2016 IEEE International Conference on Robotics and Automation (ICRA)
Stockholm, Sweden, May 16-21, 2016

Online Generation of Collision-Free Trajectories for Quadrotor Flight in Unknown Cluttered Environments

Jing Chen, Tianbo Liu, and Shaojie Shen

Abstract— **We present an online method for generating collision-free trajectories for autonomous quadrotor flight through cluttered environments. We consider the real-world scenario that the quadrotor aerial robot is equipped with limited sensing and operates in initially unknown environments. During flight, an octree-based environment representation is incrementally built using onboard sensors. Utilizing efficient operations in the octree data structure, we are able to generate free-space flight corridors consisting of large overlapping 3-D grids in an online fashion. A novel optimization-based method then generates smooth trajectories that both are bounded entirely within the safe flight corridor and satisfy higher order dynamical constraints. Our method computes valid trajectories within fractions of a second on a moderately fast computer, thus permitting online re-generation of trajectories for reaction to new obstacles. We build a complete quadrotor testbed with onboard sensing, state estimation, mapping, and control, and integrate the proposed method to show online navigation through complex unknown environments.**

I. Introduction

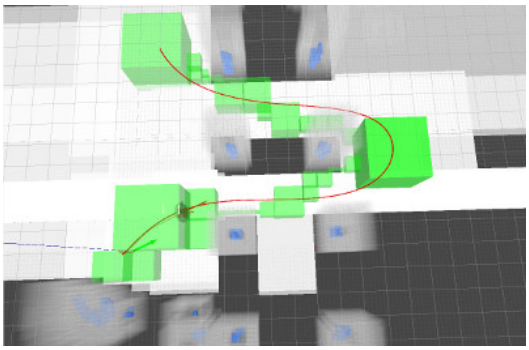
We address the problem of online generation of collision-free trajectories for a quadrotor to fly from an initial state to a final state through unknown cluttered environments. We consider the scenario that onboard sensing is limited and obstacles in the environment can only be mapped during flight. This requires that the generated trajectory satisfies three important requirements: first, the trajectory has to be smooth and dynamically feasible for a quadrotor; second, the *entire* trajectory, rather than a limited number of control points, has to be *guaranteed* to be collision-free; third, all sensing, map building, and trajectory planning have to be done in an online manner such that the quadrotor is able to react to new obstacles observed during flight.

Quadrotors have complex and underactuated dynamics, making it difficult to explicitly consider their full state. Fortunately, as shown in the recent results from Mellinger and Kumar [1], a quadrotor is differentially flat with respect to the position and yaw angle. In other words, the full state of a quadrotor can be expressed as a function of the instantaneous values of the x, y, z position and the yaw angle, and their derivatives. Therefore, any trajectory that is smooth and has bounded derivatives, where the bounded derivatives are specified by the mechanical configuration and limitations in motor dynamics, can be followed by a quadrotor.

All authors are with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong, China. jchenbr@connect.ust.hk, tliuam@connect.ust.hk, eeshaojie@ust.hk

This work was supported by HKUST project R9341. The authors would also like to thank the research donation and equipment support from DJI.

(a) Autonomous navigation in unknown outdoor environments



(b) Visualization of map structure (blue), initial (green) and final (white) flight corridor, and the smooth trajectory (red).

Fig. 1. Our experimental testbed developed from the DJI Matrice 100 quadrotor. It is equipped with an Intel NUC (Intel Core i54250U, 16GB RAM) for onboard computation. Onboard sensors include a laser range finder, two forward-facing cameras and a downward-facing camera. State estimation is obtained by onboard UKF fusion of stereo vision- and laser-based visual odometry, as well as optical-flow-based velocity estimation. The system achieves fully onboard autonomous navigation with obstacle detection using the onboard laser. A video of the experiment can be found at http://www.ece.ust.hk/~eeshaojie/icra2016jing.mp4.

We follow the minimum snap cost function, as in [1], and use piecewise polynomials to ensure smoothness of the trajectory, followed by our method to ensure dynamical feasibility (Sect. V-D). This satisfies the first requirement. Our key contribution is addressing the second requirement with our two-step trajectory generation approach that bounds the entire trajectory (Sect. V) within a free-space flight corridor, where the corridor is generated online using efficient operations in an octree-based map representation (Sect. IV). The third requirement is ensured by the fact that our method is able to compute valid trajectories in fractions of a second (Fig. 10). We are able to apply our method in an online fashion in which the map is built incrementally during flight.

Our trajectory generation method is integrated into our multi-sensor quadrotor testbed with onboard sensing, state estimation, mapping, and control, to form a complete naviga-

978-1-4673-8026-3/16/$31.00 ©2016 IEEE

tion system. Results in both simulation and real-world online experiments are presented to demonstrate autonomous flight with velocities up to 1.8m/s (Sect. VI). We also show superior performance compared to state-of-the-art approaches [2]. To the best of our knowledge, we are the first to show a quadrotor smoothly navigate through cluttered unknown environments with a safety guarantee (Fig. 12).

II. Related Works

Prior works on the topic of robotic motion planning and trajectory generation are extensive, covering a wide range of applications in robotic manipulation, self-driving, autonomous flight, etc. Any attempt at presenting a review of prior works within the limited space available will be incomplete, but we provide a discussion about recent advances that are the most relevant to autonomous flight of aerial robots.

The challenge in trajectory planning for UAVs operating in cluttered environments lies in the difficulty of modeling a large number of possibly non-convex obstacles or determining the traversable space. Earlier studies considered the simplified convex case and proposeed mixed-integer linear programming (MILP) approaches to encode obstacle information [3]–[6]. In [7], building on the pioneering work on minimum-snap trajectory generation in [1], the authors proposed to model faces of non-convex obstacles using mixed-integer quadratic programming. Instead of modeling obstacles directly, [8] proposed finding large convex segments to cover the free-space [9] and then performing simultaneous segment assignment and safe trajectory generation using mixed-integer programming. However, the computational complexity of all these mixed-integer approaches can quickly become intractable with an increasing number obstacles or free-space regions.

To generate trajectories with reasonable computational complexity, local [10] and reactive [11] methods have also been widely studied, but these have the downside of lacking global optimality. Waypoint-based approaches, where trajectory generation is seeded by position constraints, are also widely favorable for reducing computational cost. In an initial step, a collision-free waypoint path can be found using search-based [12] or randomized [2] algorithms. The waypoint path is then converted into time parameterized trajectories [2, 12, 13] or direct control inputs [14]. A noticeable work in this category is [2], in which collision-free paths are obtained with RRT* [15], followed by smooth trajectory generation using quadratic programming(QP). However, collisions may occur as the smooth trajectory deviates from the original path. [2] proposed to add intermediate waypoints to pull the trajectory closer to the safe waypoint path. However, the iterative addition of waypoints leads to trajectories with higher control costs. It also gives no guarantee of how many additional waypoints are needed. Another waypoint-based approach, which relies on the heuristics of c-space expansion to prevent collision and is thus unable to provide a safety guarantee, was presented in [12].

Our approach, on the other hand, enjoys the same collision-free guarantee as the mixed-integer method proposed in [8], but runs orders of magnitudes faster (in fractions of a second compared to minutes). Instead of using convex regions to represent the free-space, we utilize efficient operations in an octree-based environment representation for online generation of a collision-free flight corridor. We use efficient quadratic programming to find the smooth trajectories with guaranteed safety. In fact, in confined environments, our method runs even faster than the waypoint-based method in [2], as shown in Sect. VI-A, due to the difficulty for RRT* to find a valid path through narrow corridors.

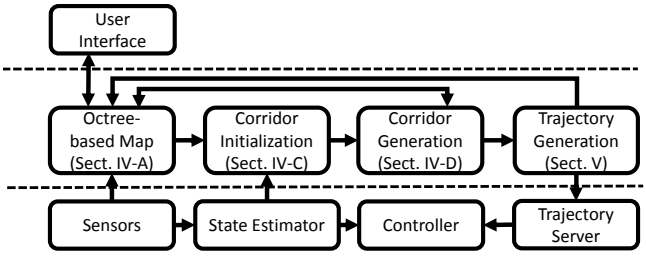


Fig. 2. A diagram of the overall system, indicating three system layers: 1) the user specifies the goal locations via a GUI. 2) our trajectory generator finds smooth and collision-free trajectories towards the goal and sends them to the trajectory server. 3) the trajectory server is continuously outputs desired states to the estimation and control modules to achieve autonomous flight.

III. Overview of Methodology

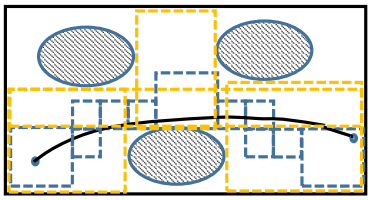
An overview of the full system pipeline is shown in Fig. 2. The 3-D environment is represented using a memory-efficient octree-based [16] structure (Sect. IV-A). Given a goal location, a search-based algorithm performs initialization of a virtual flight corridor that consists of a sequence of connected free-space 3-D grids (Sect. IV-C). These grids are inflated to a maximal volume flight corridor using efficient operations in the octree data structure (Sect. IV-D). A smooth trajectory that completely fits within the flight corridor is then generated using the approach to be described in Sect. V. Note that the 3-D grids may have different sizes, representing the different widths of the flight corridor. The trajectory generator is allowed to generate motions that go anywhere within the grid. This matches our intuition that the quadrotor should be given more freedom to maneuver within the free-space grid in order to reduce its control cost, rather than stick to the single shortest path. The whole pipeline consumes only fractions of a second onboard a quadrotor, allowing real-time reaction to obstacles in unknown environments.

IV. Virtual Flight Corridor Generation

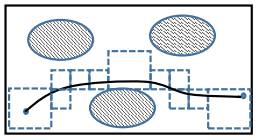
A. Octree-based Environment Representation

As obstacle information from onboard sensors streams in, the flight environment can be conveniently represented using the octree data structure [16]. Each node in the tree structure stands for a 3-D grid. The collection of variable-sized free-space grids forms the total free-space of the environment. Note that a large volume of free-space can be represented efficiently by a few large grids.

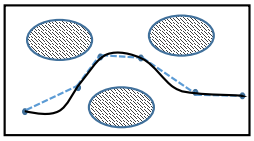
Suppose the map volume can be divided into a maximum of m finest-resolution grids. The depth of its tree structure is



(a) The proposed method



(b) Our previous method [17]



(c) Waypoint-based method [2]

Fig. 3. Our method generates collision-free trajectories that fit entirely within a large flight corridor (Fig. 3(a)). It shows improvements over our previous method [17], which is more conservative on corridor generation. It also out-performs a state-of-the-art method that seeds trajectory generation with RRT*-based path planning [2]. Performance comparison is presented in Sect. VI-A.

$h = O(log(m))$. To take the actual size of the flight system into account, we inflate each new obstacle point to a fixed size cubic volume as we insert it into the map. Utilizing the lazy principle of the octree structure [16], we achieve that updating a cubic volume has the complexity of $O(h)$.

B. Connectivity Graph for Corridor Search

We use the standard A^* algorithm for initialization of a flight corridor. This requires maintaining a connectivity graph $\mathcal{G}$ for all free-space grids in addition to the octree-based map. Every free-space grid will be associated with a vertex in $\mathcal{G}$. Undirected edges in $\mathcal{G}$ are used to model the connectivity between adjacent grids. As new obstacles are added into the map, the graph must be modified accordingly. We perform an update to the connectivity graph after a new batch of obstacle points is added to the map. For each insertion of obstacle points, $O(h)$ new grids may be created, and the occupancy value of the $O(h)$ existing grids may be modified. As a result, the number of graph vertices to be added or removed is bounded by $O(h)$.

Edges in the graph represent the connection between two adjacent free-space grids. As new vertices being added into the map, new edges have to be created. Without loss of generality, we can associate an edge to the smaller of the two connected grids. In the octree structure, a face of the smaller grid is entirely used for the connection. This suggests that, at each obstacle insertion, the number of edges increases linearly with respect to the number of newly added free-space grids, which is again bounded by $O(h)$.

In practice, we find that our approach is efficient enough for incremental mapping of the environment using a laser range finder with a mapping frequency of 5 Hz.

C. Search-based Flight Corridor Initialization

We denote the set of all free-space grids in the octree, which corresponds to the set of vertices in the connectivity graph $\mathcal{G}$, as $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \dots\}$. A grid is represented as:

$$\mathbf{c}_i = \begin{bmatrix} {}^{x_0}c_i & {}^{x_1}c_i \\ {}^{y_0}c_i & {}^{y_1}c_i \\ {}^{z_0}c_i & {}^{z_1}c_i \end{bmatrix}, \tag{1}$$

where ${}^{l_0}c_i$ and ${}^{l_1}c_i$ stand for the lower and upper boundaries respectively of the i^{th} grid on the l^{th} dimension. We use the standard A^* algorithm to find the minimum cost grid path from the starting grid to the goal grid. We set the edge costs as the Euclidean distances between the centers of grids. The heuristic function for the A^* search algorithm is also conveniently defined as the distances between the centers of grids to the goal location.

We are able to obtain a grid path $\mathcal{C}_p = (\mathbf{c}_1^p, \cdots, \mathbf{c}_M^p)$ consisting of sequentially connected free-space grids as the initial flight corridor. This initial flight will be inflated in the next step (Sect. IV-D) to provide larger flight spaces. As free-space can be efficiently represented by a small number of grids in the octree, the initialization of the flight corridor can be done online with very small computational load.

D. Maximal Volume Flight Corridor Generation

The initial flight corridor obtained using the search-based method (Sect. IV-C) guarantees safety, as demonstrated in our earlier work [17]. However, it does not represent the largest available free-space due to the deterministic grid partitioning in the octree data structure, which may cut large open spaces into small regions. To improve the performance, we develop a method to inflate every grid in the initial corridor to a locally maximal volume, while still ensuring that the corridor is collision-free. Our method utilizes efficient operations in the octree data structure. We define an obstacle indicator function $g : \mathbb{R}^{3\times2} \times \mathbb{R}^{3\times2} \times \mathbb{R}^1 \to \{0, 1\}$ as:

$$g(\mathbf{c}, \mathbf{d}, r) = \begin{cases} 0 & \text{if grid } \mathbf{c} + r \cdot \mathbf{d} \text{ is obstacle-free} \\ 1 & \text{otherwise,} \end{cases} \tag{2}$$

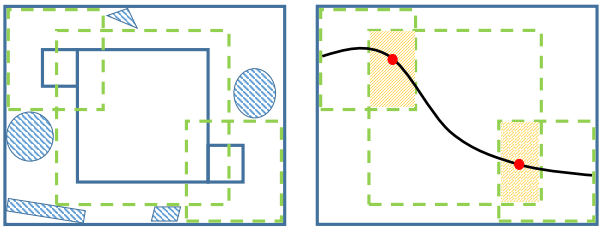
where $\mathbf{c} \in \mathbb{R}^{3\times2}$ represents the original cubic space of a corridor grid as in (1), $\mathbf{d} \in \mathbb{R}^{3\times2}$ is the inflation direction, and $r \in \mathbb{R}^1$ is the inflation distance.

Due to the simple fact that $g(\cdot)$ is non-decreasing in relation to inflation distance r, we can use binary search to determine the largest possible inflation distance r such that $\mathbf{c} + r \cdot \mathbf{d}$ is obstacle-free. The complexity of a single check as to whether a cubic volume is obstacle-free is $O(h)$ in an octree structure. The binary search iterates $O(log(\lceil \frac{r}{r_\epsilon} \rceil))$, where r_ϵ is the finest resolution of the map grid. The total complexity of finding the maximal obstacle-free volume given an inflation direction $\mathbf{d}$ is $O(h \cdot log(\lceil \frac{r}{r_\epsilon} \rceil))$.

We apply this binary search on each grid in the initial corridor using a sequence of different inflation directions. Firstly, each grid is inflated in all directions, by setting

$$\mathbf{d} = \begin{bmatrix} -1 & 1 \\ -1 & 1 \\ -1 & 1 \end{bmatrix}. \tag{3}$$

Then, we inflate the grid in the directions that connecting grids exist. Finally, the grid is expanded from each of its six faces.



(a) Maximal volume flight corridor shown as green dashed boxes (b) Adjustment of trajectory transition points within overlapping regions

Fig. 4. Using the method discussed in Sect. IV-D, we can efficiently improve the initial flight corridor (Sect. IV-C) into a better flight corridor with large overlapping volumes. The overlapping naturally allows adjustment of segment transition points for implicit time optimization (Sect. V-C).

In this way, small grids in the initial corridor can be converted into large overlapping volumes, as shown in Fig. 3(a) and Fig. 4. Overlapping volumes are beneficial for adjustment of segment transition points for implicit time optimization (Sect. V-C). We denote the inflated corridor consisting of overlapping and sequentially indexed grids as $\mathcal{C}_f = (\mathbf{c}_1^f, \cdots, \mathbf{c}_M^f)$.

V. Trajectory Generation with Corridor Constraints

In this section, we focus on generating safe trajectories that fit entirely in the flight corridor (Sect. IV) using an efficient quadratic programming approach. This method was initially proposed in our prior work [17], and we make improvements in implicit time optimization through adjustment of segment transition points. We follow the minimum-snap formulation for quadrotor trajectory generation [1], which shows that quadrotors enjoy the differential flatness property. This enables the use of piecewise polynomial trajectories to specify flat outputs of 3-D position and yaw angle $\{x, y, z, \psi\}$ and their derivatives for smooth quadrotor trajectory tracking [1, 2]. Due to the symmetric shape of a quadrotor, we focus on obtaining the translational motion for collision avoidance. The planning of the yaw angle and handling of the limited sensor field-of-view are left as future work.

A. Polynomial Trajectory Generation

The M-segment, N^{th} order piecewise polynomial trajectory will smoothly pass through the flight corridor $\mathcal{C}_f$ (Sect. IV-D). Given the trajectory's start-time and all segments' end-times $(t_0, t_1, t_2, \cdots, t_M)$, one dimension (out of $\{x, y, z\}$) of the whole trajectory can be written as:

$$f(t) = \begin{cases} \sum_{j=0}^{N} a_{1j}(t-t_0)^j & t_0 \le t \le t_1 \\ \sum_{j=0}^{N} a_{2j}(t-t_1)^j & t_1 \le t \le t_2 \\ \qquad\vdots & \\ \sum_{j=0}^{N} a_{Mj}(t-t_{M-1})^j & t_{M-1} \le t \le t_M, \end{cases} \tag{4}$$

where a_{ij} is the j^{th} order polynomial coefficient of the i^{th} segment. We aim to find a trajectory that minimizes the integral of the square of the N_ϕ^{th} derivative:

$$\min \int_{t_0}^{t_M} \left(\frac{d^{N_\phi} f(t)}{dt^{N_\phi}} \right)^2 dt. \tag{5}$$

The objective function (5) can be written as $\mathbf{p}^{\mathrm{T}}\mathbf{H}\mathbf{p}$, where $\mathbf{p} = [\mathbf{p}_1{}^{\mathrm{T}}, \cdots, \mathbf{p}_M{}^{\mathrm{T}}]^{\mathrm{T}}$ is the vector of the polynomial coefficients of all segments and $\mathbf{H}$ is the Hessian matrix. We omit its construction for brevity. We will show that both corridor (Sect. V-B) and dynamical (Sect. V-D) constraints can be formulated as either linear equality ($\mathbf{A}_{eq}\mathbf{p} = \mathbf{b}_{eq}$) or inequality ($\mathbf{A}_{lq}\mathbf{p} \le \mathbf{b}_{lq}$) constraints. As a result, the trajectory generation problem can be written as a quadratic programming problem (QP):

$$\begin{aligned} \min \quad & \mathbf{p}^{\mathrm{T}}\mathbf{H}\mathbf{p} \\ \text{s.t.} \quad & \mathbf{A}_{eq}\mathbf{p} = \mathbf{b}_{eq} \\ & \mathbf{A}_{lq}\mathbf{p} \le \mathbf{b}_{lq}. \end{aligned} \tag{6}$$

To ensure a smooth transition between trajectory segments, we enforce continuity on the first $N_\phi - 1$ derivatives at the segment-transition times $(t_1, \cdots, t_{M-1})$.

In practice, small numerical errors in higher order coefficients may result in large changes to the trajectory. We therefore utilize the method proposed in Richter *et al.*'s [2] to handle this issue, in which segment endpoint derivatives are computed, as opposed to the polynomial coefficients, to handle this issue.

B. Enforcing Corridor Constraints

The transition between the i^{th} and $(i+1)^{th}$ grids in $\mathcal{C}_f$ should happen at time t_i within the overlapping regions of grid i and grid $i+1$. Mathematically, we can write the linear inequality constraints ($\mathbf{A}_{lq}\mathbf{p} \le \mathbf{b}_{lq}$) for ensuring a safe transition as:

$$\begin{cases} \max({}^{l_0}c_i^f, {}^{l_0}c_{i+1}^f) \le f_i(t_i) \le \min({}^{l_1}c_i^f, {}^{l_1}c_{i+1}^f) \\ f_{i+1}(t_i) = f_i(t_i), \end{cases} \tag{7}$$

where $l \in \{x, y, z\}$, and $f_i(\cdot)$ is the i^{th} polynomial segment of the trajectory. An illustration of this type of constraint can be seen in Fig. 4(b).

However, (7) only ensures that at the time of grid transition (t_i), the trajectory stays within the corridor boundary. It does not provide any guarantee on the safety of the trajectory at any other time. To achieve such a guarantee, we propose a novel algorithm to enforce the grid boundary for the entire trajectory by considering extrema of the polynomial. We first solve the QP in (6) with only grid transition constraints (7) and keep the initial solution. Consider the i^{th} trajectory segment at the k^{th} iteration. We check, for each dimension $l \in \{x, y, z\}$, whether any of the $N-1$ extrema of the N^{th} order polynomial violate the grid boundary constraints (Fig. 5). We add constraint points at the violating extrema $f_i^k(t_{jk}^e)$ with margin δ_p for solving the QP in the $(k+1)^{th}$ iteration:

$$\begin{cases} f_i^{k+1}(t_{jk}^e) \le^{l_1} c_i^f - \delta_p & \text{if } f_i^k(t_{jk}^e) >^{l_1} c_i^f \\ f_i^{k+1}(t_{jk}^e) \ge^{l_0} c_i^f + \delta_p & \text{if } f_i^k(t_{jk}^e) <^{l_0} c_i^f, \end{cases} \tag{8}$$

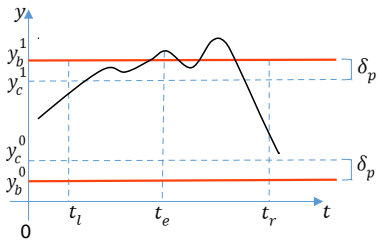


Fig. 5. Illustration of a grid boundary violation case. The extrema are used to check the boundary violation. Applying a finite number of constraint points, the trajectory can be bounded with margin δ_p (Sect. V-B).

where $f_i^k(\cdot)$ is the i^{th} polynomial segment through the i^{th} grid, t_{jk}^e is the time of the j^{th} boundary violation, all measured at the k^{th} iteration and δ_p is a constant position margin. These constraint points can be written in linear inequality form ($\mathbf{A}_{lq}\mathbf{p} \leq \mathbf{b}_{lq}$).

Note that extrema can only be checked after polynomial coefficients are found, and extrema in the current iteration may not be extrema in the next iteration. Nevertheless, the algorithm iterates by adding more constraint points to the QP until no extrema in the polynomial violates the grid boundary constraints. We show in Theorem 1 that only a finite number of constraint points are needed to enforce grid boundary constraints as far as velocities are bounded. This saves a significant amount of computational power compared to the dense constraint points method used in [1].

Theorem 1: A polynomial trajectory that fits within the grid boundary can be generated with a finite number of constraint points along the boundary with margin δ_p if the derivative of the trajectory is bounded.

Proof: For brevity, we only consider the violation of the upper boundary case, but this can be easily generalized to the lower bound violation case. Let $f : t \to y$ be the N^{th} order scalar polynomial that specifies one dimension of the trajectory segment between $[0,\ t_f]$. The upper boundary of this segment is y_b^1. A constraint point can be written as $y_c^1 = y_b^1 - \delta_p$, where δ_p is a positive margin. Since $f(0)$ and $f(t_f)$ are bounded within y_b^1 through (7), and the derivative of $f(t)$ is also bounded as $\dot{y}_m$ such that $\forall t^* \in [0,\ t_f],\ \|\frac{df}{dt}|_{t=t^*}\| \leq \dot{y}_m$.

Consider a boundary violation case (Fig. 5) in which the trajectory breaks the boundary y_b^1 between two adjacent existing constraint points, (t_l, y_c^1) and (t_r, y_c^1). It must satisfy that $f(t_l) \leq y_c^1$, $f(t_r) \leq y_c^1$, and $f(t_e) \geq y_b^1$, where $f(t_e)$ is one of the extrema of $f(t)$ that fall within the interval $[t_l,\ t_r]$. According to the *Mean Value Theorem*, $\exists t^* \in [t_l, t_e]$ such that $\frac{df}{dt}|_{t=t^*} = \frac{f(t_e)-f(t_l)}{t_e-t_l} \geq \frac{y_b^1-y_c^1}{t_e-t_l} = \frac{\delta_p}{t_e-t_l}$. Since the derivative is bounded, we have

$$\dot{y}_m \geq \frac{df}{dy}|_{t=t^*} \geq \frac{\delta_p}{t_e - t_l}, \tag{9}$$

which suggests that

$$t_e - t_l \geq \frac{\delta_p}{\dot{y}_m}. \tag{10}$$

Similarly, we can show that $t_r - t_e \geq \frac{\delta_p}{\dot{y}_m}$, and:

$$t_r - t_l = (t_r - t_e) + (t_e - t_l) \geq 2 \cdot \frac{\delta_p}{\dot{y}_m}. \tag{11}$$

As shown above, a grid boundary violation can only happen if two adjacent constraint points are separated by at least $2 \cdot \frac{\delta_p}{\dot{y}_m}$. Therefore, we only need to add at most $\lceil \Delta t/(2{\cdot}\delta_p/\dot{y}_m) \rceil$ constraint points to completely bound the trajectory within the boundary. ■

According to (11), any extrema that breaks the grid boundary will be at least $\delta_p/\dot{y}_m$ away from existing constraint points. In our approach, we iteratively add extrema that violate the grid boundary as constraint points. Therefore, all constraint points are separated by at least $\delta_p/\dot{y}_m$. As suggested in Theorem 1, we will eventually have sufficient constraint points to eliminate all grid boundary violations. In practice, we find our iterative approach very efficient, and almost all boundary violation cases are eliminated within a few iterations, with far fewer constraint points than the worst case scenario shown in Theorem 1.

C. Effects on Adjustment of Segment Transition Points

As shown in (7) and Fig. 4(b), the transition between polynomial segments can happen anywhere within the overlapping regions between two adjcent corridor grids. In fact, as illustrated in Fig. 3(a) and shown in almost all our experimental results (Sect. VI), our corridor generation step (Sect. IV-D) produces large free-space regions with large overlaps. In this way, the changes to segment length within the overlapping regions serve as an implicit time optimization. Our largely overlapped corridor grids allow nontrivial adjustment to reduce the cost of the trajectory. Our method is embedded into the QP and introduces no extra computation. This is advantageous to the iterative time optimization methods proposed in [1] and [2].

D. Enforcing High-Order Dynamical Constraints

The method presented in Sect V-B can also be utilized to bound higher-order derivatives of the trajectory to predefined maximum values to ensure the dynamic feasibility of the trajectory for the quadrotor. For instance, the velocity while flying through a grid can be bounded by incrementally adding constraints on the boundary violating extrema on the velocity profile. It is also possible to bound higher-order derivatives if necessary. All dynamical constraints can be written in the linear inequality form ($\mathbf{A}_{lq}\mathbf{p} \leq \mathbf{b}_{lq}$). Note that the N^{th} order derivative of the polynomial will be a constant and is definitely bounded. As such, the $(N-1)^{th}$ to 0^{th} order derivatives can be bounded from the bottom up. This supports the bounded derivative requirement in Theorem 1.

VI. Results

Two simulation and two real-world flight experiments are presented to validate our approach. Our trajectory generation method is implemented in C++11 using a g++ compiler with the -O3 optimization option. Additionally, we use the sparse linear algebra library in Eigen and an open source quadratic programming solver, OOQP (Object Oriented software for

Quadratic Programming) [18]. The boundary margins δ_p, δ_v and δ_a are set as $0.005\,\text{m}$, $0.05\,\text{m/s}$, and $0.05\,\text{m/s}^2$ respectively.

A. Comparison with State-of-the-Art Methods

We implement the state-of-the-art online trajectory generation method in [2] and compare it with the proposed method in simulation. In [2], the trajectory generation was seeded by waypoints from RRT^*-based planning. The comparison is done on a desktop computer with a 3.20 GHz Intel Core i5-4570 CPU. The RRT^* module in the Open Motion Planning Library (OMPL)[1] is used. We set the the RRT step as $0.2m$. The map size is $40 \times 40 \times 10\,\text{m}^3$. Our octree-based map has the finest grid resolution of $0.2 \times 0.2 \times 0.2\,\text{m}^3$. We randomly generated 160 virtual pillars for the map and inflate them to get the actual configuration space. The maximum velocity and linear acceleration are $3.0\,\text{m/s}$ and $3.0\,\text{m/s}^2$ respectively for all methods. These values are also used for computing the segment duration following a simple accelerate-cruising-decelerate heuristic. We also bound these dynamical constraints in our method (Sect. V-D). Although the obstacles are generated in a 2-D manner, the methods work in the 3-D configuration space to produce 3-D trajectories.

Fig. 6 illustrates results from four simulations using the three trajectory generation methods. The figure has $1.0\,\text{m}^2$ cells for visual reference. Blue cubes are the actual obstacles, while transparent white ones are inflated obstacles. The green trajectory is generated by Richter *et al.*'s method [2]. , the blue trajectory is from our previous method [17] that does not have corridor grid inflation, and the red trajectory is from the proposed method. The initial flight corridor is shown as light green transparent cubes, while the final flight corridor is shown in white. The lengths of the trajectories generated by all methods are approximately 26, 28, 46, and 64 meters in the four simulations.

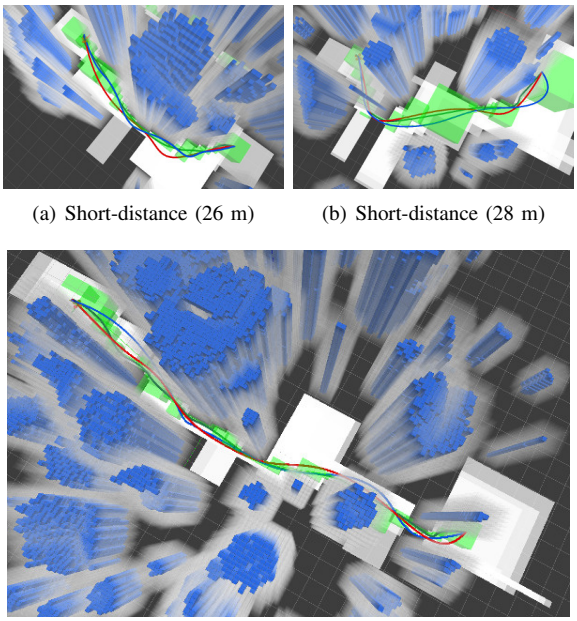
As shown in Fig. 7, the RRT^* takes lots of time in Richter *et al.*'s method [2] to find valid paths in the confined environment, and many of additional constraint points are needed to ensure that the trajectory is collision-free. Our method, on the other hand, requires much less time for computation, and guarantees safety by construction. Additionally, as shown in Table I, our method generates trajectories with much lower

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

TABLE I

COSTS OF TRAJECTORIES GENERATED BY THREE METHODS (SECT. VI-A).

method \ simulation	1	2	3	4
C.Richter, *et al* [2]	0.6246	1.0897	8.8087	99.1344
Our previous method [17]	5.8910	7.0701	7.8875	5.2311
The proposed method	0.4626	0.4186	0.3146	0.4677



(a) Short-distance (26 m) (b) Short-distance (28 m)

(c) Medium-distance (46 m)

(d) Long-distance (64 m)

Fig. 6. Screenshots from four simulations, illustrating the comparison between our methods and the state-of-art method from [2]. Detailed explanation of the figure can be found in Sect. VI-A.

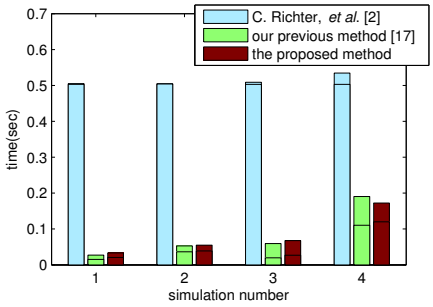


Fig. 7. The computing times for the three methods. For Richter *et al.*'s method [2], the horizontal line separates the times for RRT^* (lower) and trajectory generation (upper). For both of our methods, the horizontal line separates the times for corridor generation (lower) and trajectory generation (upper).

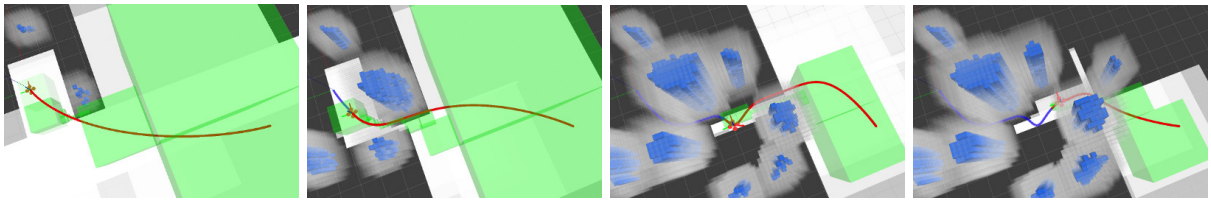


Fig. 8. A simulated quadrotor navigates through unknown environments with limited sensing. As described in Sect. VI-B, frequent trajectory re-generation occurs in order to avoid newly discovered obstacles. The red line is the generated trajectory, the blue line is the path already flown, the blue and transparent white cubes are original and inflated obstacles, respectively. , and the green cubes represent the initial corridor (Sect. IV-C). White patches represent the final inflated flight corridor (Sect. IV-D).

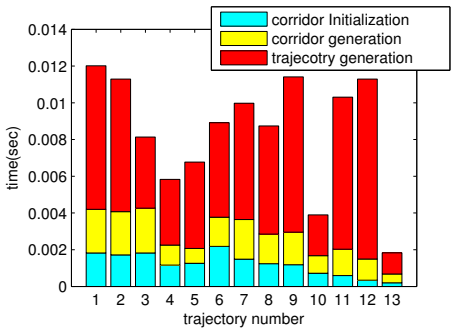


Fig. 9. The computing times for trajectory re-generation in the simulated experiment shown in Fig 8. The simulation runs on the same computer as mentioned in Sect. VI-A).

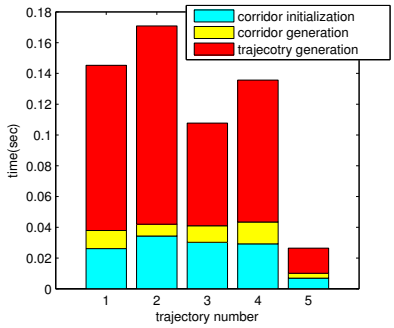


Fig. 10. The computing times for five trajectories' re-generations during the indoor flight experiment (Sect. VI-C).

cost (defined in (5)). Even without corridor inflation, as in our previous method [17], the trajectory cost outperforms Richter *et al.*'s method in [2]. With the complete corridor generation pipeline (Sect. IV) and segment duration adjustment (Sect. V-C), the proposed method enjoys dramatic reduction of the trajectory cost, with very competitive online performance.

B. Simulated Flight in Unknown Environments

We present results of a simulated quadrotor navigating through unknown environments. In this simulation, we set a limit on the sensing range to be 3 m. Obstacles can only be discovered as the quadrotor gets close to them. The generated trajectory is constantly checked for possible collision. A new trajectory will be generated if there are intersections of the current trajectory with any of the obstacles. Thirteen re-generations occurred during the experiment, with a few examples shown in Fig. 8. The computing time for each trajectory generation is shown in Fig. 9.

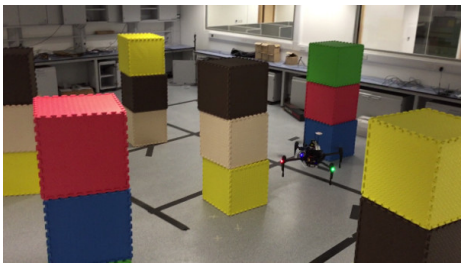


Fig. 11. A snapshot of the autonomous indoor flight experiment.

C. Autonomous Flight in Cluttered Indoor Environments

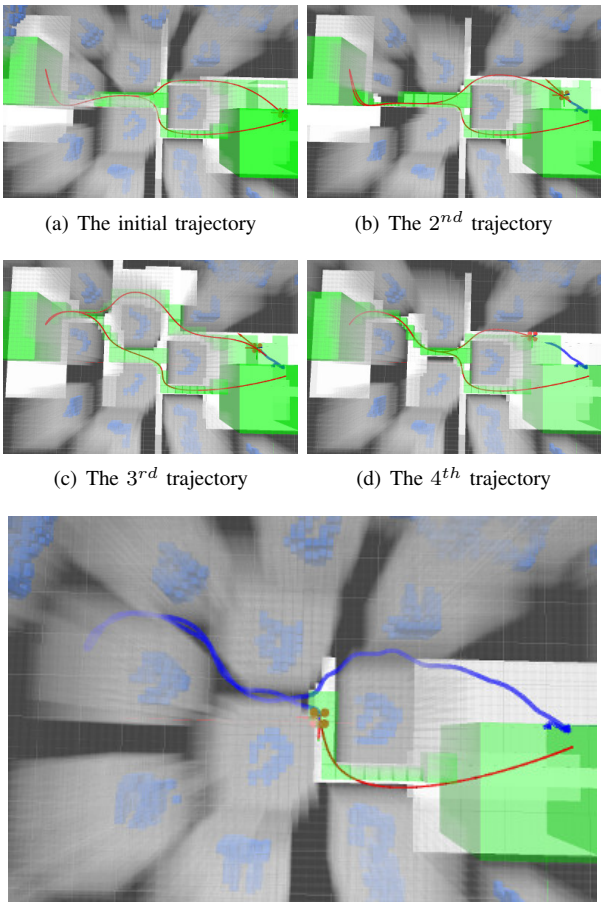
We present autonomous navigation of a quadrotor in cluttered unknown indoor environments with all computations done onboard. Our experimental testbed is as described in Fig. 1. We set 3 m/s and 2 m/s^2 as the dynamical constraints of the platform Fig. 11 shows the actual testing environment. Obstacles are detected using the onboard laser range finder. Scans are down-sampled to an average of 200 points for each scan, and then inserted into the map at 5 Hz. The average map update time for each scan is 0.014 sec. And the maximum flight velocity is $1.4m/s$. As shown in Figs. 10, 12 and 13, five trajectory re-generations occurred during the experiment, with each of them taking less than 20 ms. This suggests that our approach is suitable for real-time obstacle avoidance in unknown environments.

D. Autonomous Navigation in Outdoor Environments

We also test our approach in an autonomous outdoor flight experiment, as shown in Fig. 1. All system parameters are the same as in the indoor experiment (Sect. VI-C), and the velocity reaches $1.8m/s$. No trajectory re-generation occurs in this experiment due to the fact that all obstacles can be detected by a single laser scan. It takes only 0.003, 0.013 and 0.051 seconds respectively for corridor initialization (Sect. IV-C), corridor generation (Sect. IV-D), and trajectory generation (Sect. V).

VII. Conclusion and Future Work

We present an online method for generating collision-free trajectories for autonomous quadrotor flight through cluttered environments. Our method computes collision-free trajectories in fractions of a second on a moderately fast computer, allowing real-time operations in unknown environments. We build a complete quadrotor testbed with onboard sensing, state estimation, mapping, and control, and



(a) The initial trajectory (b) The 2^{nd} trajectory

(c) The 3^{rd} trajectory (d) The 4^{th} trajectory

(e) The final trajectory

Fig. 12. Autonomous flight in cluttered unknown indoor environments (Sect. VI-C). Visualization information can be interpreted in the same way as in Fig. 8. Trajectory re-generation occurs after detection of new obstacles.

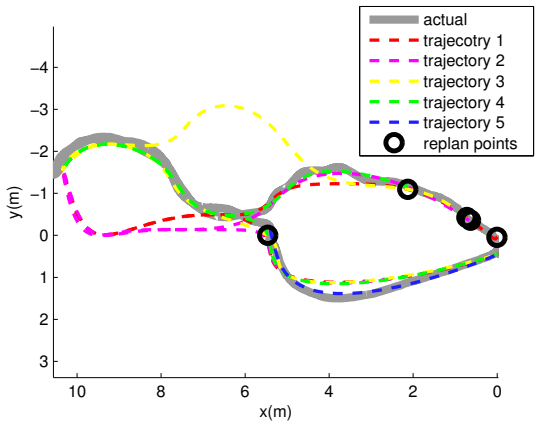


Fig. 13. Desired trajectories (colored) and the actual flight path (grey) for the indoor flight experiment (Sect. VI-C). Locations where trajectories are re-generated are marked.

integrate the proposed method to show online navigation through complex unknown environments. Our method is also compared with state-of-the-art methods and demonstrates superior performance. Extensions to this work may include large-scale experiments at high speed. We are also interested in the problems of online tracking and avoidance of moving objects.

REFERENCES

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

[2] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Proc. Intl. Sym. of Robot. Research*, Singapore, Dec. 2013.

[3] S. Tom, D. M. Bart, F. Eric, and H. Jonathan, "Mixed integer programming for multi-vehicle path planning," in *Euro. Control Conf.*, vol. 1, Porto, Portugal, 2001, pp. 2603–2608.

[4] R. Arthur, B. John, T. Michael, and H. Jonathan, "Coordination and control of multiple UAVs," in *AIAA Guidance, Navigation, and Control Conf.*, Monterey, CA, USA, Aug. 2002.

[5] C. K. Forbes, "Online trajectory planning for UAVs using mixed integer linear programming," Ph.D. dissertation, Massachusetts Institute of Technology, 2006.

[6] H. Yongxing, D. Asad, and M. Ali, "Differential flatness-based trajectory planning for multiple unmanned aerial vehicles using mixed-integer linear programming," in *Proc. of the American Control Conf.*, vol. 1, Portland, Oregon, June 2005, p. 104.

[7] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Proc. IEEE Intl. Conf. on Robot. and Autom.*, Saint Paul, MN, May 2012, pp. 477–483.

[8] D. Robin and T. Russ, "Efficient mixed-integer planning for UAVs in cluttered environments," in *Proc. IEEE Intl. Conf. on Robot. and Autom.* Seattle, Washington, USA: IEEE, May 2015, pp. 42–49.

[9] ——, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, vol. 107, pp. 109–124.

[10] H. Alvarez, L. Paz, J. Sturm, and D. Cremers, "Collision avoidance for quadrotors with a monocular camera," in *Proc. Intl. Sym. on Exp. Robot.*, Marrakech/Essaouira, Morocco, June 2014.

[11] C. Bills, J. Chen, and A. Saxena, "Autonomous MAV flight in indoor environments using single image perspective cues," in *Proc. IEEE Intl. Conf. on Robot. and Autom.*, Shanghai, China, May 2011, pp. 5776–5783.

[12] M. Shomin and R. Hollis, "Fast, dynamic trajectory planning for a dynamically stable mobile robot," in *Proc. IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*, Chicago, IL, Sept 2014, pp. 3636–3641.

[13] A. Boeuf, J. Cortes, R. Alami, and T. Simeon, "Planning agile motions for quadrotors in constrained environments," in *Proc. IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*, Chicago, IL, Sept 2014, pp. 218–223.

[14] J. van den Berg, D. Wilkie, S. J. Guy, M. Niethammer, and D. Manocha, "LQG-Obstacles: Feedback control with collision avoidance for mobile robots with motion and sensing uncertainty," in *Proc. IEEE Intl. Conf. on Robot. and Autom.*, Saint Paul, MN, May 2012, pp. 346–353.

[15] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Proc. Robot.: Sci. and Syst.*, Zaragoza, Spain, June 2010.

[16] M. D. Berg, V. K. Marc, O. Mark, and S. O. Cheong, "Computational geometry," 2000.

[17] J. Chen, K. Su, and S. Shen, "Real-time safe trajectory generation for quadrotor flight in cluttered environments," in *Proc. of the IEEE Intl. Conf. on Robot. and Bio.*, Zhuhai, China, Aug. 2015.

[18] E. M. Gertz and S. J. Wright, "Object-oriented software for quadratic programming," *ACM Trans. on Math. Softw.*, vol. 29, pp. 58–81, 2003.