Online Safe Trajectory Generation For Quadrotors Using Fast Marching Method and Bernstein Basis Polynomial

Fei Gao, William Wu, Yi Lin and Shaojie Shen

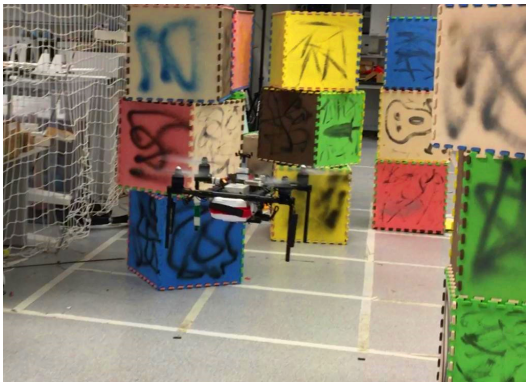
***Abstract*— In this paper, we propose a framework for online quadrotor motion planning for autonomous navigation in unknown environments. Based on the onboard state estimation and environment perception, we adopt a fast marching-based path searching method to find a path on a velocity field induced by the Euclidean signed distance field (ESDF) of the map, to achieve better time allocation. We generate a flight corridor for the quadrotor to travel through by inflating the path against the environment. We represent the trajectory as piecewise Bézier curves by using Bernstein polynomial basis and formulate the trajectory generation problem as typical convex programs. By using Bézier curves, we are able to bound positions and higher order dynamics of the trajectory entirely within safe regions. The proposed motion planning method is integrated into a customized light-weight quadrotor platform and is validated by presenting fully autonomous navigation in unknown cluttered indoor and outdoor environments. We also release our code for trajectory generation as an open-source package.**

I. Introduction

In recent years, micro aerial vehicles (MAVs), especially quadrotors, have drawn increasing attention on various applications. Thanks to their mobility, agility, and flexibility, quadrotors are able to fly rapidly and safely through complex environments while avoiding any unexpected collisions. In this paper, we propose a quadrotor motion planning method for online generating not only safe but also dynamically feasible trajectories in unknown environments. We integrate the proposed motion planning module into a fully autonomous vision-based quadrotor system and present online experiments in unknown complex indoor and outdoor environments to validate the robustness of our method.

Given measurements from onboard sensors, configuration space for a motion planning task is built incrementally to represent the environment. To online generate safe and smooth trajectories for quadrotors, our previous work [1] and other works [2, 3] have been proposed to extract the free space in the configuration space and represent the free space by using a series of convex shapes. Then the trajectory generation problem is formulated as confining the trajectory within the convex shapes by convex optimization. However, two issues exist in these methods. The first issue is the time allocation for piecewise trajectories, a poorly chosen time allocation would easily produce a low-quality trajectory.

This work was supported by HKUST project R9341, and the Joint PG Program under the HKUST-DJI Joint Innovation Laboratory. All authors are with the Department of Electronic and Computer Engineering, Hong Kong University of Science and Technology, Hong Kong, China. fgaoaa@ust.hk, wwuar@connect.ust.hk, ylinax@ust.hk, eeshaojie@ust.hk



(a) Autonomous flight in an unknown indoor environment

(b) Autonomous flight in an unknown outdoor environment

Fig. 1. Our quadrotor platform equipped with a monocular fish-eye camera and an IMU. Onboard computation resources include an Intel i7-5500U CPU running at 3.00 GHz and an Nvidia TX1. We demonstrate fully autonomous flights through complex indoor and outdoor environments. Video is available at https://www.youtube.com/watch?v=Dn6pXL3GqeY&t=1s

And the second issue is how to efficiently bound the entire trajectory within the free space and its derivatives within the feasible space with hard constraints. In this paper, we use fast marching method [4] in a velocity field based on Euclidean signed distance field (ESDF) to search a time-indexed path. The resulted time allocation is adaptive to obstacle densities in environments and achieves better trajectory quality compared to benchmark methods. To resolve the second issue, instead of using the traditional monomial polynomial basis, we use Bernstein basis to represent the trajectory as piecewise Bézier curves. In this way, the trajectory's safety and high-order dynamical feasibility are guaranteed through the optimization. We summarize our contributions as:

1) A fast marching-based method applied on a Euclidean signed distance field (ESDF)-based velocity field, for searching a time-indexed path which provides reason-

able time allocation for trajectory optimization.
2) A trajectory optimization framework, for generating trajectories which are guaranteed to be smooth, safe and dynamically feasible by using Bernstein basis.
3) Real-time implementation of the proposed motion planning method and system integration in a complete quadrotor platform. Autonomous navigation flights in unknown indoor and outdoor complex environments are presented and the source code will be released.

We review related literature in Sect. II. Our path finding method and trajectory optimization method are detailed in Sect. III and in Sect. IV, respectively. The implementation details about the proposed method and the quadrotor system settings are presented in Sect. V. In Sect. VI, benchmark results are given. Indoor and outdoor experimental results which show fully autonomous flights in unknown environments are also presented. The paper is concluded in Sect. VII.

II. Related Work

The motion planning module plays an important role in an autonomous robotic system and is built upon on the top-level of the system to coordinate the robot navigating through complex environments. Roughly speaking, the motion planning problem can be divided as front-end feasible path finding and back-end trajectory optimization.

For the front-end pathfinding, methods ranging from sampling-based [5] to searching-based [6] have been proposed and applied. The representative sampling-based method rapidly-exploring random tree (RRT) was developed by LaValle [5]. In this method, samples are drawn randomly from the configuration space and guide a tree to grow towards the target. Asymptotically optimal sampling-based methods including PRM*, RRG, and RRT*, are proposed by Karaman and Frazzoli [7]. In these methods as the samples increase the solution finally converges to global optimal. Searching-based methods discretize the configuration space and convert the pathfinding problem to graph searching. The graph can be defined in 3D space or higher-order state space. Typical recent methods include JPS [8], ARA* [6] and hybrid A* [9]. Usually, the path found by the front-end cannot be executed by vehicles directly due to the dynamical limits of the vehicles and the poor smoothness of the path itself. Thus the back-end optimization module is necessary to parametrize the path to time t and generate smooth trajectories.

The time parametrization or so-called time allocation procedure dominates the parameterization of the trajectory in continuous space, and is regarded as one of the main reasons leads to sub-optimality of the trajectory optimization. One way to achieve good time allocation is by iteratively refining allocated times using gradient descent [10]. But iteratively regenerating the trajectory is too costly and prevent this method to be used in real-time. In our previous work [1], we allocate times for each piece of the trajectory according to the Euclidean distance between the segmentation points and rely on the overlapping regions of the flight corridor to adjust the time allocation. Others also use some heuristic [3] in the time allocation. In this paper, instead of allocating times based on the discrete path searched by the independent front-end path searching, we tightly incorporate the time allocation into the path searching by using the fast marching method. We use fast marching method on a velocity field induced by ESDF of the map, to search a path which is naturally parameterized to time and adaptive to obstacle density. In [11], a similar approach named Fast Marching Square (FM^2) is proposed, in which fast marching method is used twice. In the first wave propagation, wavefronts are originated from all obstacles in the map, and a potential field is built. Based on this potential field, fast marching is called again from the start point to search a path to the target point. However, the first wave expansion costs too much time to complete, especially in dense environments, making it hard for real-time usage.

Many methods were proposed to optimize the path to generate a smooth trajectory. Gradient-based methods like CHOMP [12] formulates the trajectory optimization problem as a nonlinear optimization over the penalty of the safety and the smoothness. Mellinger et al. [13] pioneered a minimum snap trajectory generation algorithm, where the trajectory is represented by piecewise polynomial functions, and the trajectory generation problem is formulated as a quadratic programming (QP) problem. Chen et al. [2], Liu et al. [3] and our previous work [1] all carve a flight corridor consisting of simple convex regions against the environments and confine the trajectory within the corridor. The safety and dynamical feasibility are achieved by iteratively adding constraints on violated extremum and re-solve the convex program in [1, 2]. However, in this way, a feasible solution can only be obtained after a number of iterations. Another way is sampling the time instance along the trajectory and add extra constraints [3] on it. And this would make the complexity of the problem scales badly since as the velocity of the quadrotor increases, constraints must be added densely. In this paper, we use the Bernstein basis to represent the piecewise trajectory. Utilizing the properties of the Bernstein polynomial basis, the safety and dynamical feasibility constraints are enforced on the entire trajectory directly.

III. Fast Marching-Based Path Searching

A. Fast Marching Method

The Fast Marching (FM) method [4], as a special case of Level Set method [4], is a numerical method for simulating the propagation of a wavefront and is widely used in image processing, curve evolution, fluid simulation and path planning. Fast marching method calculates the time when a wave first arrives at a point from the source by assuming the wavefront propagates in its normal direction with speed f. Suppose the propagation speed $f>0$, i.e. the wavefront only evolves outwards, and f is time-invariant and depends only on the position in the space, then the evolution of the wavefront is described by the Eikonal differential equation:

$$|\nabla T(x)| = \frac{1}{f(x)}, \tag{1}$$

where T is the function of the arrival time and x is the position, $f(x)$ depicts the velocity at different position x.

An approximation solution to the Eq.1 in discrete space was introduced in [4], and we leave the details about the solution for brevity in this paper and refer readers to [4].

For path searching, we can define a velocity map for the robot to navigate. After simulating a wave expanded from the start point, arrival time of each point in the map is obtained. By backtracing the path along the gradient descent direction of the arrival time from the target point to the start point, we obtain a path with minimal arrival time. That is the main idea of applying fast marching method in path searching. Unlike other potential field-based methods, fast marching method has no local minimum and is complete and consistent as is proved in [14]. Theoretically, it has the same time complexity of $O(Nlog(N))$ as A*.

For traditional graph search methods used in discretized space, such as Dijkstra or A*, a discrete path is searched and then parametrized to time t. This may lead to sub-optimality of the back-end trajectory optimization since it's hard to parameterize the path to time reasonable given only the path length. However, the path searched by fast marching method is naturally parametrized to t. Since the path is optimal on the metric of arrival time in the velocity field, not the distance, and each node on the path is associated with the first arrival time of a simulated wave propagated from the source point.

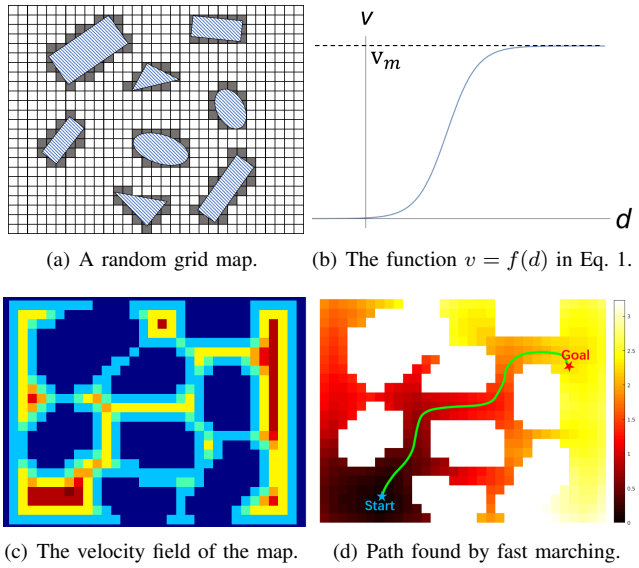
B. Fast Marching in Distance Field

For the purpose of finding a path with properly allocated times, we need to build a reasonable velocity field. We argue that it is natural for the quadrotor to reduce its speed in environments with dense obstacles since the quadrotor has to change its attitude more frequently to avoid collisions. Similarly, the quadrotor should be allowed to move fast in sparse environments. In other words, it's reasonable to assign the allowable velocity of the quadrotor according to the distance to obstacles. Therefore, Euclidean signed distance field (ESDF) is an ideal reference for designing the velocity map. In this paper, we use a shifted hyperbolic tangent function $tanh(x)$ as the velocity function:

$$f(d) = \begin{cases} v_m \cdot (tanh(d-e)+1)/2, & 0 \le d \\ 0, & d < 0 \end{cases} \quad (2)$$

where d is the distance value at position x in the ESDF, e is Euler's number, v_m is the maximum velocity. The plot of the velocity function f is shown in Fig.2(b). This shifted hyperbolic tangent function grows slowly when the distance value is small, i.e. at where near to obstacles, and asymptotically approaches the pre-defined maximum velocity v_m. In occupied areas, the velocity is 0, means wavefront can't expand into these areas. An illustration of the velocity field and the path found by fast marching method is shown in Fig. 2.

In order to speed up the propagation of the wavefront, fast marching method can be driven by heuristic function [15] to reduce the number of expanded nodes. In our implementation, since fast marching is applied on a velocity field, the admissible heuristic $h(x)$ can be estimated by calculating the minimum time to arrival the target $h(x) = d^*(x)/v_m$, where $d^*(x)$ is the Euclidean distance from x to the target point.



(a) A random grid map. (b) The function $v = f(d)$ in Eq. 1.

(c) The velocity field of the map. (d) Path found by fast marching.

Fig. 2. Path founded by fast marching method in a sampled 2-D map. In Fig. 2(a), grids in grey indicate being occupied by obstacels, while white grids are free. The velodity field of the map is obtained based on the euclidean distance field and Eq. 2, and is shown in 2(c). Dark blue indicates 0 velocity and dark red indicates maximum velocity. In Fig. 2(d), the minimum arrival time path is shown in green. Color shows the arrival time in each grid, where white indicates not expanded by the wavefront.

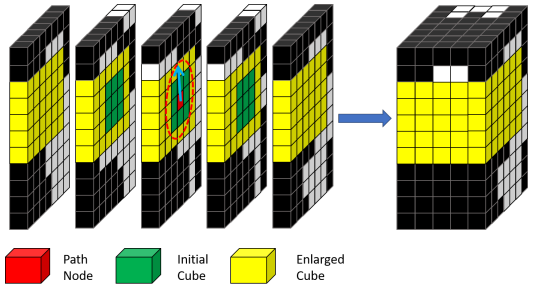
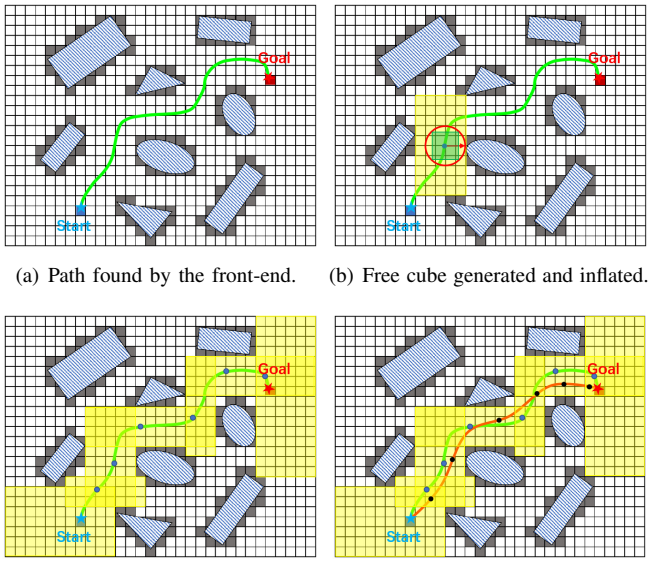


Fig. 3. Illustration of the flight corridor generation and inflation in 3-D occupancy grid map. Red voxel is the node on the path found by fast marching method. Red dashed-line sphere and blue arrow indicate the nearest obstacle to this node. The initial free cube which is shown in green is generated within the sphere and is enlarged to its maximum volume against obstacles in axis-aligned directions. The inflated cube is shown in yellow.

C. Flight Corridor Generation

After obtaining the time-indexed minimal arrival path, we extract the free space in environments to form a flight corridor for the back-end optimization (Sec. IV). We would like to fully utilize the free space, since finding the solution space and obtaining the optimal solution are equally essential for trajectory generation. Therefore our approach is to initialize a flight corridor and enlarge it. For each node in the path, the distance from the node's position to the nearest obstacle is obtained easily by query the ESDF. In this way, at each node, we obtain a sphere of safe space against the nearest obstacle, and we initialize the flight corridor as the inscribed cube of the sphere. Then we enlarge each cube by querying neighbor grids on the axis aligned direction x, y, z till the largest free



(a) Path found by the front-end. (b) Free cube generated and inflated.

(c) Complete flight corridor. (d) Trajectory generated.

Fig. 4. Illustration of the trajecotry generated in corridor. The path found by fast marching method is shown as the green curve. As in 4(b), for one node in the path, the free cube is initialized (in green) in the initial safe region (red circle) against the nearest obstacle and is inflated to its maximum axis-aligned volume (in yellow). After pruning redundant nodes, the flight corridor is formed and is marked in yellow in Fig. 4(c). Finally, the trajectory is optimized to be confined entirely within the flight corridor and satisfy dynamical feasibilities, as the red curve in Fig. 4(d).

volume it may have. The procedure is shown in Fig. 3. Since we initialize the flight corridor as a series of dense nodes, some nodes share the same enlarged cubes, which is unnecessary but increase the complexity of the back-end optimization. Therefore we prune all repeated cubes in the corridor. The pipeline of the path searching, initial corridor generation and corridor inflation is illustrated in Fig. 4.

In this paper, the flight corridor inflation has a similar purpose as in [2]. However, we do not use the octo-map data structure which introduces extra maintenance cost and we do it on a more general map structure voxel grids. Besides, compared to the method proposed in [3], we only inflate the flight corridor in axis-aligned directions, thus we do not need to do range query iteratively and save computational costs.

IV. Safe and Dynamically Feasible Trajectory Generation

The trajectory consisting of piecewise polynomials is parametrized to the time variable t in each dimension μ out of *x*, *y*, *z*. In this paper, instead of using traditional monomial polynomial basis, we use Bernstein polynomial basis similar to [16, 17] and represent the trajectory as piecewise Bézier curve, which is one special case of B-spline curve.

A. Bernstein Basis Piecewise Trajectory Formulation

The Bernstein polynomial basis is defined as:

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

where n is the degree of the basis and $\binom{n}{i}$ is the binomial coefficient. The polynomial consists of Bernstein basis is called a Bézier curve, and is written as:

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

where $[c_j^0, c_j^1, ..., c_j^n]$ denoted as $\mathbf{c_j}$ is the set of control points of the j^{th} piece of the Bézier curve. Compared to monomial basis polynomial, Bézier curve has several special properties:

1) Endpoint interpolation property. The Bézier curve always start at the first control point, end at the last control point, and never pass any other control points.
2) Fixed interval property. The Bézier curve parametrized to variable t is defined on $t \in [0, 1]$.
3) Convex hull property. The Bézier curve $B(t)$ consists of a set of control points c_i are entirely confined within the convex hull defined by all these control points.
4) Hodograph property. The derivative curve $B^{(1)}(t)$ of a Bézier curve $B(t)$ is called as hodograph, and it is still Bézier curve with the controls points defined by $c_i^{(1)} = n \cdot (c_{i+1} - c_i)$, where n is the degree.

Actually, for a Bézier curve, the control points can be viewed as the weights of the basis and also the basis can be viewed as the weights of control points. Since Bézier curve is defined on a fixed interval $[0, 1]$, for each piece of the trajectory, we need a scale factor s to scale the parameter time t to an arbitrary allocated time for this segment. Accordingly, the m-segment piecewise Bernstein basis trajectory in one dimension μ out of *x*, *y*, *z* can be written as follows:

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

where c_{ji} is the i^{th} control point of the j^{th} segment of the trajectory. $T_1, T_2, ..., T_m$ are the end times of each segment. The total time is $T = T_m - T_0$. $s_1, s_2, ..., s_m$ are the scale factors applied on each piece of the Bézier curve, to scale up the time interval from $[0, 1]$ to the time $[T_{i-1}, T_i]$ allocated in one segment. Also, in practice, multipling a scale factor in position of each piece of the curve achieves better numerical stability for the optimization program.

The cost function is the integral of the square of the k^{th} derivative. In this paper we minimize the jerk along the trajectory, so k is 3. The objective is written as:

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

which is indeed in a quadratic formulation $\mathbf{p}^T \mathbf{Q}_o \mathbf{p}$. $\mathbf{p}$ is a vector containing all control points c_{ij} in all three dimensions of *x*, *y*, *z*, and $\mathbf{Q}_o$ is the Hessian matrix of the objective function. Suppose in μ dimension the j^{th} segment of the trajectory is denoted as $f_{\mu j}(t)$, the non-scaled Bézier curve in interval $[0, 1]$ is $g_{\mu j}(t)$, as in Equ. 4. Denote $(t - T_j)/s_j = \tau$.

Then the minimum jerk objective function in the μ dimension of the j^{th} segment is derived and written as follow:

$$J_{\mu j} = \int_0^{s_j} \left(\frac{d^k f_{\mu j}(t)}{dt^k} \right)^2 dt = \int_0^1 s_j \left(\frac{s_j \cdot d^k (g_{\mu j}(\tau)}{d(s_j \cdot \tau)^k} \right)^2 d\tau$$
$$= \frac{1}{s_j^{2k-3}} \cdot \int_0^1 \left(\frac{d^k g_{\mu j}(t)}{d\tau^k} \right)^2 d\tau, \quad (7)$$

which only depends on degree n and scale s_j and can be written following the classical minimum snap formulation [13]. We leave the details of the $\mathbf{Q}_o$ for brevity.

B. Enforcing Constraints

For piecewise trajectory generation, we must enforce a number of constraints to ensure the smoothness, safety as well as the dynamical feasibility of the trajectory. For each piece of the Bézier curves, higher order derivatives of it can be represented by linear combinations of corresponding lower-order control points, which is written as

$$a_{\mu j}^{0,i} = c_{\mu j}^i, a_{\mu j}^{l,i} = \frac{n!}{(n-l)!} \cdot (a_{\mu j}^{l-1,i+1} - a_{\mu j}^{l-1,i}), \ l \geq 1, \quad (8)$$

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

1) Waypoints Constraints: Waypoints that the quadrotor needs to pass through, such as the start and target positons as well as the derivatives up to n^{th} order. Waypoints constraints are directly enforced by setting equality contraints on corresponding control points. For a fixed l^{th} ($l \leq n$) order derivative $d_{\mu j}^{(l)}$ of μ dimension exists at the beginning of the j^{th} piece of the trajectory (waypoints), we have:

$$a_{\mu j}^{l,0} \cdot s_j^{(1-l)} = d_{\mu j}^{(l)}, \quad (9)$$

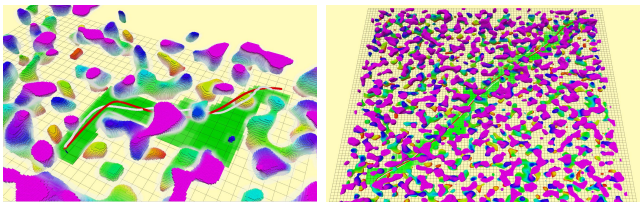
2) Continuity Constraints: The trajectory must be continuous at all the ϕ^{th} derivatives at the connecting point between two pieces ($0 \leq \phi \leq k-1$). The continuity constraints are enforced by setting equality constraints between corresponding control points in two consecutive curves. For the j^{th} and $(j+1)^{th}$ pieces curves, we have:

$$a_{\mu j}^{\phi,n} \cdot s_j^{(1-\phi)} = a_{\mu,j+1}^{\phi,0} \cdot s_{j+1}^{(1-\phi)}, \quad a_{\mu j}^{0,i} = c_{\mu j}^i. \quad (10)$$

3) Safety Constraints: Thanks to the convex hull property of the Bézier curve as is illustrated in the property. 3, the entire trajectory is confined within the convex hull formed by all its control points. Thus we can add safety constraints to enforce all control points of one segment to be inside the corresponding cube generated in III-C. For one segment of the trajectory, since the enlarged cube is a convex polygon, and all control points are inside it, then the convex hull of these control points is surely inside the cube, which means the entire segment of the trajectory is confined within the cube. The safety constraints are applied by adding boundary limit on control points, for control points of the j^{th} segment:

$$\beta_{\mu j}^- \leq c_{\mu j}^i \leq \beta_{\mu j}^+, \ \mu \in \{x, y, z\}, \ i = 0, 1, 2, ..., n, \quad (11)$$

where $\beta_{\mu j}^+$ and $\beta_{\mu j}^-$ are the upper and lower boundary of the corresponding j^{th} cube.



(a) Trajectory generated in a sparse environment. (b) Trajectory generated in a dense environment.

Fig. 5. The time-indexed path searching and trajectory generation. The initial path searched by fast marching method is shown in white curve and the generated trajectory are in red. The flight corridor is in green and is projected into $x-y$ plane for visualization. The map is generated randomly by using Perlin noise. We also release the map generator as an open-source ROS package in `https://github.com/HKUST-Aerial-Robotics/mockamap`.

4) Dynamical Feasibility Constraints: Similarily to the safety constrians, utilizing the properties 3 and 4, we can enforce hard constraints on high order derivatives of the entire trajectory. Constraints are added to bound the derivatives of the curve in each dimenson μ. In this paper, the velocity and acceleration of the quadrotor are constrained in $[v_m^-, v_m^+]$ and $[a_m^-, a_m^+]$ to make the generated trajectory dynamically feasible. For control points of the j^{th} segment we have:

$$v_m^- \leq n \cdot (c_{\mu j}^i - c_{\mu j}^{i-1}) \leq v_m^+,$$
$$a_m^- \leq n \cdot (n-1) \cdot (c_{\mu j}^i - 2c_{\mu j}^{i-1} + c_{\mu j}^{i-2})/s_j \leq a_m^+. \quad (12)$$

The waypoints constraints and the safety constraints are directly formulated as the feasible domain of the optimization variables in each segment ($\mathbf{c}_j \in \Omega_j$). The continuity constraints and high order dynamical constrains are reformulated as linear equality constraints ($\mathbf{A}_{eq}\mathbf{c} = \mathbf{b}_{eq}$) and linear inequality constraints ($\mathbf{A}_{ie}\mathbf{c} \leq \mathbf{b}_{ie}$), here $\mathbf{c} = [\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_m]$. Then the trajectory generation problem is reformulated as:

$$\begin{aligned} \min \quad & \mathbf{c}^T \mathbf{Q}_o \mathbf{c} \\ \text{s.t.} \quad & \mathbf{A}_{eq}\mathbf{c} = \mathbf{b}_{eq}, \\ & \mathbf{A}_{ie}\mathbf{c} \leq \mathbf{b}_{ie}, \\ & \mathbf{c}_j \in \Omega_j, \quad j = 1, 2, ..., m, \end{aligned} \quad (13)$$

which is a convex quadratic program (QP), and can be solved in polynomial time by general off-the-shelf convex solvers.

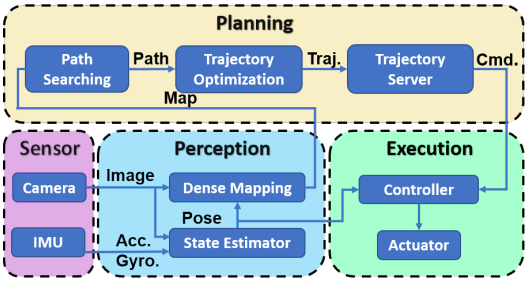
V. Implementation Details

A. System Settings

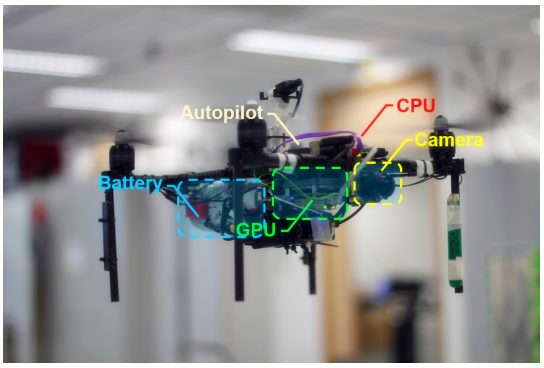
The planning method proposed in this paper[1] is implemented in C++11 using a general convex solver Mosek[2]. The flight experiments are done on a self-developed quadrotor platform (Fig. 6(b)). Motion planning, state estimation, and control modules are running on a dual-core 3.00 GHz Intel i7-5500U processor, which has 8 GB RAM and 256 GB SSD. And the monocular-fisheye mapping module is running at 10 Hz on a Nvidia TX1 which has 256 cores. The system

[1]Source code of the proposed trajectory generation method will be released in `https://github.com/HKUST-Aerial-Robotics/Btraj` after the publishing of this paper.

[2]`https://www.mosek.com`



(a) Architecture of our quadrotor system.



(b) Overview of our quadrotor system.

Fig. 6. The system architecture 6(a) and the hardware setting 6(b) of the quadrotor platform. The quadrotor uses a monocular fish-eye camera and an IMU for localizing itself and mapping the obstacles. The mapping module is running on the Nvidia TX1 GPU cores, while state estimation, control, and the motion planning are running on the Intel i7 CPU. All processings are done onboard. The quadrotor is stabilized by a DJI A3 autopilot.

architecture of the quadrotor system is shown in Fig. 6(a). To provide a large field of view for the mapping module, we use a 235° fish-eye lens and crop images on it to cover the horizontal FOV. The output dense map is fed into the front-end path searching module of the planning layer. Based on it, the path between the current quadrotor state to the target position is searched. Then the path is optimized to generate a time-parametrized smooth, safe and dynamically feasible piecewise trajectory. A trajectory server is then used to receive the trajectory and convert it to control commands. We use a geometric controller [18] to track the generated trajectory, and the attitude control is left to the DJI A3 autopilot on the quadrotor.

B. Dense Mapping For Motion Planning

We utilize our previous monocular visual-inertial state estimation and dense mapping system [19] to build the perception layer of the quadrotor platform. A frontward-looking fish-eye camera with a large field of view of 235° is used to provide the 6-DoF state estimation results for both feedback control and monocular dense mapping. Our dense mapping module fuses each keyframe's depth map into a global map by means of truncated signed distance fusion (TSDF). As an improvement, the sampling depth range is modified to adapt the inflation size for avoiding outliers, which increases the success rate in experiments. Furthermore, in this work, a simple depth map filtering

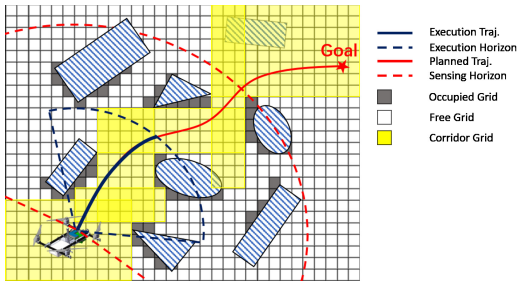


Fig. 7. The planning strategy in V-C. Grey grids are occupied grids detected by the onboard sensing, while white grids are free. Red dashed line and black dashed line indicates the sensing range of our monocular fish-eye mapping module and the execution horizon, respectively. The planned trajectory is shown in a red curve and the trajectory being executed is shown in black.

procedure is utilized to reject depth inconsistency after every dense depth map extraction.

C. Planning Strategy

For onboard usage, the building of the Euclidean distance field is usually the bottleneck of the entire motion planning pipeline. Therefore, in practice, we adopt a local buffer strategy to build the Euclidean signed distance field only in a local range which equals to the range of the mapping results, plus a buffer length. Another practical issue which has been revealed in our previous work [20] is when observations are not sufficient, the monocular fisheye mapping module may output outliers which block the way for finding a feasible path. This issue is especially obvious at where far away from the camera and when there is no significant baseline for motion stereo. One way to resolve this issue is using a double horizon planning strategy, as is shown in Fig. 7. We define an execution horizon and only check the collision status of the current trajectory within this horizon. This strategy makes sense since the camera always has more informative observations at where near it and the mapping module is more confident at a shorter range. During the navigation, if any collision happens within this execution horizon, the re-plan mechanism will be triggered and a new trajectory to the target will be generated.

VI. Results

A. Comparisons and Analysis

There exist other methods utilizing the free-space to formulate a convex flight corridor and generate trajectory within it. Method in [21] uses IRIS (iterative convex regional inflation by semidefinite programming) to obtain the maximum corridor and use combinatorial optimization to generate a trajectory within several convex regions. However, it takes several seconds to generate a trajectory and is not for real-time onboard usage. Therefore we do not compare the proposed method with this method.

Chen's method [2], Liu's method [3] and the proposed method in this paper are all system-level methods for quadrotor motion planning. Compared to these two works, our method utilizes fast marching method in a velocity field to provide a naturally time-indexed path, instead of

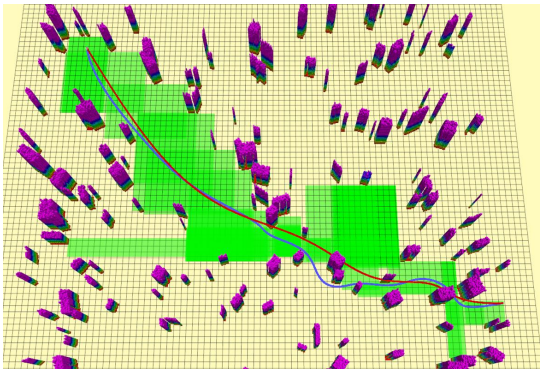
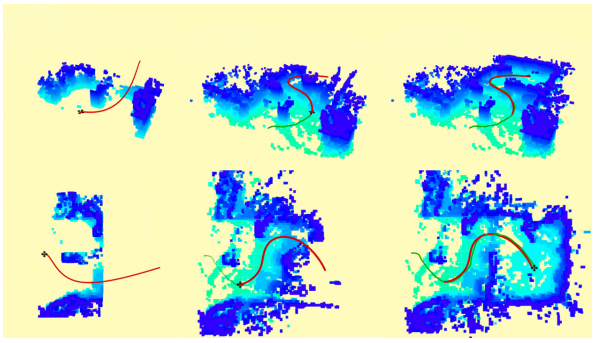


Fig. 8. An instance of the comparison between the proposed method and Chen's method [2]. The map is $80m \times 80m$ and is generated with randomly deployed obstacles. The trajectory generated by the proposed method is shown in red and Chen's method is shown in blue. The corridor in our method is projected into the $x-y$ plane for visualization.

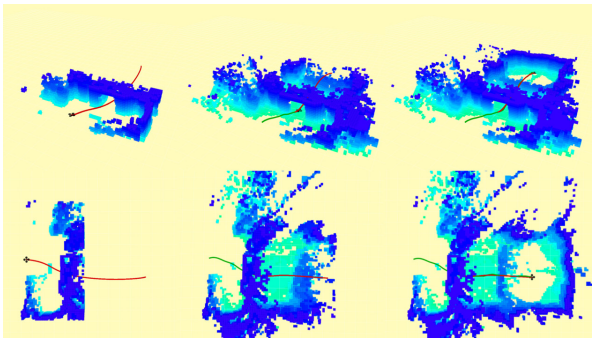
searching a path and allocating time based on some heuristic. Also, we utilize Bernstein polynomial basis to obtain safety and dynamical feasibility directly, which avoids the iterative optimization in Chen's procedure [2] and the collision risk in Liu's sample-based procedure [3]. Here we present a comparison between the proposed method with Chen's method in the aspect of trajectory quality (objective value), trajectory length, trajectory smoothness and running time. 100 trajectories are generated by randomly selecting start and target positions at least $60m$ away in 10 random $40m \times 40m$ forests with obstacles number larger than 100 (as is shown in Fig. 8). Since the velocity of the trajectory has great influence on the objective value, we set the average velocity equal for two methods. The maximum velocity is set as $2m/s$ and the acceleration limit is not set. As the results are shown in Table. VI-A, with roughly similar average velocity, the proposed method achieves lower objective cost, as well as the average and maximum jerk. Fast marching method tends to find longer path then A* since the metric is velocity in the field, not the path length. This results the proposed method generates longer trajectories than Chen's method, as is shown in the table. Besides, as the average velocity increases, computational cost in Chen's method increases. This is because as the average velocity get closer to the limit ($2m/s$), it is harder to confine it by iteratively adding constraints, more iterations are taken in this way.

B. Indoor Autonomous Flight

Indoor autonomous flights are done in previous unknown environments with randomly deployed obstacles to validate our proposed method. A snapshot is shown in Fig. 1(a). During the flights, the environment perception and trajectory generation are all performed online without any prior information about the environment. In the experiments, the dense mapping module outputs a $5m \times 5m$ local dense perception of the environment, and the buffer length is set as $1.5m$. The resolution of our map is set as $0.2m$. The degree n of the Bézier curve is set as 10^{th}. We treat all unknown areas as collision-free areas. In each flight, a target position is sent to the quadrotor and an initial trajectory to the target



(a) Trajectory genetaion in indoor test 1: passing through obstacles.



(b) Trajectory genetaion in indoor test 2: passing beneath an arch.

Fig. 9. Autonomous flights in indoor unknown environments. Two tests in different environments are given in Figs. 9(a) and 9(b). The color code indicates the height of the obstacles. The red curve is the (re)generated trajectories and the green curve is the path that the quadrotor tracked. The re-plan mechanism is triggered once new obstacles are detected, as is shown in the medium of the figures. The complete map and trajectory are shown on the right side of the figures. More indoor trials of our proposed method are presented in the video.

is generated. Re-plan mechanism is triggered based on the strategy illustrated in V-C. Representative demonstrations of the indoor flights are shown in Fig. 9.

C. Outdoor Autonomous Flight

The outdoor experiments are done in unknown complex environments. Figures record several outdoor flights are given in Fig. 10. Complete map and trajectory of a flight are given in Fig. 10(c). The system configuration, parameter settings, and planning strategy are the same as in the indoor experiments VI-B. Average computing time for path searching, corridor generation and trajectory generation are 2.04 ms, 37.17 ms and 26.85 ms, respectively. Note that time cost in trajectory generation is dominated by the degree of the Bézier curves since the number of constraints scale linearly with the number of unknown variables. We refer readers to the video for more trials and details about the outdoor flights.

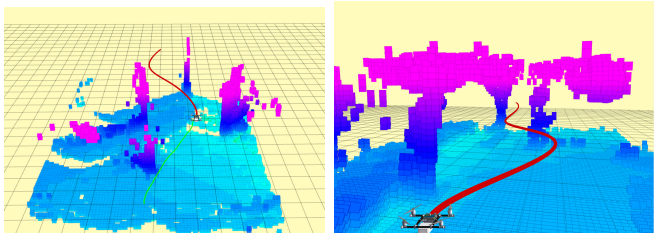
VII. Conclusion and Future Work

In this paper, we propose a novel online motion planning framework for quadrotor autonomous navigation. The proposed method adopts a fast marching-based path searching method to find a time-indexed path in a velocity field adaptive to environments. A flight corridor is generated and inflated based on the path to fully utilize the free space in environments. Finally, we use an optimization-based method

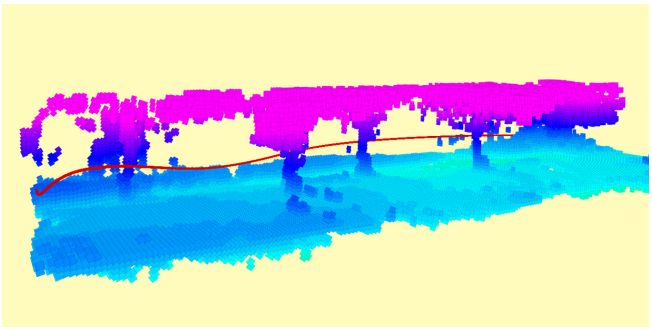
TABLE I

COMPARISON OF TRAJECTORY GENERATION

	Method	Objective Cost	Traj. Lenth (m)	Comp. Time (ms)	Average Vel.	Average Jerk	Max. Jerk
$v = 0.65m/s$	Proposed method	**0.0088**	82.8400	**100.1**	0.6427	**0.0050**	**0.0868**
	Chen's method [2]	0.3058	**80.9628**	128.4	**0.6538**	0.0091	0.1407
$v = 1.0m/s$	Proposed method	**0.0671**	86.3208	**106.5**	**1.0321**	**0.0172**	**0.2080**
	Chen's method [2]	0.4508	**86.1697**	143.4	0.9764	0.0212	0.2800
$v = 1.5m/s$	Proposed method	**0.8961**	85.3192	**101.4**	**1.4745**	**0.0799**	**1.5040**
	Chen's method [2]	6.1166	**84.9211**	198.8	1.4028	0.1098	2.8016



(a) Trajectory generated in test 1 (b) Trajectory generated in test 2



(c) Overview of outdoor test 3

Fig. 10. Autonomous flights in unknown outdoor environments. Marks can be interpreted the same as in Fig. 9. Overall map and trajectory of a flight is shown in Fig. 10(c). Re-plan is triggered using strategy in V-C and only executed trajectories are shown for visualization. More outdoor trials are presented in the video. One can take Fig. 1(b) as reference for the real scene of the flight.

to generate safe and dynamically feasible trajectories with hard constraints by utilizing Bernstein polynomial basis. We integrate the motion planning, state estimation, and dense mapping module into our self-developed quadrotor platform and present fully autonomous flights in both indoor and outdoor unknown environments to validate our method.

In this paper, the convex hull property we use is a conservative way to constrain the trajectory as well as its derivatives, as is illustrated in VI-A. Which means the solution space of the trajectory is a subset of the flight corridor and asymptotically approaches the flight corridor infinitely close as the degree of the polynomial increases. In practice, it's a trade-off between the feasibility and the complexity of the problem. In the future, we plan to do more analysis and comparison between Bernstein and monomial polynomial basis as well as the corresponding optimization methods. We also intend to challenge our quadrotor system in extreme situations such as large-scale or dynamic environments. Furthermore, we plan to extend our motion planning framework to quadrotor swarms.

REFERENCES

[1] F. Gao and S. Shen, "Online quadrotor trajectory generation and autonomous navigation on point clouds," in *Proc. of the IEEE Intl. Sym. on Safety, Security, and Rescue Robotics*, Lausanne, Switzerland, Oct 2016.

[2] J. Chen, T. Liu, and S. Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," in *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*, Stockholm, Sweden, May 2016.

[3] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, 2017.

[4] J. A. Sethian, "Level set methods and fast marching methods," *Journal of Computing and Information Technology*, vol. 11, pp. 1–2, 2003.

[5] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[6] M. Likhachev, G. J. Gordon, and S. Thrun, "Ara*: Anytime a* with provable bounds on sub-optimality," in *Advances in Neural Information Processing Systems*, 2004, pp. 767–774.

[7] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, pp. 846–894, 2011.

[8] D. D. Harabor, A. Grastien, *et al.*, "Online graph pruning for pathfinding on grid maps." in *AAAI*, 2011.

[9] D. Dolgov, S. Thrun, *et al.*, "Path planning for autonomous vehicles in unknown semi-structured environments," *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.

[10] R. Charles, B. Adam, and R. Nicholas, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Proc. of the Intl. Sym. of Robot. Research*, Singapore, Dec. 2013.

[11] A. Valero-Gomez, J. V. Gomez, S. Garrido, and L. Moreno, "Fast marching method for safer, more efficient mobile robot trajectories," *IEEE Robotics & Automation Magazine*, vol. 1070, no. 9932/13, 2013.

[12] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *Proc. of the IEEE Intl. Conf. on Robot. and Autom.*, May 2009.

[13] 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.

[14] S. M. LaValle, *Planning algorithms*. Cambridge uni. press, 2006.

[15] C. Petres, Y. Pailhas, P. Patron, Y. Petillot, J. Evans, and D. Lane, "Path planning for autonomous underwater vehicles," *IEEE Transactions on Robotics*, vol. 23, no. 2, pp. 331–341, 2007.

[16] M. E. Flores, *Real-time trajectory generation for constrained nonlinear dynamical systems using non-uniform rational B-spline basis functions*. California Institute of Technology, 2008.

[17] J. A. Preiss, W. Hönig, N. Ayanian, and G. S. Sukhatme, "Downwash-aware trajectory planning for large quadcopter teams," *arXiv preprint arXiv:1704.04852*, 2017.

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

[19] Y. Lin, F. Gao, T. Qin, W. Gao, T. Liu, W. Wu, Z. Yang, and S. Shen, "Autonomous aerial navigation using monocular visual-inertial fusion," *J. Field Robot.*, vol. 00, pp. 1–29, 2017.

[20] F. Gao, Y. Lin, and S. Shen, "Gradient-based online safe trajectory generation for quadrotor flight in complex environments," in *Proc. of the IEEE/RSJ Intl. Conf. on Intell. Robots and Syst.*, Sept 2017.

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