and propose two novel designs that enable the algorithm to plan high-speed quadrotor trajectories in real-time. One is a sampling-based corridor generation method that generates spheres with large overlapped areas (hence overall corridor size) between two neighboring spheres. The second is a *Receding Horizon Corridors* (RHC) strategy, where part of the previously generated corridor is reused in each replan. Together, these two designs enlarge the corridor spaces in accordance with the quadrotor's current state and hence allow the quadrotor to maneuver at high speeds. We benchmark our algorithm against other state-of-the-art planning methods to show its superiority in simulation. Comprehensive ablation studies are also conducted to show the necessity of the two designs. The proposed method is finally evaluated on an autonomous LiDAR-navigated quadrotor UAV in woods environments, achieving flight speeds over $13.7 m/s$ without any prior map of the environment or external localization facility.**

***Index Terms*—Motion and Path Planning; Autonomous Vehicle Navigation;**

I. Introduction

Quadrotors are proved to be one of the most agile platforms which perform increasingly complex missions in different scenarios [1]. However, high-speed flight in unknown environments is still an open problem. The limits on payload and onboard sensing make this task especially challenging for aerial robots [2]. To achieve high-speed flights, trajectory planning is of vital importance to ensure the safety, smoothness, and fast maneuvers facing unknown obstacles.

High-speed trajectory planning in unknown environments is of great challenge, especially in the re-planning phase where the high quadrotor speeds require extremely agile maneuvers to avoid newly-sensed obstacles. Existing (re-)planning methods [3–5] typically consist of a frontend that aims to find a guiding path (or flight corridor) and a backend that smooths the trajectory around the guiding path (or optimizes a smooth trajectory within the corridor). The main difficulty in this framework is how to design the frontend such that the re-planned guiding path (or flight corridor) is feasible: at least one dynamically-feasible and obstacle-free solution can be found in the backend optimization. A poorly-designed frontend may leave too little space for the quadrotor to avoid obstacles (*e.g.*, decelerate or make turns), hence leaving no dynamically feasible solution in the subsequent trajectory optimization. Another difficulty is the backend optimization, which needs to perform both temporal and spatial deformation in an efficient manner such that the maximal speed can be attained.

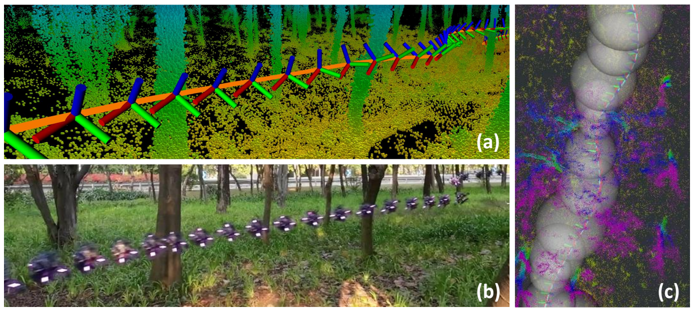

Fig. 1. High-speed navigation in the wild. (a) The generated point cloud map during the flight. (b) Composite images of the same flight. (c) The generated sphere-shaped flight corridor. Video is available at https://youtu.be/GpATPwibJ1k

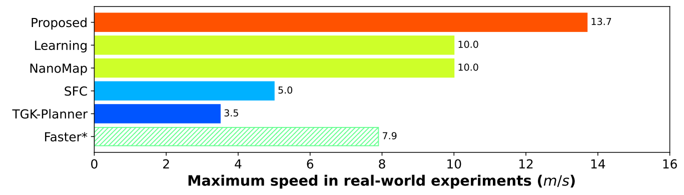

Fig. 2. Comparison of the maximum speed in real-world experiments (the flight speed of other methods are read from their original papers). The Faster* [2] baseline uses a motion capture system as state feedback and depth camera to detect obstacles. The proposed method and SFC [4] use a LiDAR for navigation while NanoMap [6], Learning [7], TGK-Planner [5] use a depth camera. Our approach reaches a maximum velocity over $13.7\ m/s$.

In this paper, we propose a robust and efficient motion planning algorithm to address the above issues systematically. The overall algorithm is based on a corridor approach. In the backend, we adopt a state-of-the-art minimum control effort optimization (MINCO) framework [8] to efficiently deform the trajectory temporal and spatial parameters. Our contribution in this paper mainly lies in the frontend, including:

1) A novel sampling-based corridor generation method that preserves large corridor volume by considering the size

*These two authors contribute equally in this work.

Y. Ren, F. Zhu, and F. Zhang are with the Department of Mechanical Engineering, University of Hong Kong {renyf, zhufc}@connect.hku.hk, fuzhang@hku.hk, W. Liu is with School of Electronics and Information Engineering, Harbin Institute of Technology, Shenzhen 180210215@stu.hit.edu.cn, Z. Wang and F. Gao are with the College of Control Science and Engineering, Zhejiang University {wangzhepei, fgaoaa}@zju.edu.cn Y. Lin is with Dji Co. ylinax@connect.ust.hk.

of each sphere as well as their overlapped spaces. The increased corridor volume allows more space for the quadrotor to maneuver (hence succeed) at high speeds.

2) A *Reciding Horizon Corridors* (RHC) scheme that re-uses corridor in the previous planning cycle. Specifically, in each re-plan, the first part of the flight corridor is directly from the previous planning cycle and the second part is generated according to newly-sensed obstacles. This receding scheme ensures the corridor in each re-plan always contains sufficient space for the quadrotor to maneuver from its current state, hence significantly improving the success rate and convergence speed of the re-plan process under high-speed flight.
3) A real-time planning system that integrates these two designs in frontend with MINCO [8] backend. Comprehensive benchmark comparison and ablation study are conducted in simulation to show the superiority of our system and the effectiveness of the two designs.
4) Implementation and validation of the proposed method on an actual fully autonomous quadrotor system. Multiple real-world tests show that our methods achieve flight speeds over 13.7 m/s (see Fig. 1).

II. Related Works

A. High-Speed Navigation in the Wild

Various approaches have been proposed to enable autonomous quadrotor flights in unknown environments. Florence *et al.* [9] propose a reactive planner, which takes depth image as input and selects the best trajectory from a pre-built motion primitives library. The work in [6] proposes an uncertainty-aware lazy search map called NanoMap on the reactive controller and achieves a maximum flight speed of 10 m/s. Although it has a low computation complexity, the pre-built set of motion primitives is relatively small, which is difficult to cover fine maneuvering skills that are necessary when the quadrotor is facing new unexpected obstacles during high-speed flights. Similar motion primitive-based method is used (as a frontend) by Zhou *et al.* [3], Liu *et al.* [10], Zhang *et al.* [11] and Kong *et al.* [12], which therefore suffer from similar drawbacks. Ye *et al.* [5] utilizes a frontend based on RRT* kino-dynamic sampling. Similar to the motion primitive methods, the sampled states are usually in low dimensions (*e.g.*, position and velocity) and few in numbers in order to ensure sufficient computation efficiency, making it very difficult to produce fine quadrotor maneuvers in high-speed flights. Unlike the previous methods [3–5], which typically have a frontend planning a rough path from the quadrotor current position to the target one and a backend which further refines the trajectory by optimization, Zhou *et al.* [13] proposed to plan a whole trajectory without considering any obstacle in the first stage and then locally modify the trajectory to fly around the detected obstacles. The local trajectory modification is achieved efficiently by directly incorporating a repulsive force from obstacles in the trajectory optimization cost function. The repulsive force is similar to a coarse-level distance field and hence suffers from the local minimum problem not suitable for high-speed trajectory planning. Another interesting method is proposed by Loquercio *et al.* [7], they use imitation learning to generate a trajectory directly from depth image and current state. Limited by the sensing range and noise, the success rate of their methods decreases when forward velocity is over 10 m/s. Compared with the methods mentioned above, our method achieves much higher flight speed in both simulation and experiments (see Fig. 2).

B. Corridor-based Trajectory Planning

To ensure safe quadrotor flights, corridor-based trajectory planning methods are popular in recent years. Chen *et al.* [14] build a discrete graph from an OctoMap structure and directly use free cubes in OctoMap as the corridor constraints. Liu *et al.* [4] uses polyhedrons to represent the free space, which is also called convex decomposition. Each cube or polyhedron on the flight corridor then imposes multiple linear hyperplane constraints in the subsequent trajectory optimization. Sphere-shaped corridor is also very commonly used. Compared with polyhedrons, a sphere imposes only one constraint in the trajectory optimization and can often be quickly obtained by *Nearest Neighbor Search* (NN-Search) using a KD-Tree structure. Gao *et al.* [15] propose a sphere-shaped corridor generation scheme under the RRT* framework. Ji *et al.* [16] propose a forward-spanning-tree-based spherical corridor generation scheme. These two methods can generate corridor in a relatively short time. However, their frontend only considers the connectivity of adjacent spheres. The found spheres often have small overlaps between adjacent ones, which over constrains the subsequent trajectory optimization and leaves very small space for the quadrotor to maneuver at high speeds. Another problem of them is the lack of explicit consideration of the quadrotor current speed, the resulting flight corridor often does not contain sufficient space for the quadrotor to maneuver from its current speed. The two problems will considerably reduce the feasible solution space and cause the backend optimization to fail. In contrast, our frontend attempts to find large individual spheres as well as their overlaps, while the receding scheme automatically incorporates the quadrotor current speed in each replan. These two designs greatly improve the success rate and convergence speed of the subsequent trajectory optimization.

Trajectory optimization with the corridor constraint is also well studied by some recent works. Ji *et al.* [16] use an alternative minimization method [17], iteratively insert waypoints to ensure that the trajectory completely falls in the corridor. However, the waypoints are selected heuristically, which leads to sub-optimal solutions. Mellinger *et al.* [18] use piece-wise polynomial to represent the trajectory and generate minimum-snap trajectory by solving a quadratic programming (QP) problem. The corridor constraints are used as inequality constraints in the QP. Gao *et al.* [15] use B-spline to represent trajectories and formulate the corridor constraints and trajectory optimization into a second-order cone programming (SOCP) problem. Both methods solve the optimization problem with hard constraints and have quite significant computation time. Our approach is most similar to [8]. The corridor constraints are first eliminated by a C^2 continuous barrier function. Then

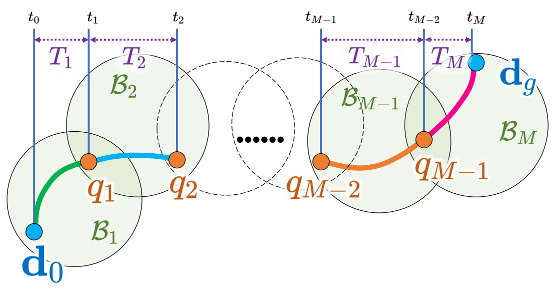

Fig. 3. The whole trajectory is composed of M pieces, each is contained in their respective sphere. The green areas $\mathcal{B}_i$ are the spheres. The orange points q_i are the intermediate waypoints, which are always constrained in the intersecting space of two adjacent spheres. T_i is the time allocation of each piece. $\mathbf{d}_0, \mathbf{d}_g$ are the given initial and goal states.

a spatial-temporal deformation is performed. The optimization problem is finally turned into an unconstrained one which can be solved by Quasi-Newton methods efficiently and robustly.

III. Preliminaries

In this section, we briefly go through the backend trajectory optimization used in our algorithm. As shown in Fig. 3, given a flight corridor $\mathcal{B}$ that consists of a sequence of overlapping spheres (each is denoted by $\mathcal{B}_i, i = 1, \cdots, M$, see Sec. IV-A), the goal of the trajectory optimization is to find a smooth trajectory $p(t) : [0, t_M] \mapsto \mathbb{R}^3$ over time duration t_M that connects the initial position $q_0 \in \mathbb{R}^3$ at time zero to the terminal one $q_M \in \mathbb{R}^3$ at time t_M and is completely contained in the sphere-shaped corridor $\mathcal{B}$.

In practice, the smoothness of the trajectory is quantitatively represented by the magnitude of its s-th order derivative $\|p^{(s)}(t)\|_2^2$ ($s = 4$ in experiments). Moreover, the trajectory $p(t)$ is usually decomposed into M pieces, each piece $p_i(t)$ is contained in sphere $\mathcal{B}_i$ for the time period $t \in [t_{i-1}, t_i]$, *i.e.*,

$$p(t) = p_i(t - t_{i-1}) \in \mathcal{B}_i, t \in [t_{i-1}, t_i] \tag{1}$$

Adjacent trajectory pieces $p_i(t)$ and $p_{i+1}(t)$ should meet at the same point $q_i \in \mathbb{R}^3$ at time t_i. Moreover, the trajectory $p(t)$ should start at a given initial state $\mathbf{d}_0$ (up to $(s-1)$-th order derivative) and terminate at a given goal state $\mathbf{d}_g$ (up to $(s-1)$-th order derivative). Considering these constraints and kinodynamic constraints (*e.g.*, speed and acceleration), the trajectory optimization can be formulated as

$$\min_{p(t)} \int_0^{t_M} \|p^{(s)}(t)\|_2^2 dt + \rho_T t_M \tag{2a}$$
$$s.t.\ \ p^{(0:s-1)}(0) = \mathbf{d}_0, p^{(0:s-1)}(t_M) = \mathbf{d}_g \tag{2b}$$
$$p(t_i) = q_i, \forall 1 \le i < M \tag{2c}$$
$$t_{i-1} < t_i, \forall 1 \le i \le M \tag{2d}$$
$$\|p^{(1)}(t)\|_2^2 \le v_{max}^2, \|p^{(2)}(t)\|_2^2 \le a_{max}^2, \tag{2e}$$
$$p(t) = p_i(t - t_i) \in \mathcal{B}_i, \forall 1 \le i \le M, t \in [t_{i-1}, t_i] \tag{2f}$$

where ρ_T is the weight penalizing the total trajectory time t_M such that the maximal allowed speed v_{max} can be attained.

The optimization in (2) can be solved in two steps: in the first step, we fix the intermediate way point $\mathbf{q} = (q_1, \ldots, q_{M-1})$ and time allocation vector $\mathbf{T} = (T_1, \ldots, T_M)$, where $T_i \triangleq t_i - t_{i-1} > 0$, and optimize only the first part (*i.e.*, the smoothness) of (2a) considering only the constraints in (2b) and (2c). Shown in [8], this optimization problem leads to an optimal solution where each piece $p_i(t)$ is a $(2s-1)$-th order polynomial and its coefficients are uniquely determined from $(\mathbf{q}, \mathbf{T})$, *i.e.*,

$$p_i(t) = \mathbf{c}_i(\mathbf{q}, \mathbf{T})^T \beta(t), t \in [0, T_i] \tag{3}$$

where $\mathbf{c}_i \in \mathbb{R}^{2s \times 3}$ is the coefficient matrix depending on $(\mathbf{q}, \mathbf{T})$ and $\beta(t) = [1, t, \ldots, t^{2s-1}]^T$ is the time basis function.

In the second step, the complete problem in (2) is optimized from the class of trajectories parameterized in (3). Since the trajectory in (3) naturally satisfies the constraints in (2b) and (2c), the complete optimization only needs to consider the constraints in (2d-2f). Even this, the constrained optimization is typically time-consuming. To address this issue, the MINCO framework [8] transforms it into an unconstrained optimization problem detailed as follows. First, the time constraints in (2d) are equivalent to $T_i > 0$, which can be prameterized as $T_i = e^{\tau_i}, 1 \le i \le M$ that always satisfies (2d) for $\tau_i \in \mathbb{R}$. Then, the feasibility constraints (2e) and (2f) can be softly penalized in the cost function by a C^2-continuous barrier function [19]:

$$\mathcal{L}_\mu(x) = \begin{cases} 0 & \text{if } x \le 0, \\ (\mu - x/2)(x/\mu)^3 & \text{if } 0 < x < \mu, \\ x - \mu/2 & \text{if } x \ge \mu. \end{cases} \tag{4}$$

where μ is a constant smoothness factor (0.02 in this paper), thus a finite weight for penalty can enforce the constraint at any specified precision. As a consequence, the optimization in (2) can be turned to an unconstrained form as:

$$\begin{aligned} \min_{\boldsymbol{\tau}, \mathbf{q}} \mathcal{J} = & \sum_{i=1}^{M} \left(\int_0^{T_i} \|p_i^{(s)}(t)\|_2^2 dt + \rho_T e^{\tau_i} \right) \\ & + \underbrace{\rho_{\text{vel}} \sum_{i=1}^{M} \int_0^{T_i} \mathcal{L}_\mu \left(\|p_i^{(1)}(t)\|_2^2 - v_{max}^2 \right) \mathrm{d}t}_{\text{velocity feasibility penalty}} \\ & + \underbrace{\rho_{\text{acc}} \sum_{i=1}^{M} \int_0^{T_i} \mathcal{L}_\mu \left(\|p_i^{(2)}(t)\|_2^2 - a_{max}^2 \right) \mathrm{d}t}_{\text{acceleration feasibility penalty}} \\ & + \underbrace{\rho_{\text{c}} \sum_{i=1}^{M} \int_0^{T_i} \mathcal{L}_\mu \left(\|p_i(t) - o_i\|_2^2 - r_i \right) \mathrm{d}t}_{\text{collision-free penalty}} \end{aligned} \tag{5}$$

where $\rho_{\text{vel}}, \rho_{\text{acc}}, \rho_{\text{c}}$ are the corresponding weight of maximum velocity, maximum acceleration and collision-free penalty, and o_i is the center and r_i is the radius of the i-th sphere. As shown in [8], all gradient of the objective (5) with respect to waypoints $\mathbf{q}$ and time allocation $\boldsymbol{\tau}$ can be computed analytically, so a Quasi-Newton method (*i.e.*LBFGS[1]) is used to solve the optimization problem effectively.

IV. Planner

In this section, we present the frontend design that enables high-speed trajectory optimization, which is the main contri-

[1]https://github.com/ZJU-FAST-Lab/LBFGS-Lite

bution of this paper.

A. Sphere-Shaped Corridor

As shown in Fig. 4, a sphere is defined by its center $o \in \mathbb{R}^3$, the nearest obstacle point $n \in \mathbb{R}^3$, and the radius:

$$r = \|o - n\|_2 - r_d \tag{6}$$

where r_d is the radius of the drone. During the trajectory optimization process, each piece of trajectory is constrained in the corresponding sphere to satisfy safety constraints.

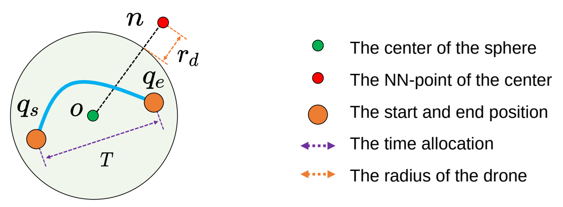

Fig. 4. The definition of one sphere and a piece of trajectory in it. q_s, q_e are the start and end point of the trajectory and T is the time allocation. o is the center of the sphere and n is the nearest obstacle point.

To generate a new sphere, we first build a KD-Tree with the obstacle point cloud. Then, for a given center of the sphere o, an NN-Search is performed on that KD-Tree to find the nearest obstacle point n, which then determines the radius as in (6). We call this process **GenerateOneSphere**(o), which will be used in the sequel.

B. Flight Corridor Generation

The main workflow of the flight corridor generation is described in Alg. 1, where a complete flight corridor $\mathcal{B}$ is generated from the given initial position p_0, goal position p_g, and a global guide path $\mathcal{T}$ generated by A*[20]. The algorithm initializes with a largest possible sphere $\mathcal{B}_{cur}$ around the initial position p_0 (Line 2-3). Then, in Line 5, a local guide point p_h is selected from the guide path $\mathcal{T}$, which is the nearest point out of the current sphere $\mathcal{B}_{cur}$, and a new sphere is generated by **BatchSample**$(p_h, \mathcal{B}_{cur})$ (Sec. IV-B1) and added to $\mathcal{B}$. This process repeats until the goal position p_g is included in the new generated sphere (Line 8-10).

With the found flight corridor $\mathcal{B}$, the initial waypoint position $\mathbf{q}$ and time allocation $\mathbf{T}$ are initialized by the function **WaypointAndTimeInitialization** $(\mathcal{B})$(Sec. IV-B2) and then optimized in the backend (Sec. III).

1) Batch sample: The problem of trajectory optimization under flight corridor constraints is highly non-convex, which means overly conservative constraints may lead to local-minimum or even infeasible solution when the quadrotor initial speed is high. Existing methods [15, 16] only considered the connectivity between two adjacent spheres. To preserve larger space for the quadrotor to maneuver hence improve the feasibility of the trajectory optimization (5) at high-speeds, we propose a novel batch sample method to generate a high-quality corridor. We consider this problem in the following aspects: **(a)** the volume of each sphere: a sphere with larger size can better approximate the real free space with fewer

Algorithm 1: GenerateCorridorAlongPath()

```
1  Notation: The flight corridor $\mathcal{B}$; global guide path $\mathcal{T}$; Initial
     and goal position: $p_0, p_g$; local guide point $p_h$
   Input: $\mathcal{T}, p_0, p_g$
   Output: $\mathcal{B}$
2  Initialize $\mathcal{B}_{cur}$ = GenerateOneSphere($p_0$);
3  $\mathcal{B}$.PushBack($\mathcal{B}_{cur}$);
4  while True do
5      $p_h$ = GetForwardPointOnPath($\mathcal{T}, \mathcal{B}_{cur}$);
6      $\mathcal{B}_{cur}$ = BatchSample($p_h, \mathcal{B}_{cur}$);
7      $\mathcal{B}$.PushBack($\mathcal{B}_{cur}$);
8      if $p_g \in \mathcal{B}_{cur}$ then
9          break;
10     end
11 end
12 WaypointAndTimeInitialization($\mathcal{B}$);
```

number of spheres, making the optimization problem less constrained, **(b)** the volume of the overlapped spaces between two adjacent spheres: as discussed in Sec. III, all waypoints of the trajectory are constrained in the intersecting space, a larger intersecting space means more freedom for the optimization process.

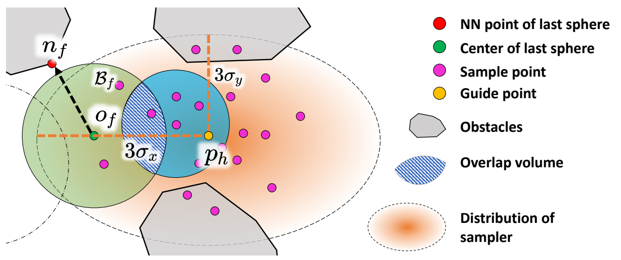

Fig. 5. The green circle $\mathcal{B}_f$ is the sphere generated in the last round of batch sample. The yellow point is the guide point p_h. The purple points are the sampled points according to the probability distribution represented by the orange-shaded area. The blue circle is the best sphere in this round.

Algorithm 2: BatchSample()

```
1  Notation: Last sphere $\mathcal{B}_f$; Guide point $p_h$; Best sphere in
     this round $\mathcal{B}_{best}$; Random sampler $\mathcal{S}$; Maximum sample
     num $K$; Safe distance $r_d$; Priority queue sorted by sphere's
     score: $\mathcal{Q}$;
   Input: $\mathcal{B}_f$, $p_h$, $K$
   Output: $\mathcal{B}_{best}$
2  Initialize: $\mathcal{S}$.init($\mathcal{B}_f$, $p_h$), $k = 0$ ;
3  while $k < K$ do
4      $p_{cand}$ = $\mathcal{S}$.GetOneRandomSample();
5      $\mathcal{B}_{cand}$ = GenerateOneSphere($p_{cand}$);
6      $\mathcal{B}_{cand}$.score = ComputeScore($\mathcal{B}_{cand}$);
7      $\mathcal{Q}$.PushBack( $\mathcal{B}_{cand}$ );
8      $k = k + 1$;
9  end
10 if $\mathcal{Q}.empty()$ then
11     return BatchSampleFailed;
12 end
13 $\mathcal{B}_{best}$ = $\mathcal{Q}$.top();
```

The sampling process is shown in Alg. 2. We first initialize the sampler $\mathcal{S}$ in Line 2. As shown in the orange area of Fig. 5, the sampler generates a random candidate point $p_{cand} \in \mathbb{R}^3$ under a 3D Gaussian distribution $N(\mu, \Sigma)$, where the mean

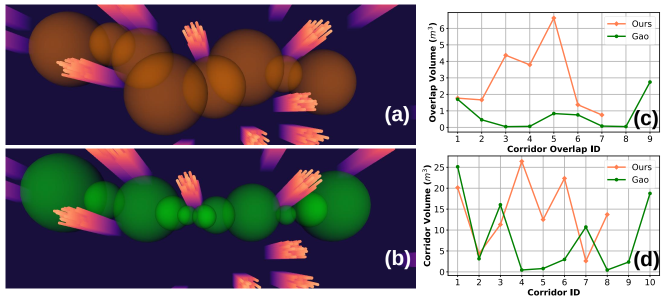

Fig. 6. Corridor generation comparison. (a) The proposed method generates fewer spheres with larger radius and overlaps, consuming only 0.74 ms. (b) The corridor generated by Gao *et al.* [15] with a RRT* like method, consuming 100 ms. (c) The comparison of overlap volume, the minimum overlap volume of the proposed method is 0.75 m^3 while Gao's minimum overlap volume is 0.049 m^3. (d) The volume of each sphere.

is set at the guide point $\mu = p_h$ and the covariance is set as $\Sigma = \text{diag}\left(\sigma_x, \sigma_y, \sigma_z\right)$, $\sigma_x = \frac{1}{2}\left\|o_f - p_h\right\|_2, \sigma_z = \sigma_y = \frac{1}{2}\sigma_x$, where o_f is the center of last sphere and the σ_x direction is aligned with the direction of $o_f - p_h$.

Then in Line 3-9, a total number of K points (called a batch) are randomly sampled with $\mathcal{S}$, each has its score computed by the function **ComputeScore**($\mathcal{B}_{cand}$) defined below:

$$\text{Score} = \rho_r V_{\text{cand}} + \rho_v V_{\text{inter}} \tag{7}$$

where $\rho_r, \rho_v \in \mathbb{R}_+$ are positive weights, V_{cand} is the volume of the candidate sphere $\mathcal{B}_{cand}$ and V_{inter} is the overlapped volume between $\mathcal{B}_{cand}$ and $\mathcal{B}_f$. Finally, the best sphere with the highest score is selected in Line 13. As shown in Fig. 6, compared with Gao *et al.*[15], the proposed method can better approximate the real free space, with fewer sphere and larger sphere sizes. Further, our algorithm has lower computational complexity. In the same environment shown in Fig. 6, the proposed method only takes 0.74 ms to generate the corridor, while it takes 100 ms for Gao's method.

2) Waypoint and Time Initialization: For a given flight corridor $\mathcal{B}$, we adopt a *Default Initialization* strategy, where the waypoint are initialized as the center of the overlap space between two adjacent spheres (pink points in Fig. 7(c)), and the time allocation is initialized as $T_i = \frac{\|q_i - q_{i-1}\|_2}{v_{max}}$.

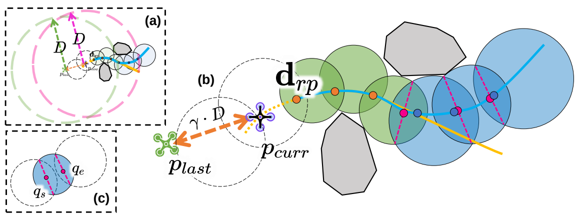

Fig. 7. The receding horizon corridors strategy. (a) The green and pink dashed circle are respectively the planing horizon in last and current re-plan. (b) Spherical corridor in green are previously generated, using the *Hot Initialization*. And corridor in blue are newly generated, using the *Default Initialization*. (c) The pink point is the center of the overlap area, which is used by the *Default Initialization*.

C. Receding Horizon Corridors in Replan

During a high-speed flight in an unknown environment, the quadrotor needs to re-plan frequently to avoid newly sensed obstacles. We use a distance-triggering re-planning strategy. Specifically, the trajectory is planned (both frontend corridor generation and backend optimization) in a fixed distance D (*i.e.* planning horizon) depending on the sensing range. Denote the position of last re-plan as p_{last} and current quadrotor position as p_{curr}. The replan process is triggered if $\|p_{last} - p_{curr}\|_2 > \gamma \cdot D$, where $\gamma \in [0, 1]$ is a constant ratio. In this way, as the drone moves forward, the newly sensed obstacle can be actively handled by the re-plan process. A replan is also triggered when the current trajectory under executing is found to collide with any obstacles.

A major challenge in the replan occurs when the quadrotor speed is high, which requires sufficient space for the quadrotor to maneuver such that the newly sensed obstacles can be avoided successfully. Corridor generation without considering the quadrotor's current state [4, 15, 16] often causes too small feasibility region in the trajectory optimization (2), which is difficulty (or even infeasible) to solve (*e.g.*, by optimizing (5)). Another problem is that with the increase of the current speed, the objective function becomes highly non-convex, causing the optimization with the *Default Initialization* to easily stuck at a local minimum.

We solve these problems by a *Receding Horizon Corridors* (RHC) strategy shown in Fig. 7. The key is to re-use a few corridors from the previous planning cycle in the current replan. Concretely, when a new re-plan is triggered, the nearest future waypoint $\mathbf{d}_{rp}$ in $\mathbf{q}$ is selected as the initial state. A few sphere corridors after $\mathbf{d}_{rp}$ will be re-used to consists the first part of the new corridor, followed by new corridor reaching the current planning horizon D. This receding scheme ensures the corridor in each replan always contains sufficient space for the quadrotor to maneuver from its current state (since the current quadrotor state is on the previous trajectory contained in the previous corridor), hence significantly enlarges the feasibility region in the backend trajectory optimization. In experiments, we re-use corridors that fall within a certain distance (*e.g.*, $3m$) of the current quadotor position p_{curr}. Furthermore, to speed up the trajectory optimization and mitigate the local minimum issue, the waypoints $\mathbf{q}$ and time allocation $\mathbf{T}$ contained in the re-used corridor, which were optimized in the previous planning cycle, are used to initialize the current trajectory optimization (*i.e. Hot Initialization*). The waypoints and time allocation in the newly generated spheres are still initialized by the default scheme (Sec. IV-B2)

V. Experiments

A. Benchmark Comparison

In this section, we compare the proposed method with a most recent planning work based on imitation learning [7] (Learning), and two model-based planning methods evaluated by it, including a frontend-backend type optimization-based method from Zhou *et al.* [21] (FastPlanner) and a reactive planner designed for high-speed flight from Florence et al.[9] (Reactive). We evaluate the performance of our method in a simulated forrest environment used by the learning method. Due to the unavailability of the simulation environment used by the original work [7], we reproduce the environment according to their description. Specifically, the forest has trees distributed in a rectangular region $R(l, w)$ of width w and length l, the origin lies in the center of R. Trees are randomly placed according to a homogeneous Poisson point process P with the intensity δ $tree/(m^2)$. The sensor input in the simulation includes a simulated LiDAR point cloud, with the sensing range of 8 m at 30 Hz (see green points in Fig. 8(b)). The quadrotor full state is assumed to be known in order to eliminate the influence of state estimation.

We use exactly the same configuration in [7] to make a fair comparison: $w = 30$ m and $l = 60$ m, and the start zone of the drone is at $(-l/2, 0)$, the goal position $(l/2, 0)$. Three different tree densities with $\delta = 1/49$ (low), $\delta = 1/36$ (medium), and $\delta = 1/25$ (high) are tested. In each experiment, we use different random seed to generate different simulated map. One flight is considered to be successful only if the drone reaches the goal without violating the velocity, acceleration or collision-free constraints. The results are shown in Fig. 9. Similar to [7], we test our method 10 times in each different density or speed and compute the success rate of each, and the results of other baseline are directly obtained from [7]. Noting that in [7], the maximum mass-normalized thrust of the simulated drone is limited to 35.3 m/s^2, while we limit our simulated drone to 15 m/s^2. As can be seen, our approach outperforms others in all cases, even with a lower thrust limit. Moreover, compared with Loquercio *et al.* [7], the proposed method generates much smoother trajectories, which is usually easier to track (see Fig. 8).

B. Ablation Study

To further validate each module of the proposed method, we compare our method in detail with Gao *et al.* [15], which generates sphere-shaped corridor in a RRT* style and optimizes a minimal snap trajectory with fixed time allocation. We use the same simulated map configuration mentioned in Sec.V-A, but further add tests with $\delta = 1/12$ (super high). The key three elements of our approach includes trajectory optimization in Sec. III (MINCO), front-end corridor generation in Sec. IV-B (Front), and receding horizon corridors strategy (RHC) in Sec. IV-C. A series of ablation study is performed, and the results are shown in Fig. 10. *Gao* is the original version from [15]. This method fails to generate trajectory with a speed over 2 m/s due to the inability to optimize time allocation in backend. To fix this issue, we replace the backend of *Gao* by MINCO (*Gao+MINCO*) and compare it with our method without RHC strategy (*Ours (Front+MINCO)*). The performance of the two are very close, showing that MINCO is able to generate more aggressive trajectories and that our frontend alone does not improve the success rate much. Furthermore, we incorporate the RHC strategy to the method *Gao* (*Gao + MINCO + RHC*) and compare it with our full algorithm (with both frontend and RHC). As can be seen, each method with RHC has a significantly higher success rate at high speeds on all map densities, verifying the effectiveness of the RHC strategy. Moreover, our full algorithm with our frontend (*Ours(Front + MINCO + RHC)*) achieves higher success rate than *Gao* with the same MINCO and RHC strategy (*Gao+MINCO+RHC*), showing the effectiveness of our frontend in the overall planning system.

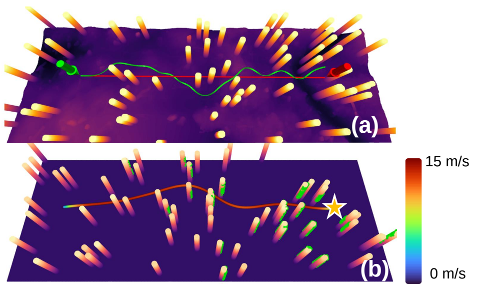

Fig. 8. (a) The executed trajectory in Loquercio *et al.* [7]. (b) The executed trajectory with the proposed method, colored with the norm of velocity from 0 m/s to 15 m/s. The yellow star is the initial position of the drone and the green points are the simulated LiDAR points.

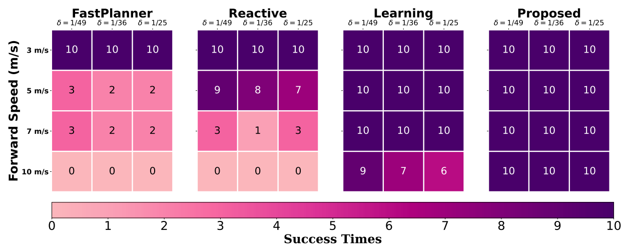

Fig. 9. The success rate comparison for different methods. The proposed method keeps a high success rate in all test environments with different densities of the forest.

TABLE I
RUN TIME COMPARISON

Method	Components	μ [ms]	σ [ms]	Total [ms]
Fast-Planner [3]	Mapping	38.20	6.9	42.92
	Planning	4.72	1.6	
Gao [15]	Mapping	12.78	3.69	157.01
	Planning	154.23	40.6	
Ours	Mapping	3.16	0.76	**4.69**
	Planning	1.53	0.63	
Ours (onboard)	Mapping	8.97	6.51	13.31
	Planning	4.34	2.33	

C. Run Time Analysis

In this section, we compare the run time of the proposed method with the baseline. We test our method both on the desktop computer, with a 2.90 GHz Intel i7-10700 CPU, and an onboard computer with a 1.1 GHz Intel i7-10710U CPU. The baseline FastPlanner [3] and Gao [15] are tested on the same desktop computer. The test environment is a simulated forest with $\delta = \frac{1}{25}$ shown in Fig. 8(b). The computation time is divided into two parts: mapping and planning. For FastPlanner, the mapping process includes the building of a Euclidean signed distance field (ESDF), and planning includes frontend path-search and backend trajectory optimization. For Gao's method, the mapping process includes a static KD-Tree update, and the planning includes corridor generation and SOCP optimization. For the proposed method, the mapping includes the update of an OctoMap [22] (no ray-casting) and an incremental KD-Tree (*i.e.*, ikd-tree [23]). The planning includes frontend A* search, corridor generation, and trajectory optimization. As shown in Tab. I, the proposed method enjoys much lower

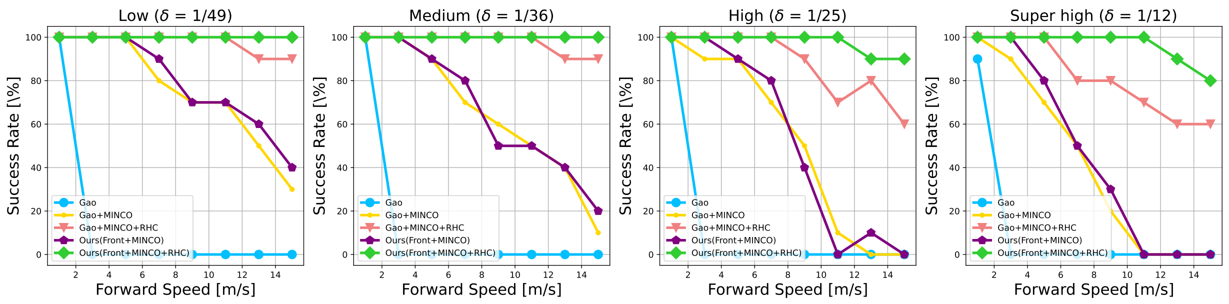

Fig. 10. The ablation study of the proposed method. The green line is the proposed method (*Ours(Front+MINCO+RHC)*). The blue line (*Gao*) is the original version of Gao *et al.*'s work [15]. The yellow line (*Gao+MINCO*) uses corridor generation from [15], but with trajectory optimization replaced by MINCO in Sec.III. The purple line (*Ours(Front + MINCO)*) is our method without the RHC strategy. The red line (*Gao+MINCO+RHC*) uses Gao *et al.*'s corridor generation method and the same MINCO optimization and RHC strategy as ours.

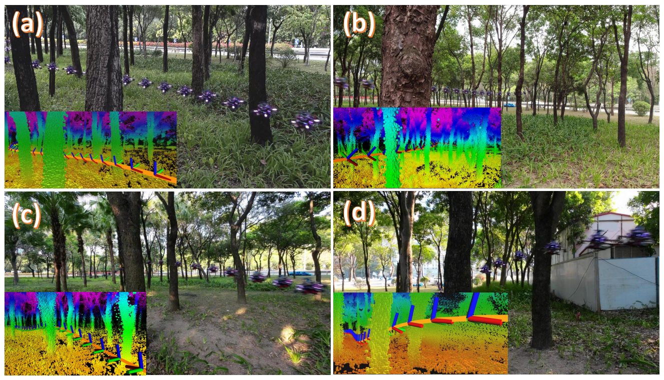

Fig. 11. The composite images and LiDAR point cloud in different tests. The maximum speed varis from $5 \sim 14 m/s$.

computational complexity, which can replan at over 50 Hz even on the onboard platform.

D. Real-world Experiments

To verify our planning method in real-world environments, we build a LiDAR-based quadrotor platform. The platform has a total weight of 1.45 kg and can produce a maximum thrust over 60 N, resulting in a thrust-to-weight ratio of 4.1. For localization and mapping, we use the Livox[2] Mid360 LiDAR[3] running FAST-LIO 2.0 [24], which provides 100 Hz high-accuracy state estimation and 25 Hz point cloud, the trajectory tracking controller is an on-manifold model predictive controller in [25], the planning horizon is set to $D = 15\ m$ and replan ratio $\gamma = 0.4$. All perception, planning, and control algorithm are running on an Intel NUC with CPU i7-10710U in real-time. We have done over ten experiments in a forest environment with maximal speed ranging from 5 m/s to 14 m/s, and all the experiments succeeded except one due to a controller failure. Fig. 11 shows the experiment environment and some of the test flights. More visual illustration of the experiments is shown in our video[4]. Due to the space limit, we investigate two typical flights called *Test 1* and *Test 2*. For *Test 1*, we limit the maximum velocity to 8 m/s and the maximum acceleration to 8 m/s^2. The planned trajectory is shown in Fig. 12 and Fig. 13, where the whole executed trajectory is of 109 m, and the executing time is about 15.56 s, leading to an average speed 7.05 m/s.

[2]https://www.livoxtech.com/
[3]https://view.inews.qq.com/a/20210903V07GSH00
[4]https://youtu.be/GpATPwibJ1k

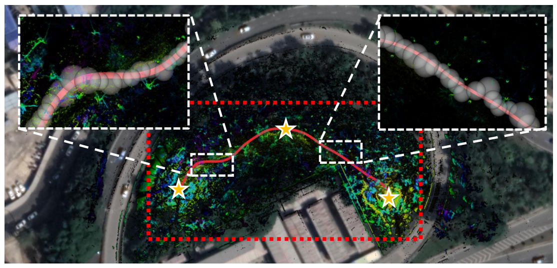

Fig. 12. Satellite view of the outdoor test site, the red rectangle area is $120\ m \times 60\ m$. The executed trajectory of *Test 1* is shown in red, which is about 109.85 m, with maximum speed 8 m/s and average speed 7.02 m/s. Yellow stars are two given waypoints of *Test 1*.

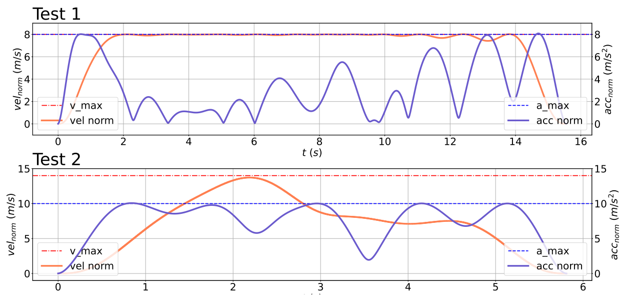

Fig. 13. The norm of velocity and acceleration. The executed trajectory of *Test 1* is about 109.89 m, and *Test 2* is about 40.62 m. The average speed is about 7 m/s for both test, and the maximum speed is over 13.7 m/s in *Test 2*.

In *Test 2*, a more agile flight is performed where the maximum velocity is 14 m/s and the maximum acceleration is 10 m/s^2. The planned trajectory is shown in Fig. 13 with a maximum speed over 13.7 m/s. To the best of our knowledge, this is the highest speed that a fully autonomous quadrotor can achieve in a real-world, cluttered, and unknown environment.

VI. Conclusion and Future Work

In this paper, we propose a novel motion planning algorithm that generates smooth, collision-free and high-speed trajectories in real-time. The whole planning system can work with fully onboard sensing and computation at a replan frequency over 50 Hz. To enable high-speed flight in the wild, we proposed two novel designs. One is a sampling-based sphere-shaped corridor generation method, which can generate high-quality corridor (*i.e.* larger size and bigger overlaps) in relatively short time. Another one is a *Receding Horizon Corridors* strategy, which fully utilizes previously generated corridors and the optimized trajectory. With these designs, the proposed method significantly increases the re-plan success rate in high-speed cases.

One limitation of our algorithm is that the re-used corridors from the last planning cycle are not guaranteed to be obstacle-free due to newly sensed obstacles that may be occluded in previous LiDAR measurements. This will cause the re-used corridor to be discarded and hence occasionally lower the success rate when the environment is extremely cluttered. This limitation can be overcome by placing the first few corridors of a (re-)plan in known free spaces (instead of free and unknown spaces), so that these free corridors can be safely re-used in the next planning cycle. Restraining the first few corridors in free spaces also enables to plan a safe backup trajectory like [2] which guarantees a safe flight. In the future, we will explore these designs and extend the method to more different missions and environments.

Acknowledgment

The authors gratefully acknowledge DJI for fund support and Livox Technology for equipment support during the whole project. The authors would like to thank Guozheng Lu and Wei Xu for the helpful discussions.

References

[1] Jon Verbeke and Joris De Schutter. Experimental maneuverability and agility quantification for rotary unmanned aerial vehicle. *International Journal of Micro Air Vehicles*, 10(1):3–11, 2018.

[2] Jesus Tordesillas, Brett T Lopez, Michael Everett, and Jonathan P How. Faster: Fast and safe trajectory planner for navigation in unknown environments. *IEEE Transactions on Robotics*, 2021.

[3] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*, 4(4):3529–3536, 2019.

[4] Sikang Liu, Michael Watterson, Kartik Mohta, Ke Sun, Subhrajit Bhattacharya, Camillo J Taylor, and Vijay Kumar. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *IEEE Robotics and Automation Letters*, 2(3):1688–1695, 2017.

[5] Hongkai Ye, Xin Zhou, Zhepei Wang, Chao Xu, Jian Chu, and Fei Gao. TGK-Planner: An efficient topology guided kinodynamic planner for autonomous quadrotors. *IEEE Robotics and Automation Letters*, 6(2):494–501, 2021.

[6] Peter R Florence, John Carter, Jake Ware, and Russ Tedrake. Nanomap: Fast, uncertainty-aware proximity queries with lazy search over local 3d data. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7631–7638. IEEE, 2018.

[7] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 6(59):eabg5810, 2021.

[8] Zhepei Wang, Xin Zhou, Chao Xu, and Fei Gao. Geometrically constrained trajectory optimization for multicopters. *arXiv preprint arXiv:2103.00190*, 2021.

[9] Pete Florence, John Carter, and Russ Tedrake. Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps. In *Proceedings of the Twelfth International Workshop on the Algorithmic Foundations of Robotics (WAFR 2016)*, 2016.

[10] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 2872–2879. IEEE, 2017.

[11] Ji Zhang, Chen Hu, Rushat Gupta Chadha, and Sanjiv Singh. Falco: Fast likelihood-based collision avoidance with extension to human-guided navigation. *Journal of Field Robotics*, 37(8):1300–1313, 2020.

[12] Fanze Kong, Wei Xu, Yixi Cai, and Fu Zhang. Avoiding dynamic small obstacles with onboard sensing and computation on aerial robots. *IEEE Robotics and Automation Letters*, 6(4):7869–7876, 2021.

[13] Xin Zhou, Zhepei Wang, Hongkai Ye, Chao Xu, and Fei Gao. EGO-Planner: An ESDF-free gradient-based local planner for quadrotors. *IEEE Robotics and Automation Letters*, 6(2):478–485, 2021.

[14] Jing Chen, Kunyue Su, and Shaojie Shen. Real-time safe trajectory generation for quadrotor flight in cluttered environments. In *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1678–1685. IEEE, 2015.

[15] Fei Gao, William Wu, Wenliang Gao, and Shaojie Shen. Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments. *Journal of Field Robotics*, 36(4):710–733, 2019.

[16] Jialin Ji, Zhepei Wang, Yingjian Wang, Chao Xu, and Fei Gao. Mapless-planner: A robust and fast planning framework for aggressive autonomous flight without map fusion. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6315–6321. IEEE, 2021.

[17] Zhepei Wang, Xin Zhou, Chao Xu, Jian Chu, and Fei Gao. Alternating minimization based trajectory generation for quadrotor aggressive flight. *IEEE Robotics and Automation Letters*, 5(3):4836–4843, 2020.

[18] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE international conference on robotics and automation*, pages 2520–2525. IEEE, 2011.

[19] Zhepei Wang, Chao Xu, and Fei Gao. Robust trajectory planning for spatial-temporal multi-drone coordination in large scenes. *arXiv preprint arXiv:2109.08403*, 2021.

[20] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[21] Boyu Zhou, Fei Gao, Jie Pan, and Shaojie Shen. Robust real-time UAV replanning using guided gradient-based optimization and topological paths. In *2020 IEEE International Conference on Robotics and Automation*, pages 1208–1214. IEEE, 2020.

[22] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[23] Yixi Cai, Wei Xu, and Fu Zhang. ikd-tree: An incremental kd tree for robotic applications. *arXiv preprint arXiv:2102.10808*, 2021.

[24] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. Fast-lio2: Fast direct lidar-inertial odometry. *arXiv preprint arXiv:2107.06829*, 2021.

[25] Guozheng Lu, Wei Xu, and Fu Zhang. Model predictive control for trajectory tracking on differentiable manifolds. *arXiv preprint arXiv:2106.15233*, 2021.