Teach-Repeat-Replan: A Complete and Robust System for Aggressive Flight in Complex Environments

Fei Gao, Luqi Wang, Boyu Zhou, Xin Zhou, Jie Pan, and Shaojie Shen

***Abstract*—In this article, we propose a complete and robust system for the aggressive flight of autonomous quadrotors. The proposed system is built upon on the classical teach-and-repeat framework, which is widely adopted in infrastructure inspection, aerial transportation, and search-and-rescue. For these applications, a human's intention is essential for deciding the topological structure of the flight trajectory of the drone. However, poor teaching trajectories and changing environments prevent a simple teach-and-repeat system from being applied flexibly and robustly. In this article, instead of commanding the drone to precisely follow a teaching trajectory, we propose a method to automatically convert a human-piloted trajectory, which can be arbitrarily jerky, to a topologically equivalent one. The generated trajectory is guaranteed to be smooth, safe, and dynamically feasible, with a human preferable aggressiveness. Also, to avoid unmapped or moving obstacles during flights, a fast local perception method and a sliding-windowed replanning method are integrated into our system, to generate safe and dynamically feasible local trajectories onboard. We name our system as *teach–repeat–replan*. It can capture users' intention of a flight mission, convert an arbitrarily jerky teaching path to a smooth repeating trajectory, and generate safe local replans to avoid unexpected collisions. The proposed planning system is integrated into a complete autonomous quadrotor with global and local perception and localization submodules. Our system is validated by performing aggressive flights in challenging indoor/outdoor environments. We release all components in our quadrotor system as open-source ros packages.**

***Index Terms*—Aerial systems: applications, motion, and path planning, autonomous vehicle navigation, collision avoidance.**

I. Introduction

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

There is a massive market for consumer drones nowadays. However, we observe that most of the operators of consumer drones are not professional pilots and would struggle in generating their ideal trajectory for a long time. In some scenarios, such as drone racing or aerial filming, it is impossible for a beginner-level pilot to control the drone to finish the race safely or take an aerial video smoothly without months of training. In a drone racing competition, each quadrotor is controlled by a human pilot to fly through several gates toward the terminal as quickly as possible. In the racing flight, collisions must be avoided to ensure safety, while the flight aggressiveness is expected to be extremely high. However, it is hard for a human pilot to master the skill of balancing speed and safety. As opposed to drone racing, aerial filming/videography does not prefer high speed, but good motion smoothness, because gentle transitions are typically good for generating aesthetical videos.

For industrial applications, there is also considerable demand in applying drones to repetitive inspections or search-and-rescue missions, where a human provides a preferable routine. In these situations, demonstrating a desirable trajectory and letting the drone to repeat it is a common requirement. However, the taught trajectory generated by an unskilled pilot is usually incredibly hard or dynamically infeasible to repeat, especially in some cluttered environments. Moreover, most of the vision-based teach-and-repeat applications [1]–[3], such as our previous work [1], are sensitive to changing environments. In [1], even if the environment changes very slightly, the global map has to be rebuilt, and the teaching process has to be redone.

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

Manuscript received June 30, 2019; revised January 18, 2020; accepted May 4, 2020. Date of publication May 27, 2020; date of current version October 1, 2020. This article was recommended for publication by Associate Editor K. Alexis and Editor P. Robuffo Giordano upon evaluation of the reviewers' comments. *(Luqi Wang and Boyu Zhou contributed equally to this work.)* This work was supported in part by the HKUST-DJI Joint Innovation Laboratory, and in part by the HKUST Institutional Fund. *(Corresponding author: Fei Gao.)*

Fei Gao and Xin Zhou are with the State Key Laboratory of Industrial Control and Technology, Zhejiang University, Hangzhou 310007, China (e-mail: fgaoaa@zju.edu.cn; xinzhou@zju.edu.cn).

Luqi Wang, Boyu Zhou, Jie Pan, and Shaojie Shen are with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong (e-mail: lwangax@ust.hk; bzhouai@ust.hk; jpanaa@ust.hk; eeshaojie@ust.hk).

This article has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors.

Color versions of one or more of the figures in this article are available online at https://ieeexplore.ieee.org.

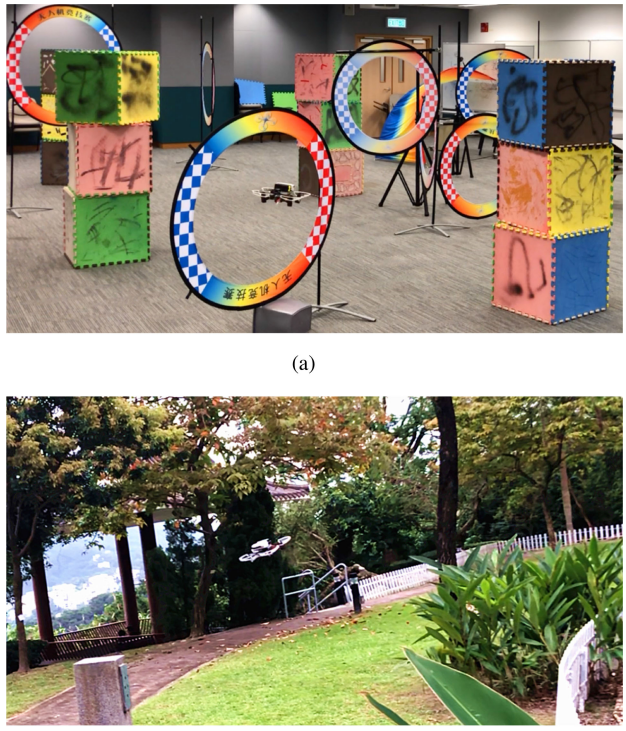
Digital Object Identifier 10.1109/TRO.2020.2993215

The aggressiveness of the generated repeating motions is tunable, which can meet speed requirements ranging from drone racing to aerial filming. Moreover, during the repeating flight, our system locally observes environmental changes and replans sliding-windowed safe trajectories to avoid unmapped or moving obstacles. In this way, our system can deal with changing environments. Our proposed system extends the classical robotics teach-and-repeat framework and is named as *teach–repeat–replan*. It is complete, flexible, and robust.

In our proposed system, the surrounding environment is reconstructed by onboard sensors. Then, the user's demonstrated trajectory is recorded by virtually controlling the drone in the map with a joystick or remote controller. Then, we find a flight corridor that preserves the topological structure of the teaching trajectory. The global planning is decoupled as spatial and temporal planning subproblems. Having the flight corridor, an energy-optimal spatial trajectory, which is guaranteed to be safe, and a time-optimal temporal trajectory, which is guaranteed to be physically feasible, are iteratively generated. In repeating, while the quadrotor is tracking the global spatial–temporal trajectory, a computationally efficient local map [4] is fused onboard by stereo cameras. Based on local observations, our proposed system uses a fast sliding-window replanning method [5] to avoid possible collisions. The replanning module utilizes gradient information to locally wrap the global trajectory to generate safe and dynamically feasible local plans against unmapped or moving obstacles.

The concept of generating optimal topology-equivalent trajectories for quadrotor teach-and-repeat was first proposed in our previous research [1]. In [1], once the repeating trajectory is generated, the drone executes it without any other considerations. In that work [1], the environment must remain intact during the repeating, and the localization of the drone is assumed to be perfect. These requirements are certainly not guaranteed in practice and, therefore, prevent the system from wider applications. Also, in [1], the flight corridor is represented by axis-aligned cubes. Cubes are easily found but inferior for grouping large free space in highly nonconvex surroundings. The flight corridor may, therefore, fail to cover a whole teaching trajectory or result in only a poor solution. In this article, to adapt to arbitrarily cluttered maps and provide sufficient solution space, we propose a method to generate general, free, large convex polyhedrons. To summarize, in this article, we extend the framework of the classical teach-and-repeat and propose several new contributions to make our system complete, robust, and flexible. We present a whole set of experiments, as shown in Fig. 1, and comparisons in various scenarios to validate our system. Detailed contributions are the following.

1) We advance our flight corridor generation method. Compared to our previous work [1], the flight corridor we use now provides much more optimization freedom, which facilitates the generation of more efficient and smooth global trajectories. Moreover, we propose methods to accelerate the corridor generation on CPU and GPU.
2) We integrate our previous works on online mapping [4] and replanning [5] into our system, to improve the robustness against errors of global maps, drifts of localization, and environmental changes and moving obstacles.
3) We release all components in the proposed system as open-source packages, named *Teach–Repeat–Replan*,[1] which include local/global planning, perception, localization, onboard controller, as well as a quadrotor simulator, for the reference of the community.



(a)

(b)

Fig. 1. Experiments in a challenging indoor drone racing site and an outdoor forest. A high-resolution video is available at: https://youtu.be/urEC2AAGEDs. (a) Snapshot of the indoor quadrotor flight. (b) Snapshot of the outdoor quadrotor flight.

In what follows, we discuss related literature works in Section II and introduce our system in Section III. Our methods for finding a flight corridor consisting of large convex polyhedrons and spatial–temporal trajectory optimization are detailed in Sections IV and V, respectively. The local replanning is introduced in Section VI. Experimental and benchmarked results are given in Section VII. Section VIII concludes this article.

II. Related Works

A. Robotic Teach-and-Repeat

Many robotics teach-and-repeat works, especially for mobile robots, have been published in recent years. Most of them focus on improving the accuracy or robustness in repeating/following the path given by operators, which is fundamentally different from our motivation. A lidar-based teach-and-repeat system is proposed in [6], where laser scans are used to localize the

[1] [Online]. Available: https://github.com/HKUST-Aerial-Robotics/Teach-Repeat-Replan

ground vehicle against its taught path driven by the user. Furgale *et al.* [7], [8] also develop a lidar-based ground robot, which is especially designed for repeating long-term motions in highly dynamic environments. This system equips a local motion planner that samples local trajectory to avoid dynamic elements during route following. A map maintenance module is used to identify moving objects and estimate their velocities. An iterative learning controller is proposed in [9] to reduce the tracking error during the repeating of the robot. This controller can compensate disturbances such as unmodeled terrains and environmental changes by learning a feedforward control policy. Vision-based teaching-and-repeat systems are also proposed in several works, such as the visual localization used by the rover in [3]. In this work, the authors build a manifold map during the teaching and then use it for localization in the repeating. In [10], a multiexperience localization algorithm is proposed to address the issue of environmental changes. The ground robot is localized robustly against several past experiences. In [11] and [12], to further improve the accuracy and robustness in localization, illumination and terrain appearances are considered in their proposed visual navigation system used for teach-and-repeat.

Compared to ground teach-and-repeat works, research on aerial teach-and-repeat is few. In [2], a vision-based drone is used to inspect infrastructure repetitively. In the teaching phase, the desired trajectory is demonstrated by the operator, and some keyframes in the visual simultaneous localization and mapping are recorded as checkpoints. While repeating, local trajectories are generated to connect those checkpoints by using the minimum-snap polynomials [13]. To function properly, in [2], the teaching trajectory itself must be smooth, and the environment must have no changes during the whole repeating process. In contrast, our proposed system can convert an arbitrarily poor path to a safe and efficient trajectory with expected flying aggressiveness. Moreover, our system is flexible. Since it records the teaching path by virtually controlling the drone in simulation, a manually piloted teaching process is not necessary. Finally, our proposed system is robust to environmental changes and can even avoid moving obstacles.

B. Quadrotor Trajectory Planning

Trajectory optimization is essential in generating safe and executable repeating trajectories from poor teaching. The minimum-snap trajectory optimization is proposed by Mellinger and Kumar [13]. In [13], piecewise polynomials are used to represent the quadrotor trajectory and are optimized by quadratic programs (QPs). A method for obtaining a closed-form solution of the minimum-snap program is proposed in [14]. In this work, a safe geometric path is first found to guide the generation of the trajectory. By adding intermediate waypoints to the path iteratively, a safe trajectory is finally generated after solving the minimum-snap problem several times. Our previous works [15]–[17] carve a flight corridor consisting of simple convex shapes (sphere and cube) in a complex environment. The flight corridor constructed by a series of axis-aligned cubes or spheres can be extracted very fast on an occupancy map or a Kd-tree. Then, we use the flight corridor and physical limits to constrain a piecewise Bézier curve, to generate a guaranteed safe and dynamically feasible trajectory. Other works are proposed to find general convex polyhedrons for constraining the trajectory. In [18], a piecewise linear path is used to guide and initialize the polyhedron generation. In [19], by assuming all obstacles are convex, semidefinite optimization and QPs are iteratively solved to find the maximum polyhedron seeded at a random coordinate in 3-D space. Gradient information in maps is also valuable for local trajectory optimization. In CHOMP [20], the trajectory optimization problem is formulated as a nonlinear optimization over the penalty of safety and smoothness. In [21] and [22], gradient-based methods are combined with piecewise polynomials for local planning of quadrotors. In this article, we also utilize gradient-based optimization for local replanning.

Time optimization or the so-called optimal time parameterization is used to optimize the time profile of a trajectory, given the physical limits of a robot. Methods can be divided as direct methods [23] and indirect methods [24]. Direct methods generate an optimal spatial–temporal trajectory directly in the configuration space. For indirect methods, a trajectory independent of time is first generated; then, the relationship between the time and the trajectory is optimized by an additional optimization process. The method in [25] finds a mapping function between the time and the trajectory, by recursively adding key points into the function, and squeezes out the infeasibility of the time profile. This method obtains an optimal local solution and is computationally expensive. Roberts and Hanrahan [24] also propose a mapping function, which maps time to a virtual parameterization of the trajectory. This mapping function is then optimized under a complicated nonlinear formulation. However, its global optimality is not guaranteed, and a feasible initial solution is required to bootstrap the optimization. Convex optimization [26] and numerical integration [27] are two typical methods for robotics time-optimal path parameterization problem. Although numerical integration [27], [28] has shown superior performance in computing efficiency, convex optimization [26] has the advantage of adding regularization terms other than the total time into its objective function. This specialty suits well for our application, where the user defines an expected aggressiveness of the drone, as sometimes the drone may not be expected to fly as fast as possible. As for efficiency, since we do temporal optimization offline before the repeating, computing time is not critical.

III. System Overview

A. System Architecture

Our hardware architecture is given in Fig. 2. The quadrotor is equipped with an Intel RealSense[2] stereo camera pair for imagery and depth sensing, a DJI N3[3] autopilot for stabilizing the drone and feeding inertial measurement unit data, a DJI manifold 2-C[4] for onboard computing, and a DJI Lightbridge2[5] for manual piloting and remote monitoring. A more detailed

[2][Online]. Available: https://www.intelrealsense.com/depth-camera-d435/
[3][Online]. Available: https://www.dji.com/n3
[4][Online]. Available: https://www.dji.com//manifold-2
[5][Online]. Available: https://www.dji.com/lightbridge-2

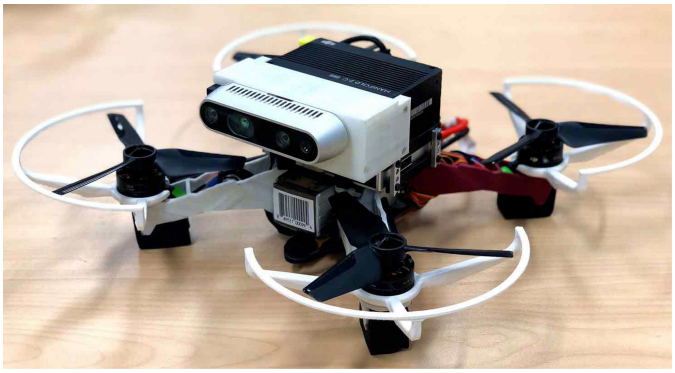


Fig. 2. Hardware setting of our autonomous drone system.

hardware specification, including the selection of the drone frame and the dynamic system, and the design of customized components can be found in the project webpage.[6]

The overall software architecture of our quadrotor system is shown in Fig. 3. Global mapping, flight corridor generation, and global spatial–temporal planning are done on an offboard computer. Other processing runs onboard the drone during flights. Before teaching, a global map is built by sensors. During teaching, a flight corridor is generated by inflating the teaching trajectory. Then, the spatial and temporal trajectories are optimized iteratively within the flight corridor under a coordinate descent scheme [29]. A local planner runs onboard to avoid unexpected obstacles observed in repeating flights. We use a geometric controller [30] for trajectory tracking.

B. Globally Consistent Localization and Mapping

We use VINS [31], a robust visual–inertial odometry (VIO) framework, to localize the drone. Moreover, loop closure detection and global pose graph optimization are used in our system, to globally correct the pose estimation. The global mapping is done by fusing depth measurements from the stereo cameras with the pose estimation. By using our previous research on the deformable map [32], our global mapping module maintains a series of submaps with each attached to a keyframe of the pose graph. In this way, the map is attached to the pose graph and is, therefore, globally driftless. During the mapping, when a loop closure is detected, keyframes in the global pose graph are corrected, and all submaps are deformed accordingly. The global pose graph optimization is also activated during the repeating. When loop closure is detected, the pose of the drone is corrected accordingly to eliminate the drift.

C. Global Spatial–Temporal Planning

Given an extremely poor teaching trajectory, both the geometric shape and the time profile of it are far from optimal. They are, therefore, useless or even harmful for conducting optimization. However, the topological information of the teaching trajectory is essential, since it reflects the human's intention. To preserve the topological information, we group the free space around the teaching trajectory to form a flight corridor (see Section IV). The corridor shares the same topological structure as the teaching trajectory and provides large freedom for trajectory optimization. It is hard to concurrently optimize a trajectory spatially and temporally in the flight corridor. However, generating a safe spatial trajectory given a fixed time allocation (see Section V-A) and optimizing the time profile of a fixed spatial trajectory (see Section V-B) are both conquerable. Therefore, we iteratively optimize the trajectory in the space–time joint solution space by designing a coordinate descent [29] framework. An objective with weighted energy and time duration is defined for optimization. We first generate a spatial trajectory whose energy is minimized, and then, we use the temporal optimization to obtain the optimal time profile of it. The optimal time profile is used to parameterize the trajectory again for next spatial optimization. The spatial–temporal optimizations are done iteratively until the total cost cannot be reduced any more.

[6][Online]. Available: https://github.com/HKUST-Aerial-Robotics/Teach-Repeat-Replan/tree/experiment

D. Local Collision Avoidance

In practice, the accumulated drift of VIO is unavoidable, and the recall rate of loop closure is unstable. Although we build a dense global map, when the drift is significant and not corrected by loop detection in time, the quadrotor may have collisions with obstacles. Moreover, the environment may change or contain moving obstacles. Our previous work [1] has to rebuild the map when changes happen and cannot deal with dynamic obstacles. To resolve the above issues, we integrate our previous local map fusion module [4] into our system to detect collisions locally and serve the local trajectory optimization. Moreover, we propose a sliding-window local replanning method based on our previous research on quadrotor local planning [5] to avoid collisions on the flight.

In the repeating phase, the drone controls its yaw angle to face its flying direction and builds a local map by stereo cameras. We consistently check the local trajectory within a replanning time horizon. If any collision along the local trajectory is reported, replanning is triggered to wrap the trajectory away from obstacles by gradient-based optimization [5].

IV. Flight Corridor Generation

As stated in Section III-C, the first step of our global planning is to build a flight corridor around the teaching trajectory for spatial–temporal trajectory optimization. In our previous work [1], the flight corridor is constructed by finding a series of axis-aligned cubes, which may sacrifice much space, especially in a highly nonconvex environment, as shown in Fig. 4. A more illustrative comparison is shown in Fig. 5, where the convex polyhedron captures much more free space than the simple cube. Using simple axis-aligned cubes significantly limits the solution space of trajectory optimization, which may result in a poor solution. What is more, in situations where the free space is very limited, such as flying through a very narrow circle, a cube-based corridor [1] may even fail to cover all teaching trajectory and result in no solutions existing in the corridor. Therefore, to utilize the free space more sufficiently and adapt to even extremely

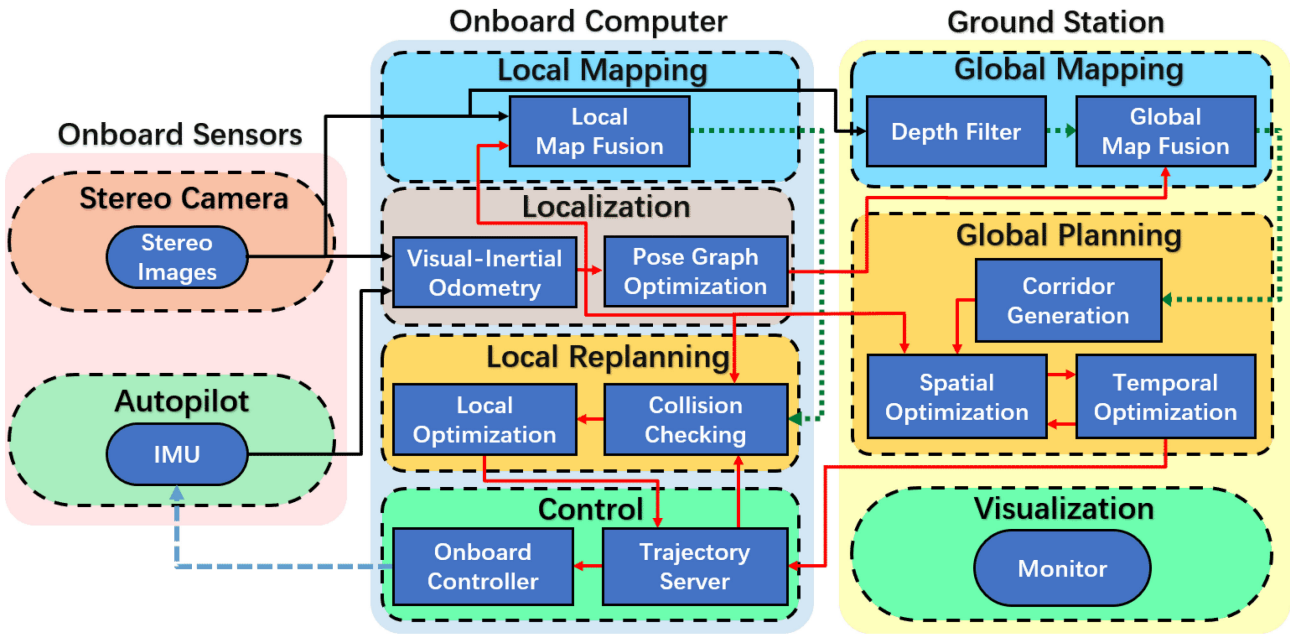


Fig. 3. Software architecture of our quadrotor system. Global mapping, planning, and visualization are running on a ground station, while state estimation, local sensing, and replanning are running onboard.

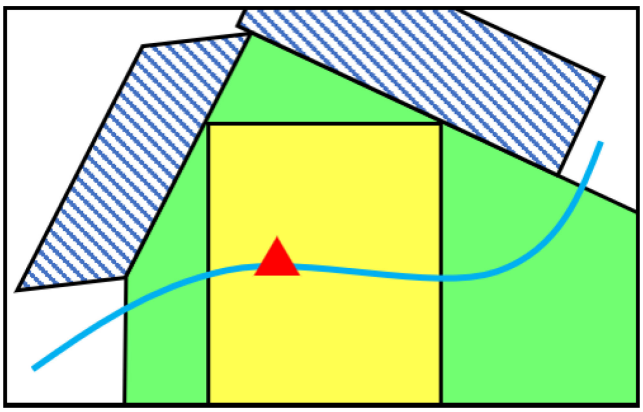


Fig. 4. Illustration of free space captured by an axis-aligned cube and a general polyhedron. Obstacles are shown in dashed lines. The blue curve is the teaching trajectory of humans. The red triangle is the seed for finding local free space. The axis-aligned cube and a corresponding general convex polyhedron are shown in yellow and green, respectively.

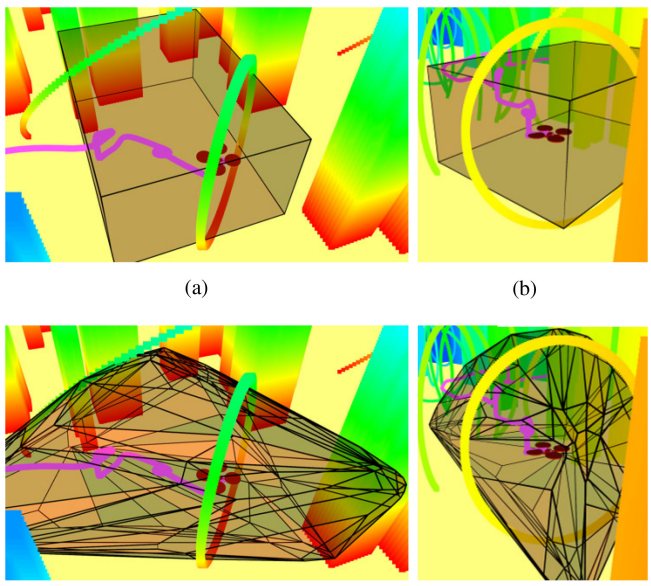


Fig. 5. Comparison of an axis-aligned cube and a general convex polyhedron. The cube and the polyhedron are generated to their largest volume, from the same seed coordinate. The way to inflate the cube is stated in our previous paper [1]. The method to find the general free polyhedron will be detailed later in Section IV-A. (a) Side view and (b) front view of the axis-aligned cube. (c) Side view and (d) front view of the general convex polyhedron.

cluttered maps, we propose a method to generate general, free, and large convex polyhedrons.

Since the human's teaching trajectory may be arbitrarily jerky, we cannot assume that there is a piecewise linear path to initiate the polyhedron generation, as in [18]. We also make no requirements on the convexity of obstacles in the map as in [19]. Our method is based on convex set clustering, which is similar to [33], but is different and advanced at the following.

1) We make no assumption of the growing directions of convex clusters and generate completely collision-free polyhedrons based on our dense occupancy map.
2) We introduce several careful engineering considerations, which significantly speed up the clustering.
3) We fully utilize the parallel structure of this algorithm and accelerate it over an order of magnitude in GPUs.
4) We introduce a complete pipeline from building the convex polyhedron clusters to establishing constraints in trajectory optimization.

A. Convex Cluster Inflation

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

Definition: A set $\mathcal{S}$ in a vector space over $\mathcal{R}$ is called a convex set if the line segment joining any pair of points of $\mathcal{S}$ lies entirely in $\mathcal{S}$. [34].

The pipeline for iteratively inflating such a cluster while preserving convexity is stated in Algorithm 1. Our method operates on a 3-D occupancy map $\mathcal{M}$, where voxels are labeled as *obstacle* or *free*. Three voxel sets are maintained in the algorithm. $\mathcal{C}$ stands for the targeting convex voxel cluster. $\mathcal{C}^+$ is the set of voxels that are tentative to be added to $\mathcal{C}$ in this iteration. $\mathcal{C}^*$ contains newly added voxels, which preserve the convexity. The cluster inflation starts by adding the seed voxel p to $\mathcal{C}$ and adding all neighboring voxels of p to $\mathcal{C}^+$. In an iteration, each voxel p^+ in $\mathcal{C}^+$ is checked whether it can preserve convexity using the function CHECK_CONVEXITY($p^+, \mathcal{C}, \mathcal{M}$). This function, as shown in Algorithm 2, casts rays from p^+ to each existing voxel in $\mathcal{C}$. According to the definition of the convex set, $\mathcal{M}$ with p^+ is convex if and only if all of its rays are collision free. Based on this criterion, qualified voxels are considered as *active* voxels and are added into $\mathcal{C}$ and $\mathcal{C}^*$. Neighboring voxels of all *active* voxels p^* are traversed and collected by the function GET_NEIGHBORS($\mathcal{C}^*$) for the next iteration. The inflation ends when $\mathcal{C}^+$ becomes empty, which implies that no additional voxels can be added into $\mathcal{C}$. Fig. 6 also illustrates the procedure of the *convex cluster inflation*.

Having a *convex cluster* of voxels, we convert it to the algebraic representation of a polyhedron for the following spatial trajectory optimization. The quickhull algorithm [35] is adopted here for quickly finding the convex hull of all clustered voxels. The convex hull lies in vertex representation (V-representation) $\{V_0, V_1, \ldots, V_m\}$ and is then converted to its equivalent hyperplane representation (H-representation) by using the double description method [36]. The H-representation of a 3-D polyhedron is a set of affine functions

$$a_i^x \cdot \mathbf{x} + a_i^y \cdot \mathbf{y} + a_i^z \cdot \mathbf{z} \leq k_i, \quad i = 1, \ldots, N. \tag{1}$$

Here, $\{a_i^x, a_i^y, a_i^z\}$ is the normal vector of a 3-D hyperplane, k_i is a constant, and N is the number of hyperplanes.

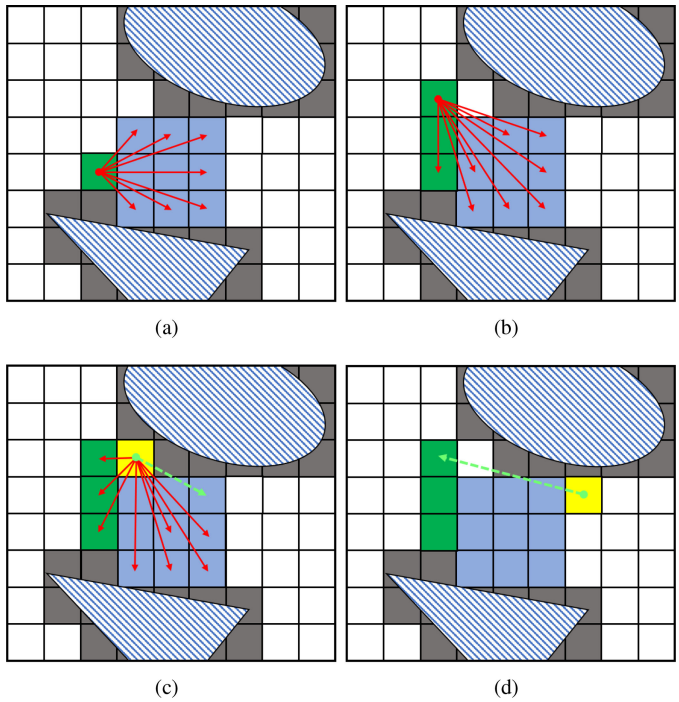


Fig. 6. Illustration of the *convex cluster inflation*. In (a) and (b), all qualified neighbor voxels are added to the *convex cluster*. In (c) and (d), since an occupied voxel occludes a ray (the green arrow) to one of the clustered voxels, the testing voxel (in yellow) is excluded to the convex cluster.

Algorithm 1: Convex Cluster Inflation.

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

B. CPU Acceleration

As shown in Algorithm 1, to determine whether a voxel preserves the convexity, we need to conduct ray casting to all existing voxels in the *convex cluster*. Iterating with all voxels and rays makes this algorithm impossible to run in real time, especially when the occupancy grid map has a fine resolution. To generate the polyhedron in real time, we take careful engineering considerations on the implementations and propose some critical techniques that increase the overall efficiency significantly.

1) Polyhedron Initialization: We initialize each convex cluster as an axis-aligned cube using our previous method [15], which can be done very fast, since only index query ($\mathcal{O}(1)$)

Algorithm 2: Convexity Checking.

Notation:
Ray Cast: l, Candidate voxel: p^+
Input: p^+, $\mathcal{C}$, $\mathcal{M}$, **Output**: True / False
function CHECK_CONVEXITY($p^+, \mathcal{C}, \mathcal{M}$)
  for each $p \in \mathcal{C}$ **do**
    $l \leftarrow$ CAST_RAY(p^+, p)
    if PASS_OBS($l, \mathcal{M}$) **then**
      returnFalse
    end if
  end for
  returnTrue
end Function

operations are needed. After inflating the cube to its maximum volume, as in Fig. 4, we switch to the convex clustering to further group convex free space around the cube.

The proposed polyhedron initialization may result in a final polyhedron different from the one which is clustered from scratch. This is because an axis-aligned cube only inflates in x, y, z directions, while a *convex cluster* grows in all possible directions (26 connections in a 3-D grid map). However, this initialization process is reasonable. Our purpose is not to make each polyhedron optimal but to capture as much free space as possible that a simple cube cannot. In practice, the initialization process provides a fast discovery of nearby space, which is easily to group, and does not prevent the following *convex cluster inflation* to refine the polyhedron. In Section VII-C1, we show that the initialization process significantly improves the computing efficiency with only a neglectable sacrifice on the volume of the final polyhedron.

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

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

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

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

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

Proof: Obviously, the *convex cluster* is a closed set with *outer* voxels at its boundary. The candidate voxel is outside this set. Therefore, casting a ray from any *inner* voxel to the candidate voxel must pass one of the *outer* voxels. According to *Theorem 1*, checking convexity of this ray can be terminated after the ray passes an *outer* voxel. This means for a candidate voxel, checking rays cast to *outer* voxels is sufficient. ■

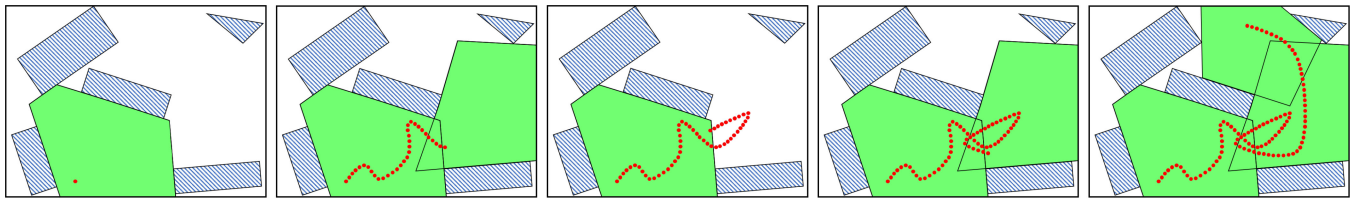
Algorithm 3: Parallel Convex Cluster Inflation.

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

By introducing the above techniques, the proposed *convex cluster inflation* can work in real time for a mediate grid resolution (0.2 m) on a general portable CPU. The efficacy of these techniques is numerically validated in Section VII-C1.

C. GPU Acceleration

We propose a parallel computing scheme that significantly speeds up the inflation by one order of magnitude if a GPU is available. As shown in Section IV-A, when the *convex cluster* discovers a new neighboring voxel, sequentially traversing and checking all rays is naturally parallelizable. With the help of a multicore GPU, we can cast rays and check collisions parallelly. Moreover, to fully utilize the massively parallel capability of a GPU, reduce the serialize operations, and minimize the amount of data transferring between CPU and GPU, we examine all potential voxels of the cluster parallelly in one iteration. Instead of discovering a new voxel and checking its rays, we find all neighbors of the active set $\mathcal{C}^*$ and check their rays all in parallel. The detailed procedure is presented in Algorithm 3, where GET_NEIGHBORS($\mathcal{C}$) collects all neighbors of a set of voxels, and PARA_CHECK_CONVEXITY($\mathcal{C}^+, \mathcal{C}, \mathcal{M}$) checks the convexity of all candidate voxels parallelly in the GPU. Note that in the serialized version of the proposed method, the voxel discovered earlier may prevent later ones from being clustered, as illustrated in Fig. 6. However, in the parallel clustering, voxels examined at the same time may add conflicting voxels to the cluster. Therefore, we introduce an additional variable r to record sequential information of voxels. As shown in Algorithm 4, the kernel function runs on the GPU per block. It checks the ray cast from every candidate voxel in $\mathcal{C}^+$ to a cluster voxel in $\mathcal{C}$ and to each other candidate voxel that has a prior index. After that, the function CHECK_RESULTS(r) selects all qualified voxels and adds them into $\mathcal{C}$. First, candidate voxels that have collisions with $\mathcal{C}$ are directly excluded. After that, candidate voxels having collisions with other candidates that have already been added into $\mathcal{C}$ are excluded. In this way, we finally get the same result



(a) (b) (c) (d) (e)

Fig. 7. (a)–(e) Flight corridor generation process. Red dots are coordinates along the teaching trajectory. In (b), a new convex polyhedron is generated and added to the flight corridor when the drone leaves the corridor enters undiscovered space. In (c), the drone leaves the last polyhedron and returns back to the second to last one, so the last polyhedron is deleted from the corridor.

Algorithm 4: Parallel Convexity Checking.

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

Input: $r, \mathcal{C}^+$, **Output**: $\mathcal{C}^*$
function CHECK_RESULTS$r, \mathcal{C}^+$
  for each $p_i^+ \in \mathcal{C}^+$ **do**
    if $r[i].status == True$ **then**
      $\mathcal{C}^* \leftarrow \mathcal{C}^* \cup p^+$
    else if $r[i].status == Pending$ **then**
      for each $j \in r[i].list$ **do**
        if $p_j^+ \in \mathcal{C}^*$ **then**
          go to 28
        end if
      end for
      $\mathcal{C}^* \leftarrow \mathcal{C}^* \cup p^+$
    end if**end for****return**$\mathcal{C}^*$**end Function**
  end for
  return
end function

Algorithm 5: Flight Corridor Generation.

Notation: Flight Corridor $\mathcal{G}$, Drone Position p, Convex Polyhedron $\mathcal{P}$
$\mathcal{P} \leftarrow$ CONVEX_INFLATION($p, \mathcal{M}$)
$\mathcal{G}$.push_back($\mathcal{P}$)
while Teaching **do**
  $p \leftarrow$ UPDATE_POSE()
  if OUTSIDE(p, $\mathcal{G}$[-1]) **then**
    if INSIDE(p, $\mathcal{G}$[-2]) **then**
      $\mathcal{G}$.pop_back()
    else
      $\mathcal{P} =$ CONVEX_INFLATION($p, \mathcal{M}$)
      $\mathcal{G}$.push_back($\mathcal{P}$)
    end if
  end if
end while
return $\mathcal{G}$

as in the serialized version of the clustering. The efficacy of the parallel computing scheme is shown in Section VII-C1.

D. Corridor Generation and Loop Elimination

Since the trajectory provided by a user may be arbitrarily jerky and contain local loops, we introduce an especially designed mechanism to eliminate unnecessary loops, i.e., repeatable polyhedrons. The exclusion of repeatable polyhedrons is essential. In the following trajectory optimization (see Section V), each polyhedron is assigned with one piece of the trajectory. Repeatable polyhedrons would result in an optimized trajectory loop as the user does, which is obviously not efficient. The pipeline of the corridor generation is shown in Algorithm 5 and Fig. 7. At the beginning of the teaching, the flight corridor is initialized by finding the maximum polyhedron around the position of the drone. Then, as the human pilots the drone to move, we keep checking the drone's position. If it goes outside the last polyhedron ($\mathcal{G}[-1]$), we further check whether the drone discovers new free space or not. If the drone is contained within the second last polyhedron ($\mathcal{G}[-2]$), we can determine that the teaching trajectory has a loop, as shown in Fig. 7(c). Then, the last polyhedron in the corridor is regarded as repeatable and is, therefore, popped out from the corridor. Otherwise, as shown in

Fig. 7(d), the drone is piloted to discover new space. Then, a new polyhedron $\mathcal{P}$ is inflated and added to the tail of the corridor. The corridor generation is terminated when the teaching finishes. Since no obstacles are included in the corridor, the final flight corridor shares the same topological structure as the teaching trajectory. Moreover, the corridor has no unnecessary loops.

In some situations where the drone is piloted to collide with obstacles, no matter by intention or occasion, the teaching trajectory and the flight corridor would be broken. To utilize such a rough path to complete the teaching, we locally find the shortest path to bypass the obstacle that blocks the teaching trajectory. Therefore, the corridor generation method always finds a solution.

V. Spatial–Temporal Global Trajectory Optimization

Inside the flight corridor found in Section IV, our proposed system generates a global flight trajectory. Since the teaching trajectory may be arbitrarily jerky, its time profile cannot be used by the repeating trajectory. We, thus, cannot obtain an efficient space–time trajectory by merely smoothing the path spatially without temporal optimization. Instead, we solve the global trajectory generation problem by spatial and temporal alternative optimization.

A. Spatial Trajectory Optimization

For the spatial optimization, we use the Bernstein basis to represent the trajectory as a piecewise Bézier curve, since it can be easily constrained in the flight corridor by enforcing constraints on control points. An ith-order Bernstein basis is

$$b_n^i(t) = \binom{n}{i} \cdot t^i \cdot (1-t)^{n-i} \tag{2}$$

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

$$f_\mu(t) = \begin{cases} \sum_{i=0}^{n} c_{\mu,1}^i b_n^i(t/T_1), & t \in [0, T_1] \\ \sum_{i=0}^{n} c_{\mu,2}^i b_n^i(t/T_2), & t \in [0, T_2] \\ \quad\vdots & \quad\vdots \\ \sum_{i=0}^{n} c_{\mu,N}^i b_n^i(t/T_N), & t \in [0, T_N] \end{cases}. \tag{3}$$

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

We minimize the third-order derivative (jerk) of the Bézier curve, following the minimum-snap formulation [13]. The jerk of a curve corresponds to the angular velocity; the minimization of jerks alleviates the rotation and, therefore, facilitates visual tracking. The objective of the piecewise curve is

$$J = \sum_{\mu}^{x,y,z} \sum_{m=1}^{N} \int_0^{T_m} \left(\frac{d^3 f_{\mu,m}(t)}{dt^3} \right)^2 dt \tag{4}$$

which is in a quadratic form denoted as $\mathbf{c}^T \mathbf{Q} \mathbf{c}$. Here, $\mathbf{c}$ is composed of all control points in *x, y, z* dimensions. $\mathbf{Q}$ is a semidefinite Hessian matrix.

For a Bézier curve, its higher order derivatives can be represented by linear combinations of corresponding lower order control points. For the first- and second-order derivatives of the mth piece of the curve in (3), we have

$$\begin{aligned} f'_m(t) &= \sum_{i=0}^{n-1} n(c_m^{i+1} - c_m^i) b_{n-1}^i \left(\frac{t}{T_m} \right) \\ f''_m(t) &= \sum_{i=0}^{n-2} n(n-1)(c_m^{i+2} - 2c_m^{i+1} + c_m^i) b_{n-2}^i \left(\frac{t}{T_m} \right). \end{aligned} \tag{5}$$

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

$$\begin{aligned} & c_0^0 = p^0 \\ & c_N^n = p^f \\ & n(c_0^1 - c_0^0) = v^0 \\ & n(c_N^n - c_N^{n-1}) = v^f \\ & n(n-1)(c_0^2 - 2c_0^1 + c_0^0) = a^0 \\ & n(n-1)(c_N^n - 2c_N^{n-1} + c_N^{n-2}) = a^f. \end{aligned} \tag{6}$$

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

$$\begin{aligned} & c_j^n = c_{j+1}^0 \\ & (c_j^n - c_j^{n-1})/T_j = (c_{j+1}^1 - c_{j+1}^0)/T_{j+1} \\ & (c_j^n - 2c_j^{n-1} + c_j^{n-2})/T_j^2 = (c_{j+1}^2 - 2c_{j+1}^1 + c_{j+1}^0)/T_{j+1}^2. \end{aligned} \tag{7}$$

3) Safety Constraints: By enforcing each piece of the curve inside the corresponding polyhedron, the safety of the trajectory is guaranteed. Thanks to the convex hull property, an entire Bézier curve is confined within the convex hull formed by all its control points. Therefore, we constrain control points using hyperplane functions obtained in (1). For the ith control point $c_{j,x}^i, c_{j,y}^i, c_{j,z}^i$ of the jth piece of the trajectory in x, y, z dimensions, constraints are

$$\begin{aligned} & a_0^x \cdot c_{j,x}^i + a_0^y \cdot c_{j,y}^i + a_0^z \cdot c_{j,z}^i \le k_0 \\ & a_1^x \cdot c_{j,x}^i + a_1^y \cdot c_{j,y}^i + a_1^z \cdot c_{j,z}^i \le k_1 \\ & \quad\vdots \\ & a_n^x \cdot c_{j,x}^i + a_n^y \cdot c_{j,y}^i + a_n^z \cdot c_{j,z}^i \le k_n. \end{aligned} \tag{8}$$

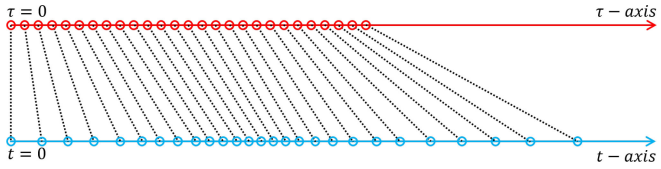


Fig. 8. Effect of the temporal optimization. t and τ are the time profiles of the spatial trajectory before and after optimization.

Constraints in (6) and (7) are affine equality constraints ($\mathbf{A}_{\text{eq}}\mathbf{c} = \mathbf{b}_{\text{eq}}$) and (8) is in affine inequality formulation ($\mathbf{A}_{\text{ie}}\mathbf{c} \leq \mathbf{b}_{\text{ie}}$). Finally, the spatial trajectory optimization problem is formulated as a QP as follows:

$$\begin{aligned} \min \quad & \mathbf{c}^T\mathbf{Q}\mathbf{c} \\ \text{s.t.} \quad & \mathbf{A}_{\text{eq}}\mathbf{c} = \mathbf{b}_{\text{eq}} \\ & \mathbf{A}_{\text{ie}}\mathbf{c} \leq \mathbf{b}_{\text{ie}}. \end{aligned} \tag{9}$$

Unlike our previous works on a corridor-constrained trajectory [15], [17], herein, the kinodynamic feasibility (velocity and acceleration) is not guaranteed by adding higher order constraints into this optimization program, but by temporal optimization (see Section V-B). For a rest-to-rest trajectory, (9) always has a mathematically feasible solution.

Note that, in our system, all intentional teaching loops will be removed after the global planning, since each piece of the repeating trajectory is assigned to a polygon and repeating polygons are all canceled out (see Section IV-D). This may violate the human's original intention if a loop is necessary for some inspection reasons. To add repeating loops, one can directly add extra intermediate waypoint constraints into (9). Enforcing a trajectory to pass these waypoints is trivial, as shown in early works such as [13] and [14]. It is equivalent as adding extra linear boundary constraints, as in (6), with only position terms. Since waypoints only pose affine constraints, the overall optimization program [see (9)] retains convex.

B. Temporal Trajectory Optimization

In spatial optimization, a corridor-constrained spatial trajectory is generated given a fixed time allocation. To optimize the trajectory temporally, we design a retiming function $\{t(\tau) : \tau \to t\}$ to map the original time variable t to a variable τ. The relation between τ and t is shown in Fig. 8. In this article, the retiming function $t(\tau)$ is named as the temporal trajectory, and finding the optimal $t(\tau)$ is called the temporal optimization. For the N-piece spatial curve defined in (3), we write $t(\tau)$ as a corresponding N-piece formulation:

$$t(\tau) = \begin{cases} t_1(\tau), & t_1(0) = 0, t_1(\mathcal{T}_1^*) = T_1, t_1 \in [0, T_1] \\ t_2(\tau), & t_2(0) = 0, t_2(\mathcal{T}_2^*) = T_2, t_2 \in [0, T_2] \\ \vdots & \vdots \\ t_N(\tau), & t_N(0) = 0, t_N(\mathcal{T}_N^*) = T_N, t_N \in [0, T_N] \end{cases} \tag{10}$$

where $T_1, T_2, \ldots T_N$ are original time durations of the spatial curve $f_\mu(t)$, and $\mathcal{T}_1^*, \mathcal{T}_2^*, \ldots \mathcal{T}_N^*$ are time durations after temporal optimization. Since physically time only increases, $t(\tau)$ is a monotonically increasing function. Therefore, we have $\dot{t}(\tau) \geq 0$. For clarity, in what follows, we use $c' = dc/dt$ to denote taking derivatives with respect to t, and $\dot{c} = dc/d\tau$ for taking derivatives with respect to τ. By substituting t with $t(\tau)$ in $f_\mu(t)$ and taking derivatives with chain rule, we can write the velocity as

$$\dot{f}(t(\tau)) = f'(t) \cdot \dot{t}$$

and acceleration as

$$\ddot{f}(t(\tau)) = f'(t) \cdot \ddot{t} + f''(t) \cdot \dot{t}^2. \tag{11}$$

The velocity and acceleration are also piecewise functions.

C. Minimum-Time Formulation

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

$$\mathcal{T} = \int_0^{\mathcal{T}} 1 d\tau = \sum_{m=1}^{N} \int_0^{T_m} \frac{1}{\dot{t}_m} dt \tag{12}$$

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

$$\mathcal{J} = \sum_{m=1}^{N} \int_1^{T_m} \left(\frac{1}{\dot{t}_m} + \rho \cdot \ddot{t}_m^{\,2} \right) dt \tag{13}$$

where ρ is a weight of the aggressiveness. By setting a larger ρ, we can obtain more gentle motions in the temporal trajectory. If $\rho = 0$, the temporal optimization is solved for generating motions as fast as possible. The motions generated with a large ρ can be viewed in our previous work [1].

Following the direct transcription method in [26], $\alpha(t)$ and $\beta(t)$ are introduced as two additional piecewise functions

$$\alpha_m(t) = \ddot{t}_m, \quad \beta_m(t) = \dot{t}_m^2, \quad m = 1, 2, \ldots, N. \tag{14}$$

According to the relationship between $\ddot{t}_m$ and $\dot{t}$, we can have

$$\beta_m(t) \geq 0, \quad \beta_m'(t) = 2 \cdot \alpha_m(t). \tag{15}$$

Then, the objective function in (13) is reformulated as

$$\mathcal{J} = \sum_{m=1}^{N} \int_0^{T_m} \left(\frac{1}{\sqrt{\beta_m(t)}} + \rho \cdot \alpha_m(t)^2 \right) dt, \tag{16}$$

2) Constraints: The continuities of $t(\tau)$ are enforced by setting constraints between every two consecutive pieces of it. In each dimension $\mu \in x, y, z$, we have

$$f'_{\mu,m}(T_m) \cdot \sqrt{\beta_m(T_m)} = f'_{\mu,m+1}(0) \cdot \sqrt{\beta_{m+1}(0)} \tag{17}$$

$$\begin{aligned} & f'_{\mu,m}(T_m) \cdot \alpha_m(T_m) + f''_{\mu,m}(T_m) \cdot \beta_m(T_m) \\ & = f'_{\mu,m+1}(0) \cdot \alpha_{m+1}(0) + f''_{\mu,m+1}(0) \cdot \beta_{m+1}(0). \end{aligned} \tag{18}$$

Then, to satisfy the initial and the final velocity and acceleration a_0, v_0, a_f, v_f, we set boundary constraints

$$f'_{\mu,1}(0) \cdot \sqrt{\beta_1(0)} = v_0 \tag{19}$$

$$f'_{\mu,N}(T_N)\cdot\sqrt{\beta_N(T_N)} = v_f \quad (20)$$

$$f'_{\mu,1}(0)\cdot\alpha_1(0)+f''_{\mu,1}(0)\cdot\beta_1(0) = a_0 \quad (21)$$

$$f'_{\mu,N}(T_N)\cdot\alpha_N(T_N)+f''_{\mu,N}(T_N)\cdot\beta_N(T_N) = a_f. \quad (22)$$

Finally, kinodynamic feasibility constraints are set as

$$-v_{\max}\le f'_{\mu,m}(t)\cdot\sqrt{\beta_m(t)}\le v_{\max} \quad (23)$$

$$-a_{\max}\le f'_{\mu,m}(t)\cdot\alpha_m(t)+f''_{\mu,m}(t)\cdot\beta_m(t)\le a_{\max} \quad (24)$$

where $v_{\max}$ and $a_{\max}$ are the physical limits of the drone.

3) Second-Order Cone Program (SOCP) Reformulation: The above optimization problem has convex objective and constraints and is, therefore, a convex program. To make it easily solvable, for each piece of the trajectory, $t_m\in[0,T_m]$ is discretized to $t_m^0,t_m^1,\ldots t_m^{K_m}$ according to a given resolution δt. $K_m=\lceil T_m/\delta t\rceil+1$. Then, $\alpha_m(t)$ is considered as piecewise constant at each discretization point. According to (15), $\beta_m(t)$ is piecewise linear. In this way, $\alpha_m(t)$ and $\beta_m(t)$ are modeled by a series of discrete variables α_m^k and β_m^k, where β_m^k is evaluated at t_m^k and α_m^k is evaluated at $(t_m^k+t_m^{k+1})/2$.

By applying the above discretization, the objective in (16) is derived as

$$\mathcal{J}=\sum_{m=1}^{N}\sum_{k=0}^{K_i-1}\left(\frac{2}{\sqrt{\beta_m^{k+1}}+\sqrt{\beta_m^k}}+\rho\cdot(\alpha_m^k)^2\right)\cdot\delta t \quad (25)$$

which is mathematically equivalent to the affine formulation

$$\sum_{m=1}^{N}\sum_{k=0}^{K_i-1}\left(2\cdot\gamma_m^k+\rho\cdot(\alpha_i^k)^2\right)\cdot\delta t \quad (26)$$

by introducing γ_m^k and

$$\frac{1}{\sqrt{\beta_m^{k+1}}+\sqrt{\beta_m^k}}\le\gamma_m^k,\quad k=0,\ldots K_i-1;m=1,\ldots,N \quad (27)$$

as slack variables and additional constraints.

Equation (27) is further derived to a quadratic form as

$$\frac{1}{\zeta_m^{k+1}+\zeta_m^k}\le\gamma_m^k,\quad k=0,\ldots K_i-1;m=1,\ldots,N \quad (28)$$

$$\zeta_m^k\le\sqrt{\beta_m^k},\ k=0,\ldots K_i;\quad m=1,\ldots,N \quad (29)$$

by introducing ζ_m^k as slack variables.

Equation (28) can be formulated as a standard rotated quadratic cone

$$2\cdot\gamma_m^k\cdot\left(\zeta_m^{k+1}+\zeta_m^k\right)\ge\sqrt{2}^2 \quad (30)$$

which is denoted as

$$(\gamma_m^k,\zeta_m^{k+1}+\zeta_m^k,\sqrt{2})\in Q_r^3. \quad (31)$$

Also, (29) can be written as a standard (nonrotated) quadratic cone

$$\left(\beta_m^k+1\right)^2\ge\left(\beta_m^k-1\right)^2+\left(2\cdot\zeta_m^k\right)^2 \quad (32)$$

and is denoted as

$$(\beta_m^k+1,\beta_m^k-1,2\zeta_m^k)\in Q^3. \quad (33)$$

Finally, a slack variable s is introduced to transform the objective in (26) to an affine function

$$\sum_{m=1}^{N}\sum_{k=0}^{K_i-1}(2\cdot\gamma_m^k+\rho\cdot s)\cdot\delta t \quad (34)$$

with a rotated quadratic cone

$$2\cdot s\cdot 1\ge\sum_{m=1}^{N}\sum_{k=0}^{K_i-1}(\alpha_m^k)^2 \quad (35)$$

i.e.

$$(s,1,\boldsymbol{\alpha})\in Q_r^{2+\sum_{m=1}^{N}(K_i)} \quad (36)$$

where $\boldsymbol{\alpha}$ contains α_m^k in all pieces of the trajectory.

Also, the discretization is applied to $\alpha_m(t)$ and $\beta_m(t)$ in constraints listed in Section V-C2. Details are omitted for brevity. After that, we reformulate these constraints as affine equality and inequality functions. Besides, although we assume α_k is piecewise constant, we bound the changing rate of α_k considering the response time of the actuators of our quadrotor. We also write this changing rate constraint in an affine form

$$-\delta\alpha\le(\alpha_m^k-\alpha_m^{k-1})/\delta t\le-\delta\alpha \quad (37)$$

where $\delta\alpha$ (not jerk) is a predefined bound of the changing rate of acceleration. Since the difference of τ between t_m^k and t_m^{k-1} cannot be determined during the optimization, we only bound the changing rate of α_k in t domain.

The temporal optimization problem in Section V-C is formulated as a standard SOCP as follows:

$$\begin{aligned}
\min\quad & \mathbf{h}^T\boldsymbol{\gamma}+\rho\cdot s\\
\text{s.t.}\quad & \mathbf{A}_{\text{eq}}\cdot\mathbf{x}=\mathbf{b}_{\text{eq}}\\
& \mathbf{A}_{\text{ie}}\cdot\mathbf{x}\le\mathbf{b}_{\text{ie}}\\
& (s,1,\boldsymbol{\alpha})\in Q_r^{2+\sum_{m=1}^{N}(K_i)},\quad m=1,\ldots N\\
& (\gamma_m^k,\zeta_m^{k+1}+\zeta_m^k,\sqrt{2})\in Q_r^3,\quad k=0,\ldots,K_i-1\\
& (\beta_m^k+1,\beta_m^k-1,2\zeta_m^k)\in Q^3,\quad k=0,\ldots,K_i. \quad (38)
\end{aligned}$$

Here, $\boldsymbol{\gamma}$ and $\mathbf{x}$ consist of all γ^k and $\alpha^k,\beta^k,\zeta^k,\gamma^k$. δt is the resolution of discretization of the problem. The effect of different ρ and δt to the temporal trajectory and a more detailed derivation of the SOCP can be viewed in [37].

In our *teach–repeat–replan* system, since the global repeating trajectory always has static initial and final states, (38) is always mathematically feasible regardless of the solution of spatial optimization. That's because a feasible solution of the optimization program can always be found by infinitely enlarging the time. Combined with the fact that the spatial optimization also always has a solution (see Section V-A), once a flight corridor is given, a spatial–temporal trajectory must exist.

VI. Online Local Replanning

In our previous work [1], once the global planning finished, the drone would execute the trajectory without other considerations. This strategy is based on assumptions that 1) the map

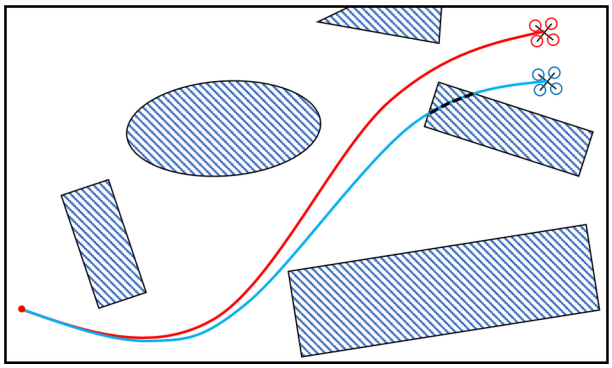


Fig. 9. Illustration of colliding with obstacles when there are significant pose drifts but no timely loop closure corrections. Obstacles are depicted in the global frame. The flight path of the drone in the VIO frame is shown in the red curve, and the actual trajectory in the global frame is the blue curve, which collides with obstacles on the global map.

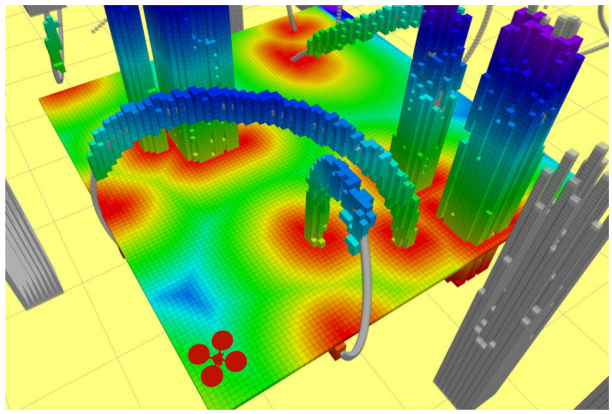


Fig. 10. Local occupancy map its corresponding ESDF map visualized at a given height of 0.6 m.

of the environment is perfectly built and remains intact and 2) globally consistent pose estimation is provided. We use a VIO system with loop closure to correct local pose drifts, and our dense map is globally deformed according to the global pose graph. However, the first assumption does not always hold, especially when new obstacles suddenly appear or the environment changes. As for the second assumption, our global pose estimation relies on the loop closure detection, which also does not guarantee an extremely high recall rate. In situations where there are significant pose drifts but no timely loop closure corrections, the drone may have collisions with obstacles, as in Fig. 9.

A. Local Replanning Framework

To address the above issues fundamentally, we propose a local replanning framework, which reactively wraps the global trajectory to avoid unmodeled obstacles. A sliding local map is maintained onboard, where obstacles are fused, and an Euclidean signed distance field (ESDF) [38] is updated accordingly. Note that the dense global map is attached to the global pose graph, but the local map introduced here is associated with the local VIO frame and slides with the drone.

1) ESDF Mapping: We adopt our previous work FIESTA [4], which is an advanced incremental ESDF [39] mapping framework, to build the local map for online replanning. FIESTA fuses the depth information into a voxel-hashed occupancy map [40] and updates the distance value of voxels as few as possible using a breadth-first search framework. It is lightweight and efficient and produces near-optimal results. Details can be checked in [4]. The ESDF is necessary for the following gradient-based trajectory wrapping. An example of a local occupancy map and its corresponding ESDF map are shown in Fig. 10. Note that, in our system, the range of current depth observation decides the range of the local map.

2) Sliding-Window Replanning: Due to limited onboard sensing range and computing resource, it is impossible and unnecessary to conduct global replanning. In this article, we maintain a temporal window sliding along the global trajectory and conduct local replanning within it. As shown in Fig. 11, when obstacles are observed to block the trajectory in the window, a replanned trajectory is generated to avoid obstacles and rejoin the global trajectory afterward.

B. Gradient-Based B-Spline Optimization

1) B-Spline Trajectory Formulation: A B-spline is a piecewise polynomial function defined by a series of control points $\{\mathbf{Q}_0, \mathbf{Q}_1, \ldots, \mathbf{Q}_N\}$ and knot vector $[t_0, t_1, \ldots, t_m]$. For a p-degree B-spline, we have $m = N + p + 1$. Following the matrix representation of the de Boor–Cox formula [41], the value of a B-spline can be evaluated as

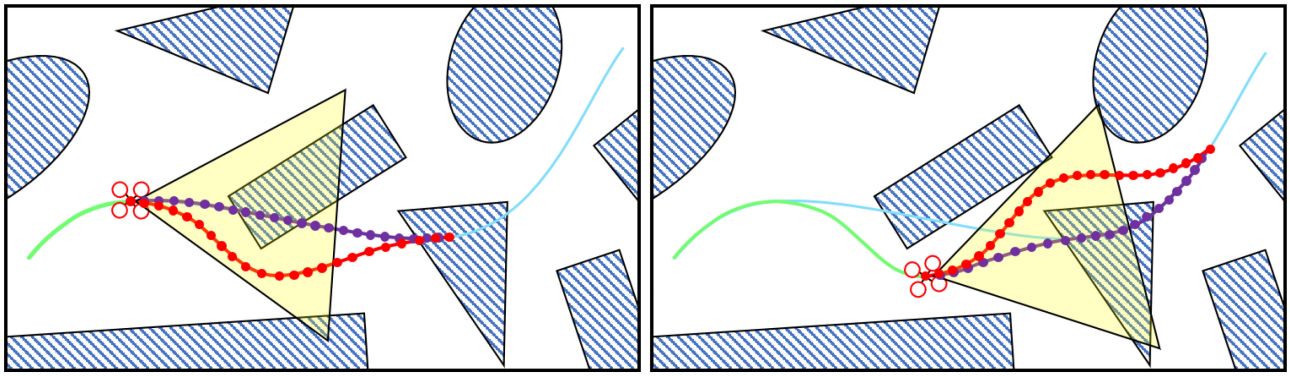
$$P(u) = [1, u, \ldots, u^p] \cdot \mathbf{M}_{p+1} \cdot [\mathbf{Q}_{i-p}, \mathbf{Q}_{i-p+1}, \ldots, \mathbf{Q}_i]^T. \tag{39}$$

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

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

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

3) Elastic Band Optimization: The basic requirements of the replanned B-spline are threefolds: smoothness, safety, and dynamical feasibility. We define the smoothness cost J_s using a



(a) (b)

Fig. 11. (a) and (b) Illustration of the online replanning mechanism. The blue and green curves are the global trajectory and the actual flight path of the drone, respectively. The purple curve and dots are the global trajectory in the sliding window and its corresponding control points. The red curve and dots are the replanned local trajectory and its control points. The yellow frustum shows the sensing horizon of the drone.

jerk-penalized elastic band cost function [42], [43]

$$J_s = \sum_{i=1}^{N-1} \| \underbrace{(\mathbf{Q}_{i+2} - 2\mathbf{Q}_{i+1} + \mathbf{Q}_i)}_{\mathbf{F}_{i+1,i}} - \underbrace{(\mathbf{Q}_{i+1} - 2\mathbf{Q}_i + \mathbf{Q}_{i-1})}_{\mathbf{F}_{i-1,i}} \|^2 = \sum_{i=1}^{N-1} \|\mathbf{Q}_{i+2} - 3\mathbf{Q}_{i+1} + 3\mathbf{Q}_i - \mathbf{Q}_{i-1}\|^2 \tag{40}$$

which can be viewed as a sum of the squared jerk of control points on the B-spline. Note here that we use this formulation, which is independent of the time parameterization of the trajectory instead of the traditional time-integrated cost function [13]. Because the time duration in each span of the B-spline may be adjusted after the optimization (see Section VI-B4), (40) captures the geometric shape of the B-spline regardless of the time parameterization. Besides, (40) bypasses the costly evaluation of the integration and is, therefore, more numerically robust and computationally efficient in optimization.

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

$$J_c = \sum_{i=p}^{N-p} F_c(d(\mathbf{Q}_i)) \tag{41}$$

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

$$F_c(d) = \begin{cases} (d - d_0)^2, & d \le d_0 \\ 0, & d > d_0 \end{cases} \tag{42}$$

where d_0 is the expected path clearance. J_v and J_a are applied to velocities and accelerations, which exceed the physical limits. The formulations of J_v and J_a are similar to (41) and are omitted here. The overall cost function is

$$J_{\text{total}} = \lambda_1 J_s + \lambda_2 J_c + \lambda_3 (J_v + J_a) \tag{43}$$

where λ_1, λ_2, and λ_3 are weighting coefficients. J_{total} can be minimized for a local optimal solution by general optimization methods such as Gauss–Newton or Levenberg–Marquardt.

In practice, a proper selection of weighting parameters is vital for robust replanning and should be tuned in simulation until a satisfactory performance. Weighting parameters are relative to flight speed and obstacle density, as well as replanning horizon. Basically, larger λ_1 results in shorter/smoother local trajectories, but with lower clearance to obstacles. A default parameter setting is given in our open-source project.

4) Iterative Refinement: In the above optimization problem, although collisions and dynamical infeasibilities are penalized, there is no hard guarantee on generating a strictly feasible solution. To improve the success rate in practice, we add a postprocess to refine the trajectory iteratively. In each iteration, we check collisions and feasibilities of all optimized control points. If collisions are detected, we increase the collision term J_c by increasing λ_2 and solve the optimization problem [see (43)] again.

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

$$\Delta t_i^{'} = \min \left\{ \alpha, \max \left\{ \frac{v_m}{v_{max}}, \left(\frac{a_m}{a_{max}} \right)^{\frac{1}{2}} \right\} \right\} \cdot \Delta t_i \tag{44}$$

where α is a constant slightly larger than 1, v_m and a_m are infeasible velocity and acceleration, respectively, and $v_{\max}$ and $a_{\max}$ are maximum allowed acceleration and velocity of the drone,

respectively. The time duration is iteratively enlarged until obtaining a feasible solution or exceeding the maximum iteration limit. If no feasible solution exists after the time adjustment, λ_3 is increased, and the trajectory is optimized again.

VII. Results

In this section, we validate the robustness and effectiveness of our proposed system by presenting several benchmark testing results. First, we test our overall system, conduct benchmark comparisons, and analyze the performance of several submodules in simulation. Second, we conduct aggressive teach–repeat–replan flights in both static and dynamic indoor complex environments. Finally, we test our system in two different outdoor environments.

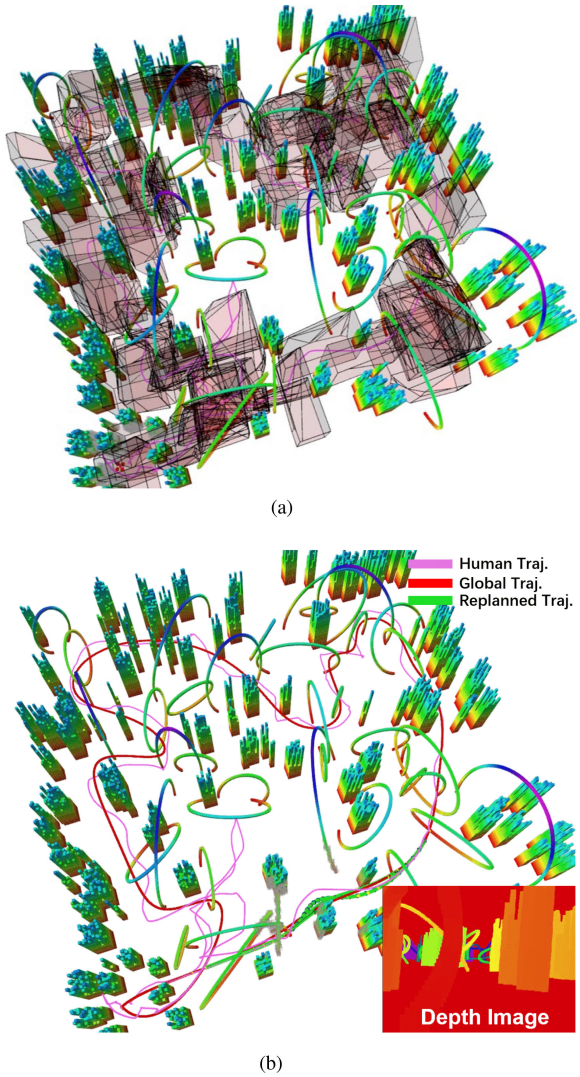
A. Implementation Details

The global planning method proposed in this article is implemented with a QP solver OOQP[7] and a SOCP solver Mosek.[8] The local replanning depends on a nonlinear optimization solver NLopt.[9] The source code of all modules in our quadrotor system, including local/global localization, mapping, and planning, is released as ros packages for the reference of the community. Readers of this article can easily replicate all the presented results. The state estimation, pose graph optimization, local mapping, local replanning, and the controller run onboard. Other modules are running on an offboard laptop, which has a GTX 1080[10] graphics card.

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

B. Simulated Flight Test

We first test our global and local planning methods in simulations. The simulated environments are randomly deployed with various types of obstacles and circles for drone racing, as shown in Fig. 12. The simulating tool we use is a lightweight simulator MockaFly,[11] which contains quadrotor dynamics model, controller, and map generator. The simulator is also released as an open-source package with this article. In the simulation, a drone is controlled by a joystick to demonstrate the teaching trajectory. The simulated drone is equipped with a depth camera whose depth measurements are real time rendered in GPU by back-projecting the drone's surrounding obstacles. We randomly add noise on the depth measurements to mimic a real sensor. The replanning module is activated in the simulation and is triggered by the noise added on the depth. The teaching trajectory and the flight corridor is shown in Fig. 12(a). The global trajectory, local replanned trajectory, and depth measurement are shown in Fig. 12(b). More details about the simulation are presented in the attached video.



(a)

(b)

Fig. 12. Trajectory generated in a complex simulated environment. The flight corridor consists of large free convex polyhedrons are shown in (a), and the optimized space–time trajectory is shown in (b). (a) Flight corridor captures local free space. (b) Local replanning trajectory and current depth image.

C. Benchmark Comparisons

1) Corridor Generation: We test the performance of the flight corridor generation (see Section IV) to show the efficacy of the proposed techniques for CPU (see Section IV-B) and GPU (see Section IV-C) accelerations. For convenience, we denote the basic process for doing *convex cluster inflation* as CPU_raw; CPU_raw with cube initialization as CPU+; the one with cube initialization, vertex selection, and early termination as CPU++; and the parallel version of the *convex cluster inflation* as GPU. We first compare the time consumed for finding the largest flight corridor with these methods to validate the improvements of efficiency by using our proposed CPU and GPU acceleration techniques. Then, we compare the ratio of space capturing by

[7][Online]. Available: http://pages.cs.wisc.edu/ swright/ooqp/
[8][Online]. Available: https://www.mosek.com
[9][Online]. Available: https://nlopt.readthedocs.io
[10][Online]. Available: https://www.nvidia.com/en-us/geforce/20-series/
[11][Online]. Available: https://github.com/HKUST-Aerial-Robotics/mockasimulator

TABLE I
COMPARISON OF COMPUTING TIME OF CORRIDOR GENERATION

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

TABLE II
COMPARISON OF SPACE CAPTURED OF CORRIDOR GENERATION

Space Ratio (%)	w/ Initialization	w/o Initialization	Previous [1]
Res. = $0.25m$	99.22	100.00	82.28
Res. = $0.20m$	99.56	100.00	82.92
Res. = $0.15m$	98.93	100.00	81.82
Res. = $0.10m$	97.06	100.00	82.78
Res. = $0.075m$	97.14	100.00	83.03

methods with and without the polyhedron initialization, and by our previous method [1]. The motivation of the latter comparison is twofold.

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

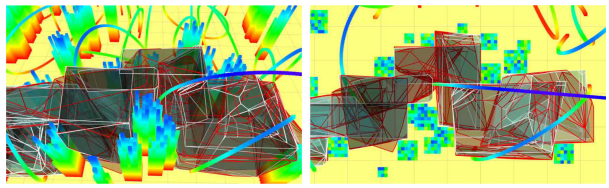
We generate ten random maps, with 10–20 random teaching trajectories given in each map. The average length of teaching trajectories is 20 m. Results are given in Tables I and II .

As shown in Table I, as the resolution of the map being finer, the computing time of the simple *convex cluster inflation* quickly becomes unacceptably high. In CPU, with the help of *polyhedron initialization*, the computational efficiency is improved several times. Moreover, according to Table I, introducing the *voxel selection* and *early termination* can increase the speed more than one order of magnitude in a fine resolution. The efficacy of the GPU acceleration is even more significant. As shown in Table I, the GPU version improves the computing speed 30 times at a fine resolution (0.075 m) and ten times at a coarse resolution (0.25 m). For a finer resolution, more candidate voxels are discovered in one iteration of Algorithm 3; thus, more computations are conducted parallelly to save time.

For the second comparison, we count the number of free voxels included in the flight corridor found by each method. At each resolution, we take the result of the method without initialization as 100% and compare others against it, as shown in Fig. 13. Table II indicates two conclusions.

1) Using polyhedrons instead of axis-aligned cubes can significantly increase the volume of the flight corridor.
2) Using initialization only slightly sacrifices the volume of the flight corridor, and the sacrifice is neglectable in a medium or coarse resolution (0.15–0.25 m).

The first conclusion holds because a simple cube only discovers free space in x, y, z directions and sacrifices much space in a highly nonconvex environment, as in Fig. 4. The second conclusion comes from the fact that in a highly nonconvex environment, a regular shaped polyhedron (a cube) does not prevent the following voxel clustering in its nearby space. It shows that the initialization plus the clustering refinement does not harm the volume of the final polyhedron and is acceptable in practice, especially for a coarse resolution.



(a) (b)

Fig. 13. (a) and (b) Flight corridor generated with and without the initialization process. Polyhedrons with bounding edges in white and red are found by methods with and without the initialization, respectively.

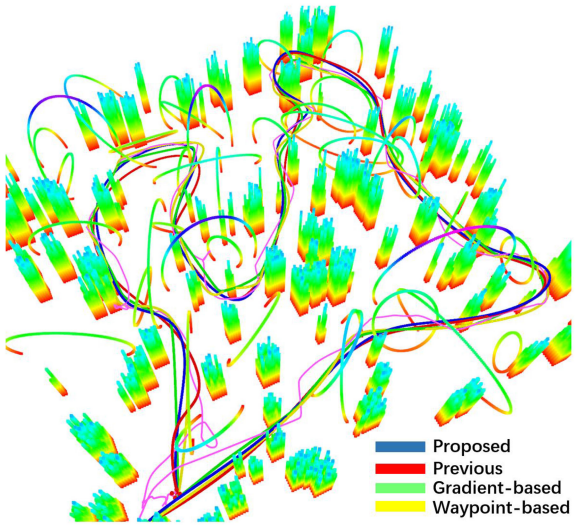


Fig. 14. Comparison of trajectories optimized by different methods. The manual flight trajectory is shown as the purple curve. Blue, red, green, and yellow trajectories are generated by our proposed method, our previous method [1], gradient-based method [44], and waypoint-based method [14], respectively.

Besides, we conduct an additional test of the corridor generation against teaching with different levels of smoothness to show our method is insensitive to users' piloting skills. Details can be viewed in the supplementary video.[12]

2) Global Planning: We compare the proposed global planning method against our previous work [1] and other representative optimization-based trajectory generation methods, such as the waypoint-based method [14] and the gradient-based method [44]. For the latter two benchmarked methods, there is no explicit way to capture the topological structure of the teaching trajectory. Therefore, we convert the teaching trajectory to a piecewise path by recursively finding a collision-free straight line path along with it. Then, we use this path to initialize these methods [14], [44]. For a fair comparison, parameters in benchmarked methods are tuned to achieve the best performances before the test. We randomly generate ten simulated

[12][Online]. Available: https://youtu.be/r0pLwPlKS2E

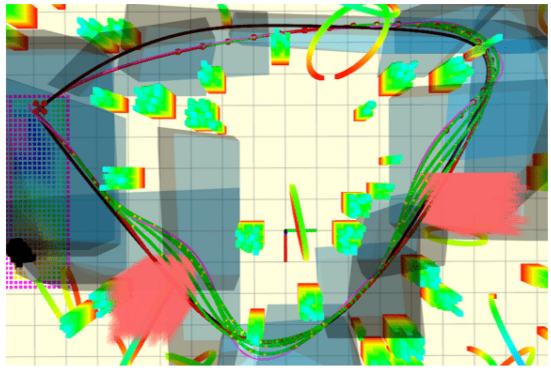
TABLE III
COMPARISON OF TRAJECTORY OPTIMIZATION

Method	Length (m)	Time (s)	Energy $((m/s^3)^2)$
Proposed Method	**84.607**	**55.154**	**83.350**
Previous Method	86.723	57.736	89.883
Gradient-based [44]	89.622	111.398	109.575
Waypoint-based [14]	97.045	94.895	204.267

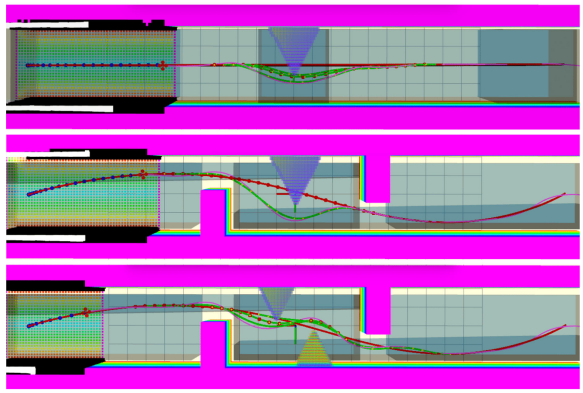
environments with dense obstacles, as in Section VII-B, and conduct ten teach-and-repeat trials in each map. A sample result is shown in Fig. 14.

As shown in Table III, our proposed method outperforms in all length, time, and energy aspects. The waypoint-based [14] method can only add extra intermediate waypoints on the initial path. Therefore, it is mostly dominated by its initialization and likely to output a low-quality solution. The gradient-based [44] method has no such restriction and can adjust the path automatically by utilizing gradient information. However, its optimization formulation is underlying nonconvex, since the collision cost is defined on a nonconvex ESDF. Therefore, the gradient-based [44] method can only find a locally optimal solution around its initial guess. Compared to these two methods, our method is initialization free and enjoys the convexity to find the global energy-optimal and time-optimal solutions in the flight corridor. Also, a smoother trajectory tends to generate a faster time profile. Therefore, finally, under the same coordinate descent framework, our method always outperforms [44] and [14]. Compared to [1], the proposed corridor generation method (see Section IV) can always capture more free space, as shown in Section VII-C1. Naturally, it provides much more freedom for global planning and results in much better solutions.

3) Local Replanning: We also present a case study of our local replanning module. We test the average computing time and the iteration number of the iterative refinement operation (see Section VI-B4), with different flight aggressiveness and environments. Simulated flights are first conducted in a random complex environments similar to Fig. 14, as shown in Fig. 15(a). Then, we manually generate three testing scenarios with different difficulty levels, named *Simple*, *Normal*, and *Hard*, as shown in Fig. 15(b). In Table IV, the *Nominal Velocity* only means a rough expected flight speed of the drone, and the *Approaching Velocity* is the real speed when the drone starts to replan a trajectory. In each testing scenario, we incrementally increase the expected aggressiveness and record results until the drone fails to replan successfully. Note that, in each test, due to the limited sensing field of view and horizon, the drone may replan several times even there is only one obstacle. The computing time and the iteration number are measured by taking an average of all replannings occurred in five flights of each scenario. As shown in Table IV, our local replanner survives at up to 5.5, 4.5, and 3.5 m/s approaching velocity. In each scenario, the average of computing time is around 5–6 ms, which is totally acceptable for onboard usage. Considering that this case study is done on a Laptop with i5-6300HQ CPU, which is far less potent than our i7-8550 U onboard computer (almost 1.4 times more powerful[13]), the efficiency of our local replanner is validated. Also, as shown in Table IV, our replanner usually only iterates one to two times before it finally finds a feasible solution.



(a)



(b)

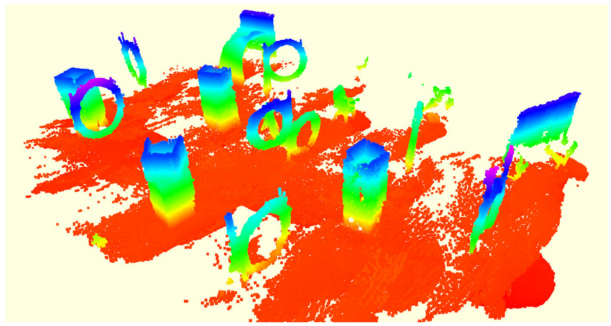
Fig. 15. Benchmark test of the local replanning system. (a) Complex test. (b) Simple, normal, and hard tests from the top-down. Transparent blue polyhedrons represent the flight corridor. Other marks are interpreted as the same as in Fig. 14.

TABLE IV
BENCHMARK STUDY OF LOCAL REPLANNER

	Nominal Velocity (m/s)	Approaching Velocity (m/s)	Computing Time (ms)	Iteration
Complex	1.5	2.182	4.204	1.142
	2.5	2.894	5.01	1.368
	3.5	2.89	4.448	1.242
	4.5	2.858	4.44	1.248
	5.5	3.24	4.882	1.312
Easy	1.5	1.54	5.742	1.9
	2.5	2.52	5.392	1.9
	3.5	3.596	4.394	1.416
	4.5	4.334	4.67	1.416
	5.5	4.954	5.194	2.034
Normal	1.5	1.578	5.414	2.532
	2.5	2.488	6.944	2.4
	3.5	3.526	5.784	1.584
	4.5	4.284	3.932	1.55
	5.5	/	/	/
Hard	1.5	1.746	5.632	1.62
	2.5	2.646	4.162	1.066
	3.5	3.522	5.014	1.506
	4.5	/	/	/
	5.5	/	/	/

[13][Online]. Available: https://cpubenchmark.net/compare/Intel-i7-8550U-vs-Intel-i5-6300HQ/3064vs2632

(a)



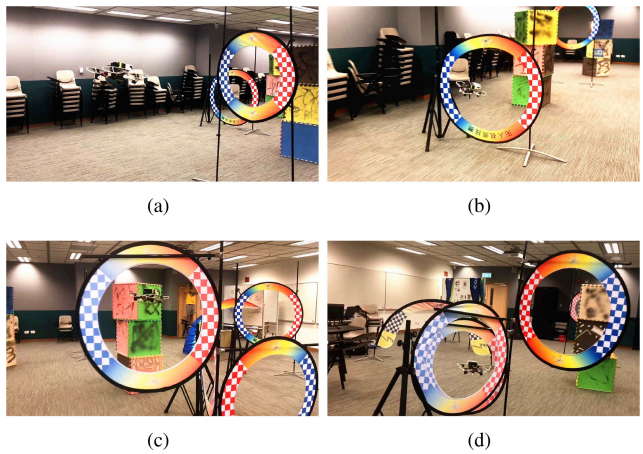
(b)

Fig. 16. Experimental setup of the fast indoor drone racing flights. (a) Obstacle deployment. (b) Prebuilt globally consistent map. The colored code indicates the height of obstacles. (a) Indoor testing environment. (b) Global consistent dense map.

D. Indoor Flight Test

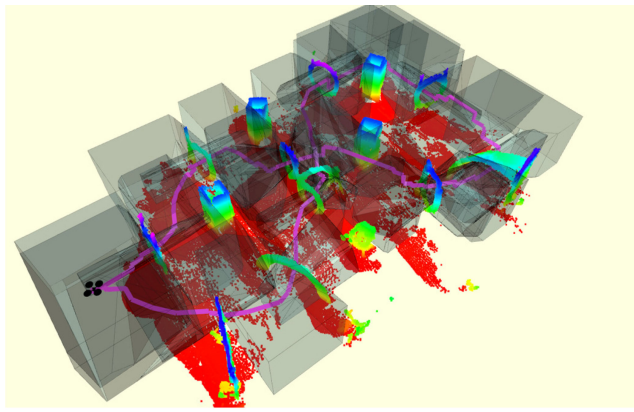
1) Fast Flight in a Static Environment: First, we conduct an experiment in a cluttered drone racing scenario. This experiment validates the robustness of our proposed system and also pushes the boundary of the aggressive flight of quadrotors. Several different types of obstacles, including circles, arches, and tunnels, are deployed randomly to composite a complex environment, as shown in Fig. 16(a). The smallest circle only has a diameter of 0.6 m, which is very narrow compared to the 0.3 $\times$ 0.3 m tip-to-tip size of our drone. The maximum velocity and acceleration of the drone are set as 3 and 3 m/s^2, respectively, and the parameter ρ in (13) is set as 0, which means that the quadrotor is expected to fly as fast as possible as long as it respects the dynamical limits. A dense global consistent map is prebuilt using the method stated in Section III-B. During the teaching phase, the quadrotor is virtually piloted by a human to move amid obstacles. Then, the quadrotor autonomously converts this teaching trajectory to a global repeating trajectory and starts to track it. Snapshots of the drone in the flight are shown in Fig. 17. The teaching trajectory and the convex safe flight corridor are visualized in Fig 18(a), and the global repeating trajectory is shown in Fig. 18(b).

2) Local Replanning Against Unknown Obstacles: Our system can deal with changing environments and moving obstacles. In this experiment, we test our system also in the drone racing site to validate our local replanning module. Several obstacles are moved or added to change the drone racing environment significantly, and some others are dynamically added during the repeating flight, as shown in Fig. 19. In this experiment, the maximum velocity and acceleration for the quadrotor are set as

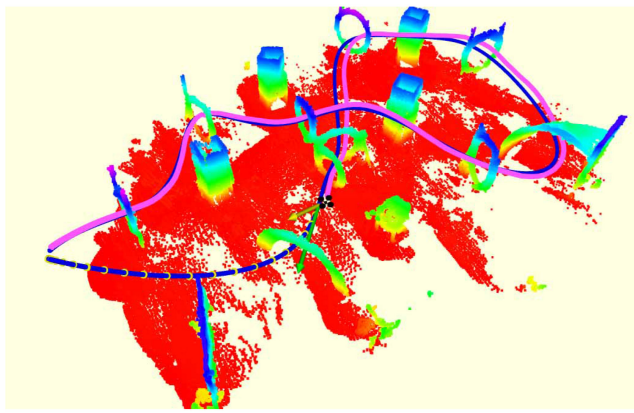


(a) (b) (c) (d)

Fig. 17. (a)–(d) Snapshots of the fast autonomous flight in a static environment.



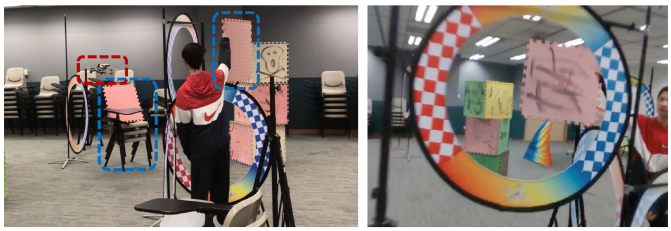
(a)



(b)

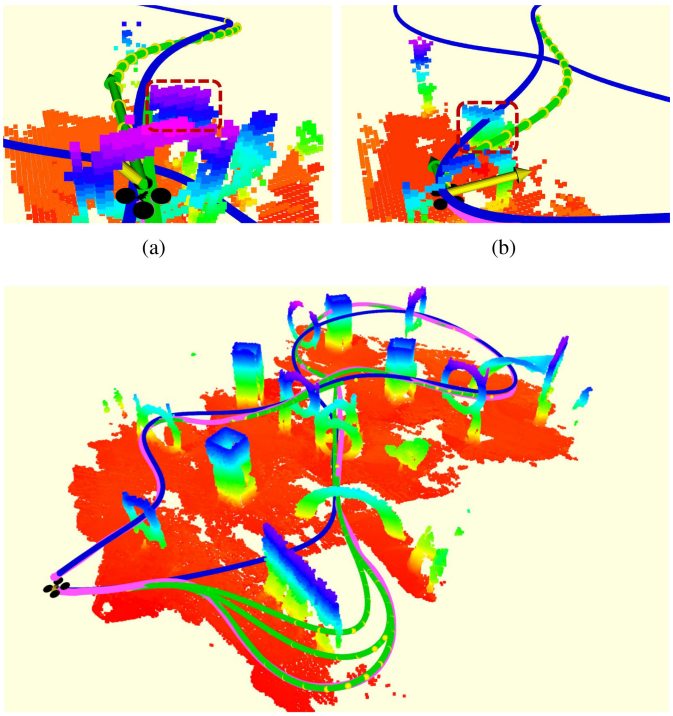
Fig. 18. Indoor flight in a static environment. In (a), the flight corridor is represented by transparent blue polyhedrons. In (b), blue, green, and purple curves are global trajectory, local trajectory, and quadrotor flight path, respectively. (a) Teaching trajectory and the flight corridor. (b) Spatial–temporal optimal repeating trajectory.

2 and 2 m/s^2, respectively. The local ESDF map is sliding with the drone using a ring-buffered updating mechanism [45]. The resolution of the local perception is 0.075 m. The size of the map is decided by points observed spreading in the current frame. The horizon and frequency of the local replanning are 3.5 s and 15 Hz,



(a) (b)

Fig. 19. Local replanning experiment against unmapped and moving obstacles. The drone and unmapped obstacles are labeled by the red and blue dashed rectangles, respectively, for clear visualization. (a) Side view. (b) Onboard first-person view.



(a) (b) (c)

Fig. 20. Indoor flight in a dynamic environment. In (a) and (b), the unmapped new obstacle and moving obstacles are labeled by red dashed rectangles, and colored voxels represent local obstacle maps. In (c), colored voxels show the global map. Other marks are interpreted as the same as in previous figures. (a) Replanning, unmapped obstacle. (b) Replanning, moving obstacle. (c) Overview of all replanning trajectories.

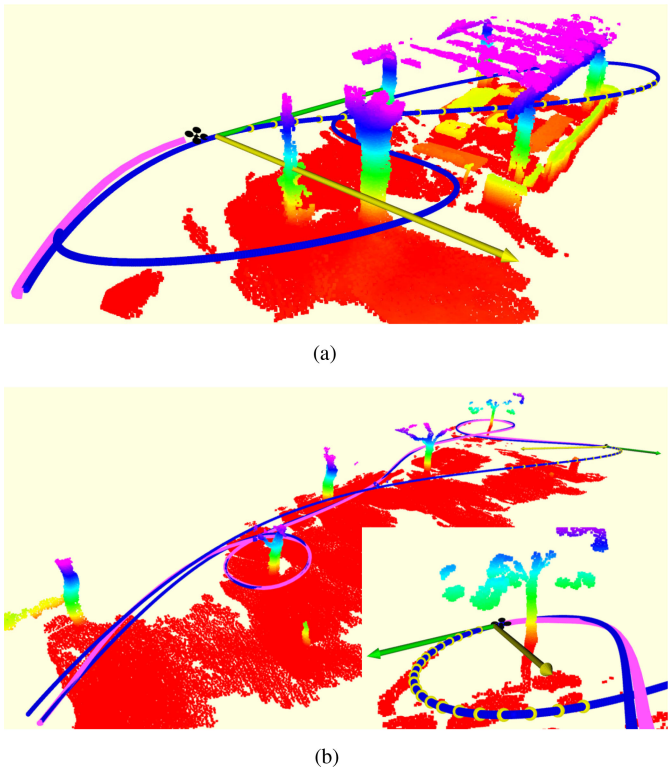
respectively. Replanning is triggered eight times during the flight in this experiment, and local safe and dynamical feasible splines are generated on time accordingly. Local trajectories, local maps, and the overview of this experiment are shown in Fig. 20. We refer readers to the attached video for more details.

E. Outdoor Flight Test

Finally, we conduct quadrotor flight experiments with a much higher aggressiveness in two different outdoor scenes, as in Fig. 21, to show the robustness of our system in natural environments. Although these experiments are conducted outdoor, GPS

(a) (b)

Fig. 21. Snapshots of the fast autonomous flights in outdoor environments. (a) Outdoor experiment, trial 1. (b) Outdoor experiment, trial 2.



(a) (b)

Fig. 22. Outdoor flights in two different scenarios. Marks are interpreted as the same as in Fig. 18. (a) Outdoor experiment, trial 1. (b) Outdoor experiment, trial 2. A closeup view is in the right-down side.

or other external positioning devices are not used. The *teach–repeat–replan* pipeline is as the same as indoor experiments VII-D. The velocity and acceleration limits for these two trials are set as 5 m/s, 6 m/s^2 and 7 m/s, 6 m/s^2, respectively. Since the flight speed is significantly higher than indoor experiments, we set a smaller replanning horizon as 2.0 s. Results such as the global and local trajectory and the global map are visualized in Fig. 22. More clear visualizations of outdoor experiments are given in the video.

VIII. Discussion, Conclusion, and Future Work

A. Discussion

In our system, the robustness of the replanning module depends on properly tuned parameters, i.e., weights in (43). Since the replanning problem has a highly nonlinear and nonconvex objective function, and an infeasible initial guess, it cannot be theoretically guaranteed a strict feasibility. In practice, we keep

replanning and try to find a feasible solution before the drone has to brake. A benchmark study of the replanner against different aggressiveness and obstacle densities is given in Section VII-C3. In our recent work [46], we significantly improve the success rate of the replanning, by using a topological path finder to guide the following B-spline optimization. In this way, several paths with distinct topologies serve as different guidance and, therefore, provide many solution alternatives subject to different local minima. The best solution is then selected from all optimized trajectories.

Our proposed system can also work in scenarios where a global map and an optimized global reference trajectory are not available. Also, in our recent work [46], we present an experiment where no prior map information is given, and the global trajectory is set as a simple straight line. In this scenario, our system shows aggressive flight performance with full autonomy by only local replanning.

B. Conclusion

In this article, we proposed a complete and robust robotic system, *teach–repeat–replan*, for quadrotor aggressive flights in complex environments. The main idea of this article was to find the topological equivalent free space of the user's teaching trajectory. Then, we used spatial–temporal optimization to obtain an energy–time-efficient repeating trajectory and incorporate online perception and local replanning to ensure the safety against environmental changes, moving obstacles, and localization drifts. The repeating trajectory captures a user's intention and respects an expected flight aggressiveness, which enables autonomous flights much more aggressive than manual piloting in complex environments. Our system is also flexible and easily replicable, as evidenced by various types of experiments presented in this article, and a third-party application.[14] We released all components of our system for the reference of the community.

C. Future Work

Most vision-based aerial vehicles, such as our platform, do not have omnidirectional perception capability. For these systems, the planning of yaw angle, or the so-called view planning, plays an essential role in choosing which direction to observe during flights. Currently, view planning is not considered in either global or local planning of our system. The yaw angle of our drone is controlled independently to follow the tangential direction along the global or local trajectory. However, this is apparently not the best strategy for commanding the yaw angle of the drone. One practical strategy is to treat all unknown space as occupied and plan trajectories only in known free space. However, this is contradictory to the wish of high flight aggressiveness. In the future, we plan to study the optimal view planning of the vision-based drone, considering localization uncertainty and map constraints.

[14]Flight demos at the Department of Electrical and Mechanical Services, Hong Kong government. Video: [Online]. Available: https://youtu.be/Ut8WT0BURrM

References

[1] F. Gao *et al.*"Optimal trajectory generation for quadrotor teach-and-repeat," *IEEE Robot Autom. Lett.*, vol. 4, no. 2, pp. 1493–1500, Apr. 2019.

[2] M. Fehr, T. Schneider, M. Dymczyk, J. Sturm, and R. Siegwart, "Visual-inertial teach and repeat for aerial inspection," 2018, *arXiv:1803.09650.*

[3] P. Furgale and T. D. Barfoot, "Visual teach and repeat for long-range rover autonomy," *J. Field Robot.*, vol. 27, no. 5, pp. 534–560, 2010.

[4] L. Han, F. Gao, B. Zhou, and S. Shen, "FIESTA: Fast incremental euclidean distance fields for online motion planning of aerial robots," 2019, *arXiv:1903.02144*.

[5] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3529–3536, Oct. 2019.

[6] C. Sprunk, G. D. Tipaldi, A. Cherubini, and W. Burgard, "Lidar-based teach-and-repeat of mobile robot trajectories," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 3144–3149.

[7] P. Furgale, P. Krüsi, F. Pomerleau, U. Schwesinger, F. Colas, and R. Siegwart, "There and back again—Dealing with highly-dynamic scenes and long-term change during topological/metric route following," in *Proc. Workshop Model., Estimation, Perception, Control Terrain Mobile Robots*, 2014. [Online]. Available: https://furgalep.github.io/bib/furgale_icra14.pdf

[8] P. Krüsi, B. Bücheler, F. Pomerleau, U. Schwesinger, R. Siegwart, and P. Furgale, "Lighting-invariant adaptive route following using iterative closest point matching," *J. Field Robot.*, vol. 32, no. 4, pp. 534–564, 2015.

[9] C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot, "Visual teach and repeat, repeat: Iterative learning control to improve mobile robot path tracking in challenging outdoor environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 176–181.

[10] M. Paton, K. MacTavish, M. Warren, and T. D. Barfoot, "Bridging the appearance gap: Multi-experience localization for long-term visual teach and repeat," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 1918–1925.

[11] M. Paton, K. MacTavish, C. J. Ostafew, and T. D. Barfoot, "It's not easy seeing green: Lighting-resistant stereo visual teach & repeat using color-constant images," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 1519–1526.

[12] L.-P. Berczi and T. D. Barfoot, "It's like Déjà Vu all over again: Learning place-dependent terrain assessment for visual teach and repeat," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 3973–3980.

[13] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robot. Autom.*, Shanghai, China, May 2011, pp. 2520–2525.

[14] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Proc. Int. Symp. Robot. Res.*, Dec. 2013, pp. 649–666.

[15] F. Gao, W. Wu, Y. Lin, and S. Shen, "Online safe trajectory generation for quadrotors using fast marching method and Bernstein basis polynomial," in *Proc. IEEE Int. Conf. Robot. Autom.*, Brisbane, QLD, Australia, May 2018, pp. 344–351.

[16] F. Gao and S. Shen, "Online quadrotor trajectory generation and autonomous navigation on point clouds," in *Proc. IEEE Int. Symp. Safety, Secur., Rescue Robot.*, Lausanne, Switzerland, 2016, pp. 139–146.

[17] F. Gao, W. Wu, W. Gao, and S. Shen, "Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments," *J. Field Robot.*, vol. 36, pp. 710–733, 2019.

[18] S. Liu *et al.*,"Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments," *IEEE Robot. Autom. Lett.*, vol. 2, no. 3, pp. 1688–1695, Jul. 2017.

[19] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic Foundations of Robotics XI*, vol. 107. New York, NY, USA: Springer, 2015, pp. 109–124.

[20] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "CHOMP: Gradient optimization techniques for efficient motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2009, pp. 489–494.

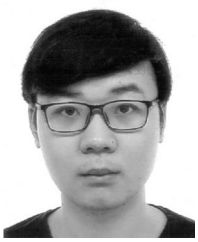
[21] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, "Continuous-time trajectory optimization for online UAV replanning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Daejeon, South Korea, Oct. 2016, pp. 5332–5339.

[22] Y. Lin *et al.*,"Autonomous aerial navigation using monocular visual-inertial fusion," *J. Field Robot.*, vol. 35, no. 1, pp. 23–51, 2018.

[23] H. M. Choset *et al.*,*Principles of Robot Motion: Theory, Algorithms, and Implementation*. Cambridge, MA, USA: MIT Press, 2005.

[24] M. Roberts and P. Hanrahan, "Generating dynamically feasible trajectories for quadrotor cameras," *ACM Trans. Graph.*, vol. 35, no. 4, 2016, Art. no. 61.

[25] J. Jamieson and J. Biggs, "Near minimum-time trajectories for quadrotor UAVs in complex environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 1550–1555.
[26] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Trans. Autom. Control*, vol. 54, no. 10, pp. 2318–2327, Oct. 2009.
[27] Q.-C. Pham, "A general, fast, and robust implementation of the time-optimal path parameterization algorithm," *IEEE Trans. Robot.*, vol. 30, no. 6, pp. 1533–1540, Dec. 2014.
[28] H. Pham and Q.-C. Pham, "A new approach to time-optimal path parameterization based on reachability analysis," *IEEE Trans. Robot.*, vol. 34, no. 3, pp. 645–659, Jun. 2018.
[29] S. J. Wright, "Coordinate descent algorithms," *Math. Program.*, vol. 151, no. 1, pp. 3–34, 2015.
[30] T. Lee, M. Leoky, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *Proc. IEEE Control Decis. Conf.*, Atlanta, GA, USA, Dec. 2010, pp. 5420–5425.
[31] T. Qin, P. Li, and S. Shen, "VINS-Mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Trans. Robot.*, vol. 34, no. 4, pp. 1004–1020, Aug. 2018.
[32] K. Wang, F. Gao, and S. Shen, "Real-time scalable dense surfel mapping," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 6919–6925.
[33] F. Blochliger, M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart, "Topomap: Topological mapping and navigation based on visual SLAM maps," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 3818–3825.
[34] S. Lay, *Convex Sets and Their Applications*. Chelmsford, MA, USA: Courier Corporation, 2007.
[35] C. B. Barber, D. P. Dobkin, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Trans. Math. Softw.*, vol. 22, no. 4, pp. 469–483, 1996.
[36] K. Fukuda and A. Prodon, "Double description method revisited," in *Proc. Franco-Jpn. Franco-Chin. Conf. Combinatorics Comput. Sci.*, 1995, pp. 91–111.
[37] F. Gao, W. Wu, J. Pan, B. Zhou, and S. Shen, "Optimal time allocation for quadrotor trajectory generation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Madrid, Spain, Oct. 2018, pp. 4715–4722.
[38] P. F. Felzenszwalb and D. P. Huttenlocher, "Distance transforms of sampled functions," *Theory Comput.*, vol. 8, no. 1, pp. 415–428, 2012.
[39] T. Schouten and E. L. van den Broek, "Incremental distance transforms (IDT)," in *Proc. 20th Int. Conf. Pattern Recognit.*, 2010, pp. 237–240.
[40] M. Klingensmith, I. Dryanovski, S. Srinivasa, and J. Xiao, "Chisel: Real time large scale 3D reconstruction onboard a mobile device using spatially hashed signed distance fields," in *Proc. Robot.: Sci. Syst. Conf.*, 2015, vol. 4, p. 1.
[41] C. de Boor, "Subroutine package for calculating with b-splines," *Los Alamos Sci. Lab.*, Los Alamos, NM, USA, Tech. Rep. LA-4728, 1971.
[42] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1993, pp. 802–807.
[43] Z. Zhu, E. Schmerling, and M. Pavone, "A convex optimization approach to smooth trajectories for motion planning with Car-like robots," in *Proc. IEEE Control Decis. Conf.*, 2015, pp. 835–842.
[44] F. Gao, Y. Lin, and S. Shen, "Gradient-based online safe trajectory generation for quadrotor flight in complex environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Sep. 2017, pp. 3681–3688.
[45] V. Usenko, L. von Stumberg, A. Pangercic, and D. Cremers, "Real-time trajectory replanning for MAVs using uniform b-splines and a 3D circular buffer," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 215–222.
[46] B. Zhou, F. Gao, J. Pan, and S. Shen, "Robust real-time UAV replanning using guided gradient-based optimization and topological paths," 2020, *arXiv:1912.12644.*



Fei Gao received the B.Eng. degree in control science and engineering from Zhejiang University, Hangzhou, China, in 2015, and the Ph.D. degree in electronic and computer engineering from the Hong Kong University of Science and Technology, Hong Kong, in 2019.

He is currently an Assistant Professor with the Department of Control Science and Engineering, Zhejiang University, where he co-directs the Field Autonomous System and Computing Laboratory. His research interests include unmanned aerial vehicles, autonomous navigation, motion planning, optimization, and localization and mapping.

Luqi Wang received the B.Eng. degree in computer engineering and aerospace engineering in 2018 from the Hong Kong University of Science and Technology, Hong Kong, where he is currently working toward the Ph.D. degree in electronic and computer engineering.

His research interests include control, navigation, and path planning for autonomous robots.

Boyu Zhou received the B.Eng. degree in mechanical engineering from Shanghai Jiao Tong University, Shanghai, China, in 2018. He is currently working toward the Ph.D. degree in electronic and computer engineering with the Hong Kong University of Science and Technology, Hong Kong.

His research interests include aerial robots, autonomous navigation, motion planning, and dense mapping.

Xin Zhou received the B.Eng. degree in electrical engineering and automation from the China University of Mining and Technology, Xuzhou, China, in 2019. He is currently working toward the M.Phil. degree in control engineering with Zhejiang University, Hangzhou, China.

His research interests include motion planning and mapping for unmanned aerial vehicles.

Jie Pan received the B.Eng. degree in software engineering from Xidian University, Xi'an, China, in 2017, and the M.Phil. degree in electronic and computer engineering from the Hong Kong University of Science and Technology, Hong Kong, in 2019.

He is currently a Research Assistant with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology. His research interests include state estimation, sensor fusion, localization and mapping, and autonomous navigation in complex environments.

Shaojie Shen received the B.Eng. degree in electronic engineering from the Hong Kong University of Science and Technology, Hong Kong, in 2009, and the M.S. degree in robotics and the Ph.D. degree in electrical and systems engineering from the University of Pennsylvania, Philadelphia, PA, USA, in 2011 and 2014, respectively.

In September 2014, he joined the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, as an Assistant Professor. His research interests include robotics and unmanned aerial vehicles, with a focus on state estimation, sensor fusion, computer vision, localization and mapping, and autonomous navigation in complex environments.