Proceedings of the 2015
IEEE Conference on Robotics and Biomimetics
Zhuhai, China, December 6-9, 2015

Real-Time Safe Trajectory Generation for Quadrotor Flight in Cluttered Environments

Jing Chen, Kunyue Su, and Shaojie Shen

***Abstract*— We address the problem of real-time generation of smooth and collision-free trajectories for autonomous flight of a quadrotor through 3-D cluttered environments. Our approach starts by generating a sequence of variable sized 3-D free space grids as the virtual flight corridor using an OctoMap-based environment representation and search-based planning method. The key contribution is a quadratic programming-based formulation for generating multi-segment polynomial trajectories that are entirely fit within the corridor, and thus collision-free. Our formulation also allows incorporating higher-order dynamical constraints to ensure that the trajectory is feasible for the platform. A novel non-iterative constraint relaxation method is also proposed for implicit optimization of segment duration. The proposed algorithm runs real-time onboard our quadrotor experimental testbed. Both simulation and online experimental results are presented for performance verification.**

I. Introduction

Multi-rotor aerial vehicles, in particular quadrotors, are becoming increasingly popular over the past few years. These platforms are ideal for missions in cluttered environments due to their small size and superior mobility. Recently, there have been major advances in estimation [1], planning [2], and control [3] towards fully autonomous operations of quadrotors in complex 3-D environments. Nevertheless, there are still significant challenges in having quadrotors rapidly fly through cluttered environments due to the inherently fast dynamics, the complexity of the environment, and limited computational resources. In this paper, we address the problem of real-time generation of smooth, collision-free, and dynamically feasible trajectories for autonomous flight of a quadrotor in cluttered 3-D environments.

The literature on robotic motion planning is extensive. For kinematics of ground robots, both search-based and randomized algorithms give promising results. However, for higher-order dynamical systems such as quadrotors, these approaches suffer significant computational cost due to the high-order continuity requirements which translate into the high-dimensional planning space. The problem gets even more challenging when we consider quadrotors operating in cluttered environments with a large number of non-convex obstacles. While a randomized algorithm [4] may be able to handle such cases by simulating the vehicle dynamics while steering between two states. Its high computational cost prohibits real-time operations. Local [5] and reactive [6] methods are also widely studied for quadrotor obstacle avoidance, but with the downside of lacking global guarantee.

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, ksu@connect.ust.hk, eeshaojie@ust.hk

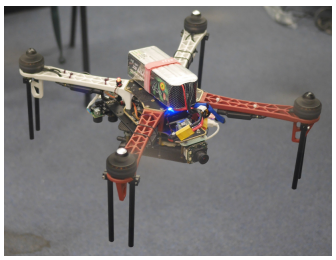


Fig. 1. Our quadrotor experimental platform that is based on the DJI F450 frame and is equipped with a PixHawk flight controller and an Intel NUC computer. Onboard sensors include an sonar, a downward-facing, and a forward facing cameras. A complete vision-based navigation and tag-based obstacle detection solution enables autonomous indoor flight experiment.

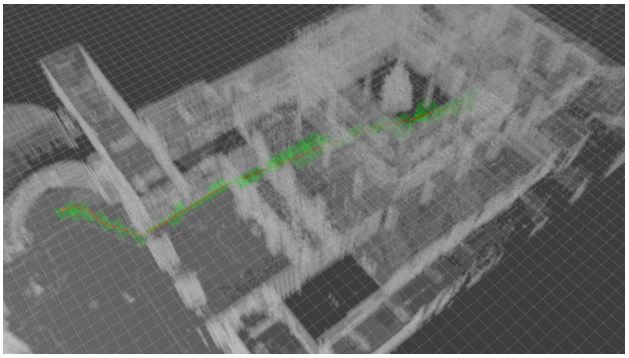


Fig. 2. Trajectory generation on the TUM NAVVIS dataset. The volume of the environment is $78m \times 50m \times 20m$ with 259,237 occupied grids at the resolution of $0.20m \times 0.20m \times 0.20m$. A trajectory with length $42.201m$ is computed in 0.267 seconds. The corridor is shown as green cubes, and the final trajectory is shown in red. The computation time includes both corridor generation (Sect. III) and trajectory generation (Sect. IV).

To achieve global trajectory planning with reasonable computational complexity, two-step approaches, where trajectory generation is seeded by position constraints, are widely favorable. In the first step, spatial waypoints are specified either by the user [2], or via search-based [7] or randomized [8] algorithms. In the second step, the waypoint path is converted into time parameterized trajectories or control inputs. [9] presents the LQG-Obstacle algorithm for collision-free steering between waypoints. However, this algorithm is unsuitable for real-time use due to the high computational cost for frequent obstacle checking. [10] decomposes the local trajectory into multiple phases and designs flight behaviors for each phase. Utilizing the differential flatness property of a quadrotor,

978-1-4673-9675-2/15/$31.00 © 2015 IEEE

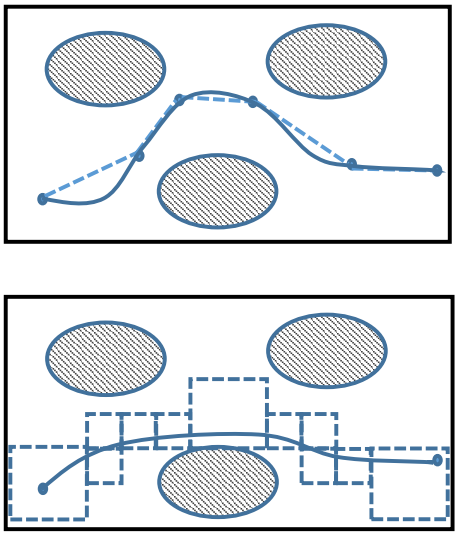


Fig. 3. Comparison between existing approaches that use smooth trajectories to connect waypoints as in [8] (top) and our approach that operates on a virtual flight corridor (bottom). Our approach is able to generate trajectories with lower control cost and guarantee collision avoidance.

[2, 11] convert predefined waypoints into polynomial trajectories using quadratic programming and encode obstacle information as a dense set of constraint points [2] or mixed-integer constraints [11]. But both approaches demand high computational power. In [8], a collision-free waypoint path that is obtained with RRT* [12] is converted into a smooth polynomial trajectory. However, as the smooth trajectory deviates from the original path, collisions may occur. [8] proposed to add intermediate waypoints to pull the trajectory closer to the waypoint path to prevent collision. In fact, the iterative addition of waypoints leads to trajectories with higher control costs. It also has no guarantee on how many additional waypoints are needed. Another two-step approach presented in [7], which relies on the heuristics of c-space expansion to prevent collision, is unable to provide any avoidance guarantee.

Our approach also consists of two steps, but it is fundamentally different from existing approaches. As shown in Fig. 3, our approach eliminates the concept of waypoints and uses a virtual corridor to represent position constraints for the quadrotor. The corridor, which approximates the flight region, is obtained by a search-based method that operates on an OctoMap-based [13] environment representation. Thanks to the efficient representation of free space in the OctoMap, the generation of this corridor can be done in real-time even with the standard A* algorithm. Our key contribution is a quadratic programming-based formulation for real-time generation of polynomial trajectories that fit entirely within the corridor. Unlike hard waypoint constraints as used in [8], our formulation allows the solver to optimize the trajectory within the corridor obtaining a smoother trajectory with lower control cost. Higher order dynamical constraints can also be incorporated in a similar manner to ensure the feasibility of the trajectory. Our formulation also naturally enables a fast and non-iterative constraint relaxation method for implicit trajectory time adjustment. Compared to iterative time allocation methods [2, 8], our method introduces almost no extra computational cost, making it useful for real-time applications.

To this end, we note that our approach does not aim to solve for the globally optimal trajectory. Instead, our corridor generation step gives a reasonable approximation of the flight regions with very small computational cost. The trajectory generator provides the optimal trajectories subject to the corridor and higher-order dynamical constraints with moderate computational load. We aim to achieve a good trade-off between computational complexity and optimality of the trajectory, while ensuring the safety of the vehicle.

Next, we give an overview of our methodology in Sect. II. Sect. III discusses the environment representation and the generation of the virtual flight corridor. Sect. IV presents the generation of trajectories that satisfy corridor and dynamical constraints. Simulation and online experimental results are presented in Sect. VI, followed by a conclusion in Sect. VII.

II. Overview of Methodology

An overview of the full system pipeline is shown in Fig. 4. We use the memory-efficient OctoMap [13] structure to represent both the free and occupied space of the 3-D environment. Given the start and a goal location, a graph search-based algorithm finds a virtual flight corridor that consists of a sequence of connected free-space 3-D voxel grids.(Sect. III). A smooth trajectory that completely fits in the flight corridor is then generated using the approach to be presented in Sect. IV. Note that 3-D voxel grids may have different sizes, and 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 maneuverer within the free-space grid in order to reduce its control cost, rather than stick with the waypoint-based shortest path.

III. Virtual Flight Corridor Generation

A. OctoMap-Based Environment Representation

As obstacle information from onboard sensors streams in, the flight environment can be conveniently represented using the OctoMap. The OctoMap is stored in the octree structure with additional probabilistic update rules for marking of free-space and occupied-space. 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. To take the size of the quadrotor into account, we inflate the obstacles to obtain free-space grids that match the actual configuration space of the flight system.

B. Search-Based Flight Corridor Generation

Given the free-space information from the OctoMap, We build a discrete graph from the OctoMap structure. The

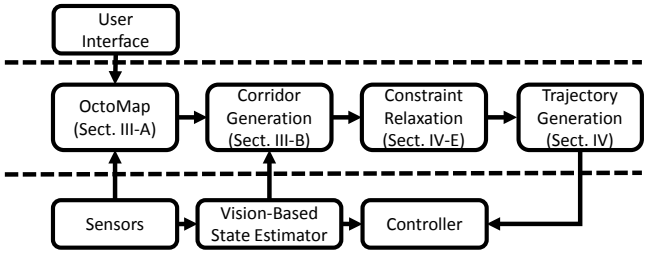


Fig. 4. A diagram of the overall system, indicating three control levels. The user input goals via a GUI interface. Our trajectory generation method finds a smooth collision-free trajectory towards the goal. The desired trajectory is sent to the lowest level of estimation and control modules to achieve autonomous flight.

rectangular common surface between two connecting grids, which we called windows, are considered as vertexes in the discrete graph. Naturally, we consider the connection between windows within a grid as edges in the graph. We denote grids in the OctoMap as $\mathcal{C}$ and windows as $\mathcal{W}$. we have $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \dots\}$ and $\mathcal{W} = \{\mathbf{w}_1, \mathbf{w}_2, \dots\}$, where

$$\begin{aligned} \mathbf{c}_i &= [c_i^{x_0}, c_i^{x_1}, c_i^{y_0}, c_i^{y_1}, c_i^{z_0}, c_i^{z_1}]^T \\ \mathbf{w}_i &= [w_i^{x_0}, w_i^{x_1}, w_i^{y_0}, w_i^{y_1}, w_i^{z_0}, w_i^{z_1}]^T \end{aligned} \tag{1}$$

and $c_i^{l_0}$ and $c_i^{l_1}$ stand for the upper and the lower boundaries of the i^{th} grid on the l^{th} dimension. $w_i^{l_0}$ and $w_i^{l_1}$ follow the similar conversion. Note that the upper and lower boundaries of one dimension of $\mathbf{w}_i$ are the same as a window has no thickness.

Given the start grid and the goal grid, the A^* algorithm is applied to find the minimum cost grid path $(\mathcal{C}_p, \mathcal{W}_p)$. We set the edge cost, which is the cost between windows on the same grid, proportional to the distance between window centers. Each edge represents one possible route to pass through a grid. The heuristic function for the A^* search algorithm is defined as

$$\mathrm{h}(\mathbf{w}_i) = \left\| \frac{1}{2} \begin{bmatrix} w_i^{x_0} + w_i^{x_1} \\ w_i^{y_0} + w_i^{y_1} \\ w_i^{z_0} + w_i^{y_1} \end{bmatrix} - \begin{bmatrix} x_g \\ y_g \\ z_g \end{bmatrix} \right\|, \tag{2}$$

We are able to obtain the grid path $(\mathcal{C}_p, \mathcal{W}_p)$ with M connecting free-space grids and a chain of windows modeling the connection between grids. We call this grid path as the virtual flight corridor. As free-space can be efficiently represented by a small number of grids in the OctoMap, the generation of this corridor can be done in real-time even with the standard A* algorithm.

IV. Trajectory Generation with Corridor Constraints

A. Quadrotor Dynamical Model and Differential Flatness

The equation of motion for a quadrotor can be written as:

$$\begin{aligned} m\ddot{\mathbf{r}} &= -mg\mathbf{z}_W + f\mathbf{z}_B \\ \mathbf{M} &= \mathbf{J}\dot{\mathbf{\Omega}} + \mathbf{\Omega} \times \mathbf{J}\mathbf{\Omega} \end{aligned}, \tag{3}$$

where $\mathbf{r}$ is the position of the quadrotor, $\mathbf{z}_W$, $\mathbf{z}_B$ are vertical axes in the world and the body frame, respectively. f and $\mathbf{M}$ are the total thrust and moment from all propellers, $\mathbf{\Omega}$ is the body angular velocity, and $\mathbf{J}$ is the inertial matrix.

As shown in [2], quadrotors enjoy the differential flatness property. The states and the control input for a quadrotor can be specified by four flat outputs and their derivatives. This enables the use of piecewise polynomial trajectories to specify flat outputs $\{x, y, z, \psi\}$ and their derivatives for smooth quadrotor trajectory tracking [2, 8]. Here $\{x, y, z\}$ is the position and ψ is the yaw angle. 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 the limited sensor field of view are left as future work.

We use the non-linear control method introduced in [2] to compute the moment $\mathbf{M}_b$ and the thrust f:

$$\begin{aligned} f &= \mathbf{z}_b^T[-k_x\mathbf{e}_\mathbf{r} - k_v\mathbf{e}_{\dot{\mathbf{r}}} + m\ddot{\mathbf{r}}_d + mg\mathbf{z}_w] \\ \mathbf{M}_b &= -k_R\mathbf{e}_R - k_\omega\mathbf{e}_\omega \end{aligned}, \tag{4}$$

where $\mathbf{e}_\mathbf{r}$, $\mathbf{e}_{\dot{\mathbf{r}}}$, $\mathbf{e}_R$, and $\mathbf{e}_\omega$ are errors in position, velocity, orientation, and angular velocity respectively. It is straightforward to convert from $\mathbf{M}_b$ and f to the motor speed.

B. Polynomial Trajectory Generation

The multi-segment N^{th} order polynomial trajectory will smoothly pass through the flight corridor. Each segment corresponds to a complete pass through a grid. Given the trajectory start time and all segment 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:

$$f(t) = \begin{cases} \sum_{j=0}^{N} a_{1j}(t - t_0)^j & t_0 \le t \le t_1 \\ \sum_{j=0}^{N} a_{2j}(t - t_1)^j & t_1 \le t \le t_2 \\ \vdots & \\ \sum_{j=0}^{N} a_{Mj}(t - t_{M-1})^j & t_{M-1} \le t \le t_M \end{cases} \tag{5}$$

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:

$$\min \int_{t_0}^{t_M} \left(\frac{d^{N_\phi} f(t)}{dt^{N_\phi}} \right)^2 dt, \tag{6}$$

Similar to [1], we set $N_\phi = 3$ and minimize the total jerk of the polynomial trajectory in order to minimize the angular velocity while flying. This property is essential for our vision-based quadrotor platform (Fig. 1) as fast rotation is harmful for vision-based state estimation.

The objective function (6) 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. $\mathbf{H}$ is the Hessian matrix. We omitted its construction for brevity. We will show that both corridor (Sect. IV-C) and dynamical (Sect. IV-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 (QP) problem:

$$\begin{aligned} \min \quad & \mathbf{p}^{\mathrm{T}}\mathbf{H}\mathbf{p} \\ \text{s.t.} \quad & \mathbf{A}_{eq}\mathbf{p} = \mathbf{b}_{eq} \\ & \mathbf{A}_{lq}\mathbf{p} \leq \mathbf{b}_{lq}, \end{aligned} \tag{7}$$

To ensure smooth grid transition, we enforce continuity on the first N_ϕ derivatives: $\frac{d^j f(t)}{dt^j}|_{t=t_{i-1}} = \frac{d^j f(t)}{dt^j}|_{t=t_i}$, where $i = 1 \cdots M-1$ and $j = 1 \cdots N_\phi$. Continuity, initial and final states constraints can all be written in the form of (7).

Small deviations in higher order coefficients caused by numerical error may result in large changes to the trajectory. We utilize the method proposed in [8] to handle this issue, where end-segment derivatives are computed, as opposed to the polynomial coefficients. Let $\mathbf{d} = [{\mathbf{d}_1}^{\mathrm{T}}, \cdots, {\mathbf{d}_M}^{\mathrm{T}}]^{\mathrm{T}}$, $\mathbf{d}_i = [d_{i0}, \cdots, d_{iN}]^{\mathrm{T}}$, and $d_{ij} = \frac{d^j f(t)}{dt^j}|_{t=t_i}$, we can obtain a linear transform that satisfies $\mathbf{p} = \mathbf{A}_d\mathbf{d}$. (7) can be rewritten as

$$\begin{aligned} \min \quad & \mathbf{d}^{\mathrm{T}}\left({\mathbf{A}_d}^{\mathrm{T}}\mathbf{H}\mathbf{A}_d\right)\mathbf{d} \\ s.t. \quad & \left(\mathbf{A}_{eq}\mathbf{A}_d\right)\mathbf{d} = \mathbf{b}_{eq} \\ & \left(\mathbf{A}_{lq}\mathbf{A}_d\right)\mathbf{d} \leq \mathbf{b}_{lq}. \end{aligned} \tag{8}$$

C. Enforcing Corridor Constraints

We enforce two types of constraints: window constraints and grid boundary constraints, to ensure that the trajectory stays within the virtual flight corridor $(\mathcal{C}_p, \mathcal{W}_p)$.

The trajectory enters and exits the i^{th} grid in the corridor at t_{i-1} and t_i respectively. Window constraints force the trajectory to arrive at the window at a predefined time:

$$w_i^{l_0} \leq f(t_i) \leq w_i^{l_1}, \tag{9}$$

where $i = 1 \cdots M-1$, and $l \in \{x, y, z\}$. Note that since the window has no thickness, one of the dimensions of (9) becomes an equality constraint.

However, even if the trajectory enters and exits a grid through windows, it is still possible that the trajectory goes outside the grid boundary while transitioning between two windows. To this end, we propose an novel algorithm to enforce the grid boundary by considering the extrema of the polynomial. We first solve the QP in (8) without any grid boundary constraints and use it as an initial solution. 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 and solve the QP again with additional boundary constraints:

$$\begin{cases} f_i^{k+1}(t_{jk}^e) \leq c_i^{l_1} - \delta_p & \text{if } f_i^k(t_{jk}^e) > c_i^{l_1} \\ f_i^{k+1}(t_{jk}^e) \geq c_i^{l_0} + \delta_p & \text{if } f_i^k(t_{jk}^e) < c_i^{l_0} \end{cases}, \tag{10}$$

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

Note that the extrema can only be checked after the 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 extremum 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 long as velocities are bounded. This saves a significant amount of computational power compared to the dense constraint points method used in [2].

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 consider the upper boundary case, which can be easily transformed into the lower boundary case. Let $f : t \to y$ be the N^{th} order scalar polynomial function that describes one dimension of the trajectory segment between $[0,\ t_f]$. The grid upper boundary of this segment is y_b. A constraint point can be written as $y_c = y_b - \delta_p$, where δ_p is a positive margin. Suppose $f(0)$ and $f(t_f)$ are bounded within y_b, and the derivative of $f(t)$ is also bounded as $\dot{y}_m$ such that $\forall t^* \in [0,\ t_f]$, $\dot{y}_m \geq \|\frac{df}{dt}|_{t=t^*}\|$.

Consider a boundary violation case (Fig. 5), where the trajectory breaks the boundary y_b between two adjacent constraint points (t_l, y_c) and (t_r, y_c). It must satisfy that $f(t_l) \leq y_c$, $f(t_r) \leq y_c$ and $f(t_e) \geq y_b$, 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 $\frac{df}{dt}|_{t=t^*} = \frac{f(t_e)-f(t_l)}{t_e-t_l} \geq \frac{y_b-y_c}{t_e-t_l} = \frac{\delta_p}{t_e-t_l}$. Since the derivative is bounded, we have:

$$\dot{y}_m \geq \frac{df}{dy}|_{t=t^*} \geq \frac{\delta_p}{t_e - t_l}, \tag{11}$$

which suggests that:

$$t_e - t_l \geq \frac{\delta_p}{\dot{y}_m}. \tag{12}$$

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

$$t_r - t_l = (t_r - t_e) + (t_e - t_l) \geq 2 \cdot \frac{\delta_p}{\dot{y}_m}. \tag{13}$$

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 $\lceil \frac{\Delta t}{2 \cdot \frac{\delta_p}{\dot{y}_m}} \rceil$ constraint points to completely bound the trajectory within the grid. ■

According to (13), any extrema that breaks the grid boundary will be at least $\frac{\delta_p}{\dot{y}_m}$ 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 $\frac{\delta_p}{\dot{y}_m}$. 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. Even when δ_p is set to $0.002m$ and the velocity varies from $0.5m/s$ to $5.0m/s$, almost all violation cases are removed in a few iterations with far less constraint points than the worst case scenario shown in Theorem 1.

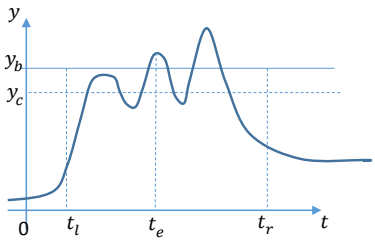


Fig. 5. Illustration of a grid boundary violation case.

D. Enforcing High-Order Dynamical Constraints

The method presented in Sect IV-C can also be utilized to bound higher-order derivatives of the trajectory to a predefined maximum value to ensure that the trajectory is dynamically feasible 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.

E. Implicit Segment Duration Adjustment via Constraint Relaxation

As shown in (5), the duration of each segment is required for the generation of polynomial trajectories. One may simply approximate segment duration to be proportional to the grid size, or use a more conservative strategy by simulating an accelerate-cruise-decelerate velocity profile. In either case, the segment duration is suboptimal and further adjustment is beneficial. Iterative gradient descent approaches are used in [2, 8] for explicit optimization of segment duration. However, these approaches introduce significantly heavier computational cost. In this section, we propose a novel, non-iterative, implicit time adjustment method that requires almost no extra computation.

Instead of directly optimizing for segment duration, we adjust the length of segments by allowing moving connecting points between two adjacent segments. As previously stated in Sect. III-B, two segments are connected at a window, moving connecting points can be done by enlarging the thickness of the window to be nonzero. Note that enlarging the window thickness is equivalent to relaxing a window constraint to be a window "volume" constraint.

Firstly, we need to ensure that the window "volume" still fits in the flight corridor. However, moving the connecting points towards the smaller grid may cause violation of corridor constraints at non-extremum points (Fig. 6). One way to fix this issue is relaxing only towards one side of the window, from the smaller grid towards the bigger grid (Fig. 7).

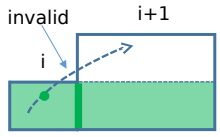


Fig. 6. Double-sided constraint relaxation that causes the trajectory go outside the corridor at non-extrema points. Such non-extrema corridor constraint violations cannot be fixed using the method presented Sect. IV-C.

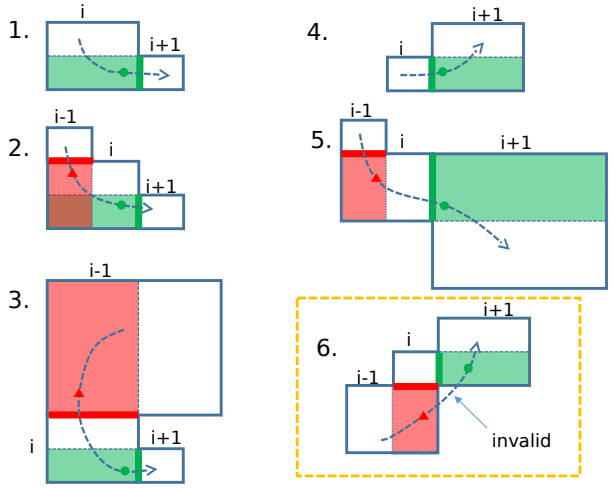


Fig. 7. All possible single-sided window constraint relaxation combinations. The entry window/volume for the i^{th} grid is shown in red, while its exit window/volume is shown in green. The corresponding connecting points between adjacent polynomial segments are shown as red triangles and green dots, respectively. Cases 1-5 represent different relaxation combinations, where the treatment of the window between the i^{th} and the $(i+1)^{th}$ grids is jointly determined by the size of the two grids and the relaxation behavior from the $(i-1)^{th}$ to the i^{th} grid. This sequential update of constraint relaxation is realized by the finite state machine in Fig. 8. Note that case 6 is invalid as it allows corridor violation at non-extremum points. This is regarded as "No Relaxation" in Fig. 8.

A list of single-sided window relaxation cases for the i^{th} grid are shown in Fig. 7. Consider the window between the i^{th} and the $(i+1)^{th}$ grids, when the i^{th} grid is larger than the $(i+1)^{th}$ grid, the window will be relaxed towards the i^{th} grid (Backward Relaxation). Otherwise, the constraint relaxation will be towards the $(i+1)^{th}$ grid (Forward Relaxation). Note that there exists an exception in the "large-small-large" ("backward-forward") case, in which the two relaxed windows on the i^{th} grid may result in corridor violation at non-extremum points, as indicated in case 6 of Fig. 7.

Formally, we can write the Backward Relaxation of the window constraints as:

$$\min(c_i^{l_0}, w_i^{l_0}) \leq f(t_i) \leq \max(c_i^{l_1}, w_i^{l_1}), \qquad (14)$$

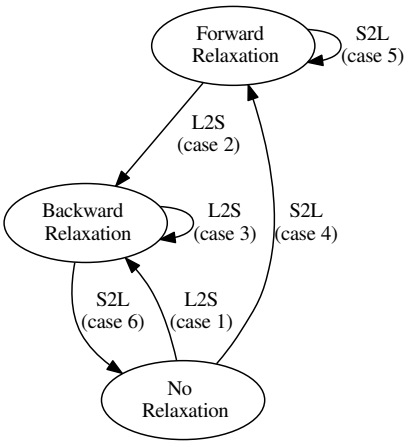


Fig. 8. A diagram of the finite state machine for window constraint relaxation. S2L means small-to-large. The similar conversion follows for other abbreviations. Each edge in the graph corresponds to one case in Fig. 7. Note that the backward+forward combination is not allowed as it may result in violation of corridor constraints as shown in case 6 in Fig. 7.

and Forward Relaxation as:

$$\min(c_{i+1}^{l_0}, w_i^{l_0}) \leq f(t_i) \leq \max(c_{i+1}^{l_1}, w_i^{l_1}), \qquad (15)$$

where l is the dimension where the i^{th} window has zero window thickness , and $w_i^{l_0} = w_i^{l_1}$. The relaxation procedure, although it looks complicated, can be summarized as a simple finite state machine as shown in Fig. 8.

Despite the fact that a window constraint can only be relaxed in one direction towards the bigger grid, it still significantly lower the trajectory cost. Moving the segment connecting points into the bigger grids effectively decreases the time required to travel through the smaller grids. Our method guarantees that every two successive grids will have at least one window "volume", thus most of the grids are able to implicitly adjust the flight time.

V. Complexity Analysis

As stated in Sect. IV, suppose the corridor consists of M grids, our approach will generate M polynomial trajectory segments. There are $O(M)$ variables for fixed order (5 in our case) polynomial trajectories. As we construct the linear equality and inequality constraints for the quadratic programming according to Sect. IV-C and Sect. IV-D, for each dimension (out of $\{x, y, z\}$), 6 equality terms are needed to specify the initial and final states, $3(M-1)$ equality constraints are needed to ensure continuity between segments. To enforce corridor constraints, we use R_0 inequality terms for boundary constraints and $2M$ inequality terms for windows constraints $R_1 + R_2$ inequality terms are used to enforce the feasibility of velocity and acceleration. It is summarized as $O(M + R_0 + R_1 + R_2)$ number of linear equality and inequality constraints for the quadratic programming in (7), where R_0, R_1, R_2 are extrema constraints that are iteratively added as in Sect. IV-C. The number of extrema constraints is theoretically bounded (Theorem 1) and practically very small. Since the quadratic programming problem (as in (7)) has the positive definite Hessian matrix $\mathbf{H}$ [14], one iteration of the algorithm can be solved in polynomial time with complexity $O(M^2 \cdot (M + R_0 + R_1 + R_2))$.

VI. Results

Three simulations and one real-world flight experiments are presented to validate our approach. The planning and trajectory generation system are implemented in C++11 using the 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) [15].

A. Comparison with Waypoint-Based Trajectory Generation

We first conduct simulations to compare our algorithm with the waypoint-based polynomial trajectory generation method presented in [8]. We use the same time allocation for both methods and fix the flight time as 24 seconds. The maximum velocity and acceleration are 1.0 m/s and 1.0 m/s^2 respectively. Constraints margins δ_p, δ_v and δ_a (Sect. IV-C and Sect. IV-D) are set as $0.002m$, $0.05m/s$ and $0.05m/s^2$ respectively.

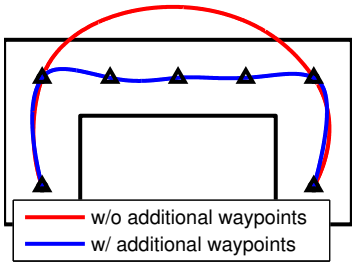
As shown in Fig. 9(a), when a minimum number of waypoints are used, the waypoint-based trajectory deviates from the straight line path and collides with obstacles. This invalid trajectory has the cost (as in (6)) of $(0.009, 0.010)$ in the x and y directions respectively. Adding more waypoints prevents collisions but it also makes the trajectory less smooth and increases the trajectory cost to $(0.025, 0.105)$. On the other hand, our approach, even without the constraint relaxation (Sect. IV-E), generates smooth and collision-free trajectory with cost $(0.021, 0.106)$. The cost is further reduced to $(0.003, 0.085)$ after the constraint relaxation is applied.

B. Performance with Different Dynamical Constraints

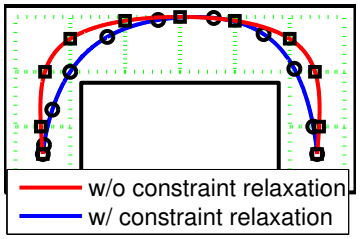
We also evaluate the performance of our method with different dynamical constraints. The map is the same as the one in Fig. 9. We bound the velocity to be 1.0 m/s and change the maximum acceleration from 0.5m/s^2 to 3.5m/s^2. We follow an accelerate-cruise-decelerate velocity profile as the segment duration allocation heuristic. As shown in Fig. 10, as the acceleration limit increases, the trajectory time decreases, the trajectory cost increases, and the velocity approaches its limit. When the acceleration is set to be 4.0 m/s^2 or more, the trajectory becomes infeasible due to the violation of the velocity constraint.

C. Simulation in Complex Environments

We design two simulation experiments to showcase performance of our algorithm in handling complex environments. All the simulations are conducted on a desktop computer with a 3.20 GHz Core i5-4570 CPU.

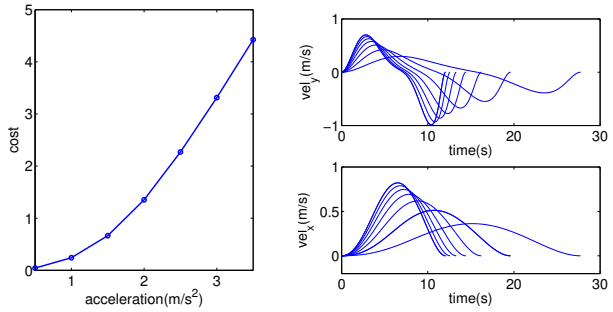


(a) Waypoint-based method [8]



(b) Proposed method

Fig. 9. Fig. 9(a) shows the polynomial trajectory generated from waypoints. Fig. 9(b) shows our result that utilizes the virtual flight corridor. Connecting points between segments are marked.



(a) Trajectory costs (b) Velocity profile

Fig. 10. Changes in trajectory costs and velocity profile with different dynamical constraints. Each curve in Fig. 10(b) corresponds to an acceleration constraint as in Fig. 10(a).

1) Maze: We generate a complex 3-D maze with volume $10m \times 10m \times 10m$, the finest grid resolution of $0.32m \times 0.32m \times 0.32m$, and a total of 6774 occupied grids. The quadrotor is required to travel between two corners of the maze with significant 3-D motions, as shown in Fig. 11. Our algorithm is able to generate a trajectory of length $34.373m$ in only 0.051 seconds. The computation time includes both corridor generation (Sect. III) and trajectory generation (Sect. IV).

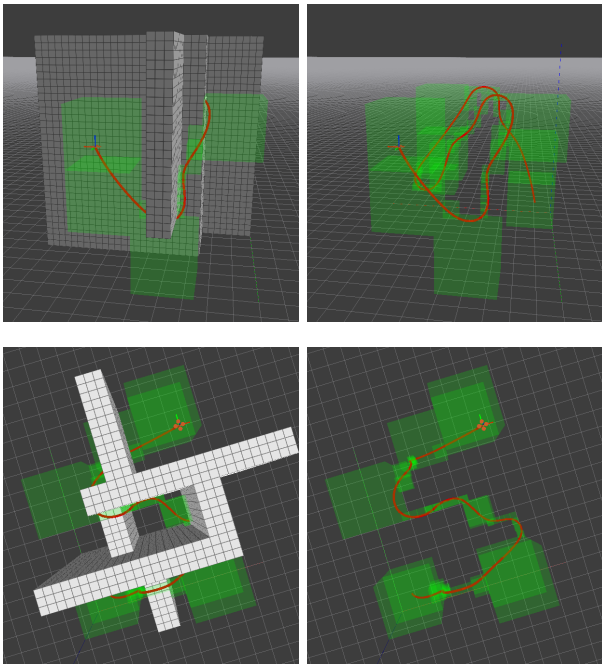


Fig. 11. Different views of a complex trajectory through a maze. The trajectory is shown both with and without obstacles.

2) TUM NAVVIS: To show that our algorithm is able to handle real sensor data, we run our algorithm on a 3-D pointcloud dataset from the TUM NAVVIS project [16]. The volume of the environment is $78m \times 50m \times 20m$ with 259,237 occupied grids at the finest resolution of $0.20m \times 0.20m \times 0.20m$. A trajectory with length $42.201m$ is shown in Fig.2. The trajectory is computed in 0.267 seconds.

D. Online Experimental Results

We build a quadrotor testbed (Fig. 1) to verify our algorithm in a real-world setting. The quadrotor is based on the DJI F450 frame[1] and it is equipped with a PixHawk flight controller[2] and an Intel NUC onboard computer. Other onboard sensors include a forward-facing camera, a downward-facing camera, and a sonar. For state estimation, we run an optical flow-based velocity estimator [17] and a tag-based SLAM with iSAM as the optimization backend [18]. All measurements are fused with the onboard IMU using an extended Kalman filter (EKF). We use a nonlinear tracking controller for trajectory tracking [3]. All software is implemented in C++ using ROS[3] as the communication middleware.

The experiment is conducted in a classroom with flight space $4m \times 3m \times 1.5m$. The finest grid resolution is $0.32m \times 0.32m \times 0.32m$. Tags are attached to obstacles for easy detection and for help with localization. The limits for flight velocity and acceleration are 1.0 m/s and 2.0 m/s^2 respectively. The trajectory can be computed in less than

[1]http://www.dji.com/
[2]http://pixhawk.org/
[3]http://www.ros.org/

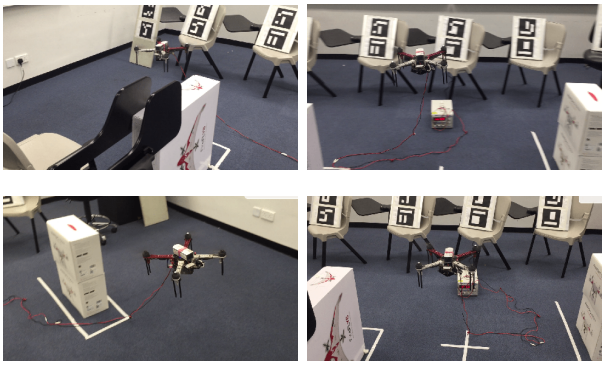
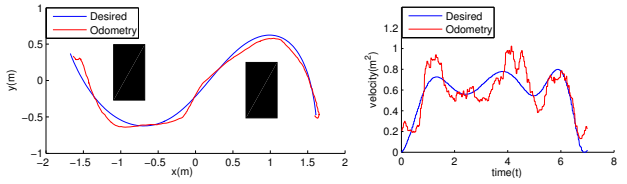
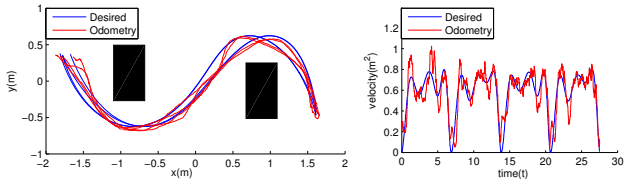


Fig. 12. Snapshots of the online experiment. A video of the experiment can be found at `http://www.ece.ust.hk/~eeshaojie/ROBIO2015.mp4`.



(a) Single trial



(b) Continuous trials

Fig. 13. Position and velocity plots for the online experiment. Obstacles are marked as black boxes. Since there are no significant changes in height, all plots are in 2-D.

0.010 seconds. Fig.12 shows our quadrotor autonomously and smoothly flying through the obstacle field with the velocity profile of single and continuous trials shown in Fig. 13.

VII. Conclusion and Future Work

In this paper, we present an algorithm for real-time generation of smooth and collision-free trajectories for autonomous flight of a quadrotor through 3-D cluttered environments. We propose the concept of the virtual flight corridor and develop an optimization-based method for efficient generation of multi-segment polynomial trajectories that fit within the corridor. The proposed algorithm runs real-time onboard our quadrotor experimental testbed. Both simulation and experimental results are presented.

Our approach can be further extended into an incremental framework such that is can be used in an exploration setting. We also aim to incorporate dynamic obstacle avoidance into our framework.

References

[1] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, "Vision-based state estimation and trajectory control towards high-speed flight with a quadrotor," in *Proc. of Robot.: Sci. and Syst.*, Berlin, Germany, 2013.

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

[3] T. Lee, M. Leoky, and N. McClamroch, "Geometric tracking control of a quadrotor uav on SE(3)," in *Proc. of the Intl. Conf. on Decision and Control*, Atlanta, GA, Dec. 2010, pp. 5420–5425.

[4] J. Jeon, S. Karaman, and E. Frazzoli, "Anytime computation of time-optimal off-road vehicle maneuvers using the RRT*," in *Proc. of the IEEE Control and Decision Conf.*, Orlando, FL, Dec. 2011, pp. 3276–3282.

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

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

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

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

[9] 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. of the IEEE Intl. Conf. on Robot. and Autom.*, Saint Paul, MN, May 2012, pp. 346–353.

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

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

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

[13] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Auton. Robots*, 2013.

[14] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

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

[16] R. Huitl, G. Schroth, S. Hilsenbeck, F. Schweiger, and E. Steinbach, "TUMindoor: An extensive image and point cloud dataset for visual indoor localization and mapping," in *Proc. of the International Conference on Image Processing*, Orlando, FL, USA, Sept. 2012, dataset available at http://navvis.de/dataset. [Online]. Available: http://navvis.de/dataset

[17] V. Grabe, H. H. Bulthoff, , and P. R. Giordano, "Robust optical-flow based self-motion estimation for a quadrotor UAV," in *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*, Vilamoura, Algarve, Portugal, Oct. 2012, pp. 2153–2159.

[18] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Incremental smoothing and mapping," *IEEE Trans. Robot.*, vol. 24, no. 6, pp. 1365–1378, Dec. 2008.