Planning Dynamically Feasible Trajectories for Quadrotors Using Safe Flight Corridors in 3-D Complex Environments

Sikang Liu, Michael Watterson, Kartik Mohta, Ke Sun, Subhrajit Bhattacharya, Camillo J. Taylor, and Vijay Kumar

***Abstract*—There is extensive literature on using convex optimization to derive piece-wise polynomial trajectories for controlling differential flat systems with applications to three-dimensional flight for Micro Aerial Vehicles. In this work, we propose a method to formulate trajectory generation as a quadratic program (QP) using the concept of a *Safe Flight Corridor* (SFC). The SFC is a collection of convex overlapping polyhedra that models free space and provides a connected path from the robot to the goal position. We derive an efficient convex decomposition method that builds the SFC from a piece-wise linear skeleton obtained using a fast graph search technique. The SFC provides a set of linear inequality constraints in the QP allowing real-time motion planning. Because the range and field of view of the robot's sensors are limited, we develop a framework of *Receding Horizon Planning*, which plans trajectories within a finite footprint in the local map, continuously updating the trajectory through a re-planning process. The re-planning process takes between 50 to 300 ms for a large and cluttered map. We show the feasibility of our approach, its completeness and performance, with applications to high-speed flight in both simulated and physical experiments using quadrotors.**

***Index Terms*—Aerial robotics, autonomous vehicle navigation, motion and path planning.**

I. Introduction

NAVIGATION of a Micro Aerial Vehicle (MAV) in an obstacle-cluttered environment is a challenging problem which requires the MAV not only to detect obstacles, but also plan and execute collision-free and dynamically feasible trajectories. In this letter, we propose an algorithm that efficiently generates these safe and smooth trajectories in real time. We use this algorithm as a foundation for a fast and safe navigation system for a quadrotor (Fig. 1).

Fig. 1. Our experimental quadrotor equipped with a Velodyne VLP-16, a stereo camera and an Intel NUC computer navigating an unknown environment with obstacles.

It has been shown that the trajectory generation problem, for differentially flat systems, can be formulated as a Quadratic Programming (QP) [1]. The trajectory can be parameterized as an k-th order polynomial in time [2]. Generating a collision-free trajectory has been solved with Mixed Integer methods in [3]–[5]. Since solving MILP/MIQP takes seconds to minutes [3]–[5], other approaches have been developed to remove the integer variables and solve the QP instead which is much faster [6]–[9]. A trajectory can be solved in closed form [6], but it requires many iterations to generate a collision-free trajectory especially when the map is complicated. [7] requires an OctoMap [10] representation and produces a sequences of axes-aligned cubes in free space to generate trajectories. This formulation of convex free space is not generic and is efficient only when obstacles are rectangular parallelopipeds. In [11], the author analyzes the high speed navigation through a obstacle field, but they fail to consider non-trivial robot dynamics and their results may not applicable for MAVs. The framework in [12] corrects trajectories based on a prior path, but it requires an accurate prior map which is a limitation for practical navagation in unkown environments.

We adopt some ideas from these related works and propose a robust and efficient solution based on our previous work [13], [9] to generate trajectories in real time. Our pipeline uses a linear piece-wise path from a fast graph search algorithm to guide the convex decomposition of the map to find a *Safe Flight Corridor* (SFC). The SFC is a collection of convex connected polyhedra that models free space in a map and can be treated as linear inequality constraints in the QP for trajectory optimization. Inspired by [14], we developed a novel convex decomposition

Manuscript received September 10, 2016; accepted January 2, 2017. Date of publication February 2, 2017; date of current version May 16, 2017. This letter was recommended for publication by Associate Editor H. Kurniawati and Editor N. Amato upon evaluation of the reviewers comments. This work was supported in part by DARPA Grants HR001151626/HR0011516850, in part by ARL Grant W911NF-08-2-0004, in part by NSF Grant IIS-1426840, and in part by a NASA Space Technology Research Fellowship.

S. Liu, M. Watterson, K. Mohta, K. Sun, C. J Taylor, and V. Kumar are with the GRASP Laboratory, University of Pennsylvania, Philadelphia, PA 19104 USA (e-mail: sikang@seas.upenn.edu; wami@seas.upenn.edu; kmohta@seas.upenn.edu; sunke@seas.upenn.edu; cjtaylor@cis.upenn.edu; kumar@cis.upenn.edu).

S. Bhattacharya is with the Department of Mechanical Engineering and Mechanics, Lehigh University, Bethlehem, PA 18015 USA (e-mail: subhrabh@math.upenn.edu).

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

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

Digital Object Identifier 10.1109/LRA.2017.2663526

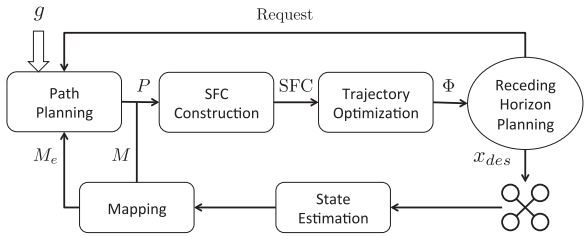

Fig. 2. Block diagram of our autonomous system. We first find a valid path in a grid map toward a goal g, based on which we construct the *Safe Flight Corridor* (SFC) through convex decomposition. The trajectory inside the SFC is achieved from solving a optimization problem. And in the end we are able to get the desired control commands for navigating the quadrotor.

method to construct the SFC using ellipsoids. The total time for trajectory generation using this pipeline is sufficiently small such that we use it with a Receding Horizon Planning (RHP) framework to build our navigation system with mapping and state estimation. We assume the robot is able to follow our generated trajectories through a non-linear controller [15]. We verify the system's robustness for collision avoidance in partially sensed complex environments through both simulation and real world experiments.

In order to guarantee safety, the stopping policy [13] is used. The three main distinguishing advantages of our algorithm can be summarized as:

1) Fast computation
2) High speed trajectory generation
3) Safety and completeness

Compared with our previous work in [9], we improve the planning speed, propose a more generic and effective decomposition method and test the pipeline with much larger traveling distance and higher flight speed. The outline of this letter is as follows: in Section II, we describe the technical approach for trajectory generation; in Section III, we analyze the computational expense and efficiency of our algorithm; experimental results are shown in Section IV; insights and conclusion follow in Section V.

II. Technical Approach

The overall architecture of our autonomous system is shown in Fig. 2. In this section, we mainly discuss the top four components through which we derive the desired trajectory for controlling the MAV to reach the goal. The source code for fast *Path Planning* and convex decomposition can be found in https://github.com/sikang/JPS3D.git and https://github.com/sikang/DecompUtil.git.

A. Path Planning

The environment is represented as an occupancy grid that can be constructed from sensor data such as laser range finder, stereo cameras or RGB-D sensors. A valid collision-free path can then be found in the grid using a graph search algorithm. Randomized methods like RRT* and PRM are probabilistically complete, which means there is no guarantee on the time it takes to find a optimal path if there exist one. Also, their random behavior make the performance of the algorithm unpredictable when we need to re-plan frequently. Search-based algorithms like Dijkstra and A*, on the other hand, are resolution complete, but their computation time for finding an optimal path is a limitation when used with large maps. *Jump Point Search* (JPS) [16] solves this problem by planning in uniform-cost grid maps. Since we are using 3-D grid maps with uniform voxels, JPS can be applied to our problem. JPS prunes the neighbors of a node being searched and potentially reduces the running time of A* by an order of magnitude. In order to use JPS with 3-D voxel maps, we extend the 2-D algorithm proposed in [16] to 3-D. We propose pruning rules for 3-D voxel grids as presented in Figs. 3 and 4. As defined in [16], the *natural neighbors* refer to the set of nodes that remain after pruning. For those neighbors which cannot be pruned due to obstacles, we call them *forced neighbors*.

The details of the recursive pruning and jump processes can be found in [16]. The proposed pruning in Fig. 4 is a compromise between checking all the situations and maintaining

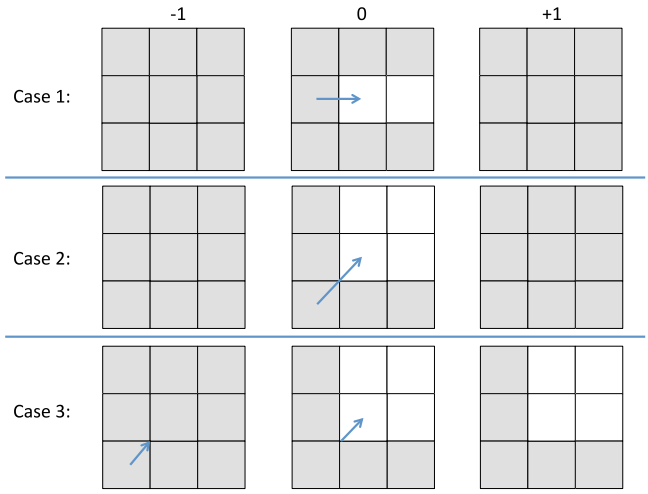

Fig. 3. Neighbor Pruning. We draw a $3 \times 3 \times 3$ voxel grid as three 3×3 2-D layers – bottom (-1), middle (0), top (+1). The center node indicated by the blue arrow is currently being expanded. The *natural neighbors* of the current node are marked white. The pruned neighbors are marked grey. The blue arrow also shows the direction of travel from its parent which includes three cases: (1). straight, (2). 2-D diagonal and (3). 3-D diagonal.

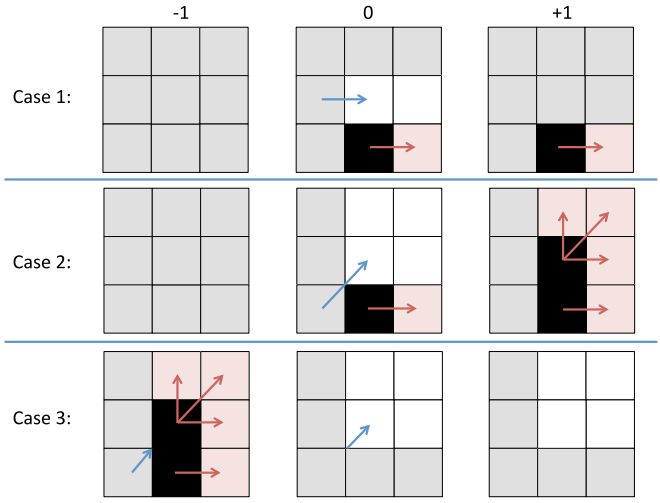

Fig. 4. Forced Neighbors. When the current node is adjacent to an obstacle (black), the highlighted *forced neighbors* (pink) cannot be pruned. The red arrow indicates the pair of an obstacle and its corresponding *forced neighbor*: if the tail voxel is occupied, its head voxel is a *forced neighbor*. For example, in Case 1, if the voxel $(0, 1, 0)$ is occupied, $(1, 1, 0)$ is a *forced neighbor*. In Case 2, the occupied voxel $(0, 0, 1)$ results in three *forced neighbors* and similarly in Case 3. For clarity of figures, we omit drawing the symmetric situations wit respect to the blue arrow.

TABLE I
TRAJECTORY GENERATION RUN TIME (SEC)

Map	Size	# of Cells	# of Trajs	Time (s)	Path Planning A*	Path Planning JPS	Convex Decomp	Traj Opt	Replan (JPS)
Random Blocks	$40 \times 40 \times 1$	1.4×10^6	130	Avg	0.57	0.034	0.0021	0.028	0.065
				Std	1.26	0.034	0.0028	0.022	0.051
				Max	9.98	0.19	0.020	0.099	0.27
Multiple Floors	$10 \times 10 \times 6$	5.9×10^5	147	Avg	6.12	0.039	0.0064	0.082	0.13
				Std	15.77	0.046	0.0038	0.041	0.081
				Max	84.56	0.22	0.021	0.23	0.45
The Forest	$50 \times 50 \times 6$	1.8×10^6	89	Avg	0.65	0.033	0.0039	0.055	0.094
				Std	1.57	0.044	0.0024	0.031	0.068
				Max	7.78	0.20	0.010	0.12	0.30
Outdoor Buildings	$100 \times 110 \times 7$	6.2×10^5	127	Avg	0.54	0.028	0.0066	0.099	0.14
				Std	1.46	0.045	0.0053	0.064	0.10
				Max	10.96	0.27	0.027	0.24	0.47

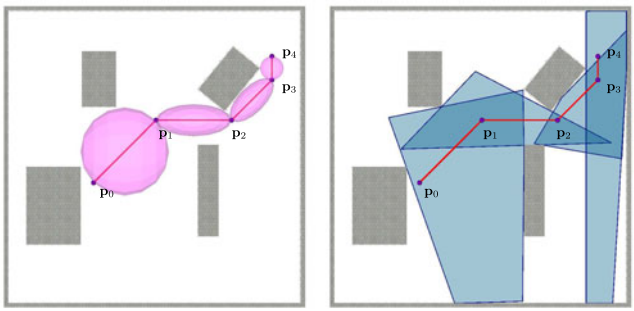

Fig. 5. Generate a *Safe flight corridor* (blue region) from a given path $P = \langle \mathbf{p}_0 \rightarrow \cdots \rightarrow \mathbf{p}_4 \rangle$. Left: find the collision-free ellipsoid for each line segment. Right: dilate each individual ellipsoid to find a convex polyhedron.

simplicity of the algorithm: we add more neighbors than required (three *forced neighbors* case) but it is easier to check (i.e. more efficient). JPS provides the same completeness and optimality guarantees as A* [16], with the only limitation being the assumption of uniform-cost grid which holds for our case. Our 3-D JPS significantly speeds up the running time of planning (column 6 in Table I) which makes it possible to run the trajectory generation within our RHP framework.

B. Safe Flight Corridor Construction

The set of points that constitute the obstacles (the occupied voxels in the 3D grid map representation of the environment) are represented as O. A piece-wise linear path P from start to goal in the free space is denoted as $P = \langle \mathbf{p}_0 \rightarrow \mathbf{p}_1 \rightarrow \cdots \rightarrow \mathbf{p}_n \rangle$, where $\mathbf{p}_i$ are points in the free space and $\mathbf{p}_i \rightarrow \mathbf{p}_{i+1}$ are directed line segments in the free space. We generate a convex polyhedron around each line segment in P to construct a valid SFC. The i^{th} line segment is represented as $L_i = \langle \mathbf{p}_i \rightarrow \mathbf{p}_{i+1} \rangle$. Denote the generated convex polyhedron from each L_i as C_i. The space covered by these convex polyhedra constitutes the *Safe Flight Corridor*. We denote the collection of these convex polyhedra as $\text{SFC}(P) = \{C_i \mid i = 0, 1, \ldots, n-1\}$. Fig. 5 shows a typical example of a path P and corresponding $\text{SFC}(P)$. One criterion for the construction of the SFC is that two consecutive polyhedra, C_i and C_{i+1}, need to intersect in a non-empty set containing $\mathbf{p}_{i+1}$. This ensures continuity in the SFC.

To generate the convex polyhedron C_i from L_i, we describe two procedures: (1) "Find Ellipsoid", that first fits an ellipsoid around L_i, and, (2) "Find Polyhedron", that constructs the polyhedron C_i from tangent planes to a sequence of dilated ellipsoids. In order to reduce the computation time, we add a bounding box to confine the space around L_i in which we consider obstacles. In addition, we propose a shrinking process to guarantee that a non-point robot is collision-free. In the following subsections we introduce the details on these procedures. For simplicity, we remove the subscripts "i" and simply use L, C to denote the corresponding line segment and polyhedron.

1) Step 1 – Find Ellipsoid: In this step we find an ellipsoid which includes the line segment L and does not contain any obstacle points from O. An ellipsoid is described as

$$\xi(\mathbf{E}, \mathbf{d}) = \{\mathbf{p} = \mathbf{E}\bar{\mathbf{p}} + \mathbf{d} \mid \|\bar{\mathbf{p}}\| \leq 1\} \tag{1}$$

For an ellipsoid in $\mathbb{R}^3$, $\mathbf{E}$ is a 3×3 symmetric positive definite matrix that represents a deformation of a sphere ($\|\bar{\mathbf{p}}\| \leq 1$). $\mathbf{E}$ can be decomposed as $\mathbf{E} = \mathbf{R}^T\mathbf{S}\mathbf{R}$ where $\mathbf{R}$ is the rotation matrix aligning the ellipsoid axes with map axes and $\mathbf{S} = \text{diag}(a, b, c)$ is the diagonal scale matrix whose diagonal elements stand for the corresponding lengths of ellipsoid semi-axes. $\mathbf{d}$ indicates the center of the ellipsoid. Without loss of generality, we assume $a \geq b,\ a \geq c$. Our goal is to find $\mathbf{E}, \mathbf{d}$ given the line segment L and obstacles O.

This ellipsoid is computed in two steps: first, we shrink an initial sphere to derive the maximal spheroid (an ellipsoid with two axes of equal length); second, we "stretch" this spheroid along the third axis to obtain the final ellipsoid. In the first step, the initial ellipsoid is a sphere centered at the mid point of L and with diameter equals to the length of L. Assume the length of ellipsoid's $\tilde{x}$-axis is fixed and aligned with L, we reduce the length of other two axes until the spheroid contains no obstacles. This is done by searching for the closest obstacle in O from the center of ξ. Fig. 6 shows the shrinking process from a 2-D perspective.

The maximal spheroid touches an obstacle at $\mathbf{p}^\star$ which, along with the line segment L, defines the plane of $\tilde{x}$-$\tilde{y}$ axes of the spheroid. Following that, we stretch the length of the $\tilde{z}$-axis of the spheroid to make it equal to a to form a new initial ellipsoid. The actual value of c can be determined through find-

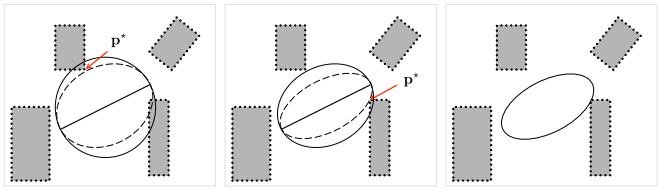

Fig. 6. Shrink ellipsoid ξ. The bold line segment is L, gray region indicates obstacle while the white region is free space. Left: start with a sphere, we find the closest point $\mathbf{p}^\star$ to the center of L and adjust the length of short axes such that the dashed ellipsoid touches this $\mathbf{p}^\star$. Middle: repeat the same procedure, find a new closest point $\mathbf{p}^\star$ and the new ellipsoid. Right: no obstacle is inside the ellipsoid, current ellipsoid is the max spheroid. Several iterations are required to ensure the final spheroid excludes all the obstacles.

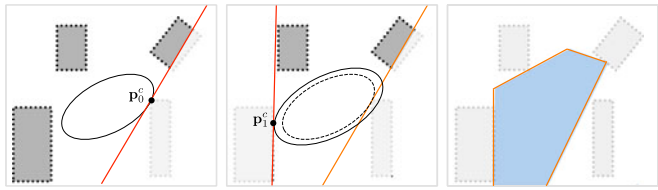

Fig. 7. Dilate ellipsoid ξ^0 to find halfspaces. Left: find the first intersection point $\mathbf{p}_0^c$ for ξ^0 and hyperplane (red line), the obstacle points outside corresponding halfspace H_0 are removed (shadowed). Middle: find the next intersection point $\mathbf{p}_1^c$ (dashed ellipsoid shows the original ellipsoid ξ^0 and the solid ellipoid shows the new ellipsoid ξ^1), keep removing obstacle points from the map that are outside the new halfspace. Right: keep dilating until no obstacle remains in the current map, the convex space C (blue region) is defined by the intersection of the halfplanes.

Algorithm 1: Given $\xi^0(\mathbf{E}, \mathbf{d})$, find the $C(\mathbf{A}, \mathbf{b})$. The set of obstacle points is denoted as O.

function FIND POLYHEDRON(ξ^0, O)
  $O_{remain} \leftarrow O$
  $j \leftarrow 0$
  while $O_{remain} \neq \emptyset$ **do**
    $\mathbf{p}_j^c \leftarrow$ ClosestPoint(ξ^0, O_{remain})
    $\xi^j \leftarrow$ DilateEllipsoid($\xi^0, \mathbf{p}_j^c$)
    $\mathbf{a}_j \leftarrow 2\mathbf{E}^{-1}\mathbf{E}^{-T}(\mathbf{p}_j^c - \mathbf{d})$
    $b_j \leftarrow \mathbf{a}_j^T \mathbf{p}_j^c$
    $O_{remain} \leftarrow$ RemovePoints($\mathbf{a}_j, b_j, O_{remain}$)
  $j = j + 1$
  end while
  $C : \mathbf{A}^T \leftarrow \begin{bmatrix} \mathbf{a}_0^T \\ \mathbf{a}_1^T \\ \vdots \end{bmatrix}$, $\mathbf{b} \leftarrow \begin{bmatrix} b_0 \\ b_1 \\ \vdots \end{bmatrix}$
  return $C(\mathbf{A}, \mathbf{b})$
end function

ing another closest point using the similar process as shown in Fig. 6.

2) Step 2 – Find Polyhedron: Denote the ellipsoid found in the previous step as ξ^0, which touches an obstacle point at $\mathbf{p}_0^c = \mathbf{p}^*$. The tangent plane to the ellipsoid at this point creates a half space $H_0 = \{\mathbf{p} \mid \mathbf{a}_0^T \mathbf{p} < b_0\}$, containing the ellipsoid. After computing H_0, we remove all the obstacles in O that lie outside H_0 (call this the set of *remaining* obstacles, O_{remain}), and "dilate" the ellipsoid (keeping its aspect ratio constant) until it is in contact with another obstacle point, $\mathbf{p}_1^c$, at which point the new ellipsoid is called ξ^1 and the new tangent hyperplane creates a new half-space H_1. This process is continued to obtain a sequence of half-spaces, $H_0, H_1, \cdots, H_m$. The intersection of these $m+1$ halfspaces gives the convex polyhedron, $C = \bigcap_{j=0}^{m} H_j = \{\mathbf{p} \mid \mathbf{A}^T \mathbf{p} < \mathbf{b}\}$, where $\mathbf{a}_j$ and b_j are the j-th column of matrix $\mathbf{A}$ and element of vector b respectively.

Fig. 7 shows an example of ellipsoid dilation. In each dilate iteration, the ellipsoid ξ^j touches an obstacle at a point $\mathbf{p}_j^c$. Algorithm 1 shows the pseudo-code. The hyperplane defining the j-th half-space, H_j, is the tangent to ξ^j at p_j^c, and is

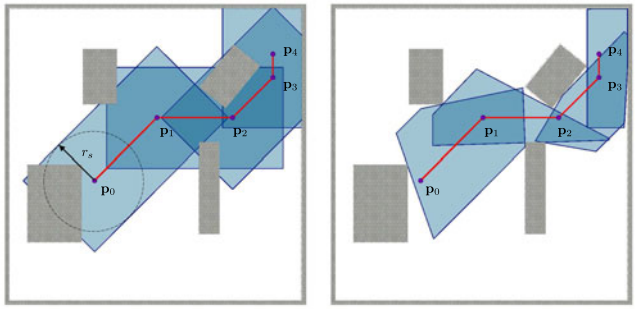

Fig. 8. Left: apply bounding box on each line segment with safety radius r_s. Right: inflate individual line segment to find the convex polyhedron. We only process the obstacles inside corresponding bounding box comparing to Fig. 5.

computed as

$$\mathbf{a}_j = \left.\frac{d\xi_r}{d\mathbf{p}}\right|_{\mathbf{p}=\mathbf{p}_j^c} = 2\mathbf{E}^{-1}\mathbf{E}^{-T}(\mathbf{p}_j^c - \mathbf{d}) \qquad b_j = \mathbf{a}_j^T \mathbf{p}_j^c \tag{2}$$

So far, we are able to generate the polyhedron C for L, given the obstacles O. We apply this method on each individual line segment of the path P to get the *Safe Flight Corridor* as $\text{SFC}(P) = \{C_i \mid i = 0, 1, \ldots, n-1\}$ (Fig. 5). Since the original ellipsoid is inside the corresponding polyhedron, we have a guarantee that the line segment L is also inside the polyhedron. Thus the whole path P is guaranteed to be inside $\text{SFC}(P)$.

3) Bounding Box: The algorithm, as presented, needs to search through all the points in O at least twice to check for the intersection with the inflated ellipsoid when constructing the polyhedron C for each line segment L. This is an expensive process. We decrease the number of points to be checked for by adding a bounding box around L, and thus only searching for the obstacle points inside it. This process saves a large amount of computation time and also prevents the trajectory from going too far away from the original path. The bounding box for L is composed of 6 rectangles such that the axis of the bounding box is aligned with L and the minimum distance from each face to L is r_s. If the maximum speed and acceleration of the MAV is $v_{\max}, a_{\max}$, the condition imposed on the safety radius is $r_s \geq \frac{v_{\max}^2}{2a_{\max}}$. Fig. 8 shows the typical result from applying the bounding box. The generated SFC contains similar halfspaces as shown in Fig. 5.

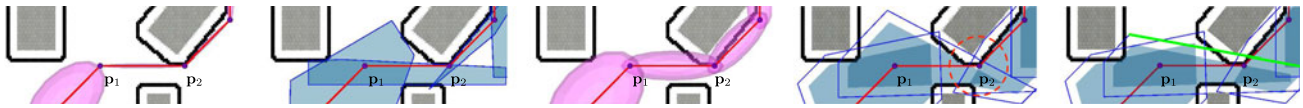

Fig. 9. Constructing the SFC(P) through the shrinking process. For clarity, we draw the contour of the expanded map M_e using black bold lines. The contours of the SFC are indicated by blue boundaries while the shrunken SFCs are drawn as blue regions inside. The SFC in (a) and (b) is derived using M_e without shrinking. Several ellipsoids and corresponding polyhedra are quite narrow. In (c) and (d), the SFC is generated using the original map M such that the corridor is "wider" compared to (a) and (b). Since this SFC also penetrates obstalces in the expanded map, we shrink it by the robot radius r_r to derive the "safe" SFC. However, this shrinking process may cause discontinuities in the SFC, for example $\mathbf{p}_2$ (circled) is outside of the shrunken polyhedron generated from line segment $\mathbf{p}_1 \rightarrow \mathbf{p}_2$ in (d). In (e), the green hyperplane is adjusted such that $\mathbf{p}_2$ is still inside the shrunken polyhedron.

4) Shrink: We model the robot as a sphere with radius r_r and expand occupied voxels in the original map M to generate the configuration space M_e such that we are able to treat the robot as a single point for planning. When constructing the SFC for path P planned in M_e, using M_e could generate narrow ellipsoids and polyhera [Fig. 9(a) and (b)]. In order to avoid such kind of bad SFC, we use the original map M to generate the SFC and shrink the SFC by the robot radius r_r in order to guarantee safety. The shrinking process is applied by pushing every support hyperplane along its normal by r_r. This process ensures the safety of the shrunken SFC as we increase the distance between obstacles and each hyperplane by r_r, but may also exclude some portion of the path [Fig. 9(d)] which may cause discontinuity of the *Safe Flight Corridor*. To guarantee the continuity, we have to make sure the line segment L is inside the shrunken polyhedron C'. For this, we modify the Algorithm 1: for any halfspace $H_j \in C$ (C is the raw polyhedron), we check the minimum distance $d(L, H_j)$ from L to the hyperplane of H_j. If $d(L, H_j) < r_r$, we adjust the normal of the hyperplane such that $d(L, H'_j) = r_r$ (H'_j is the adjusted halfspace). The hyperplane of the new halfspace H'_j also passes through the intersection point of H_j with the dilated ellipsoid [Fig. 9(e)].

C. Trajectory Optimization

In this section, we introduce the approach to generate minimum snap trajectories using the generated SFC. We adopt the similar formulation of trajectory optimization in our previous work [9]. Assume the SFC contains n convex polyhedra, the whole trajectory is composed of n polynomials and the i-th polynomial is inside the i-th polyhedron C_i. Thus, the convex optimization for minimum snap trajectories can be formed as a QP with constraints for the robot's starting and ending states as

$$\begin{aligned} \arg\min_{\Phi} \; & J = \sum_{i=0}^{n-1} \int_0^{\Delta t_i} \left| |\frac{d^4}{dt^4}\Phi_i(t)\right| |^2 dt \\ s.t \; & \frac{d^k}{dt^k}\Phi_i(\Delta t_i) = \frac{d^k}{dt^k}\Phi_{i+1}(0), \quad k = 0 \dots 4 \\ & \mathbf{A}_i^T \Phi_i(t) < \mathbf{b}_i \end{aligned} \tag{3}$$

Here the matrices $\mathbf{A}_i, \mathbf{b}_i$ correspond to the i-th polyhedron C_i. And the trajectory $\Phi(t)$ is composed as

$$\Phi(t) = \begin{cases} \Phi_0(t - t_0) & t_0 \le t < t_1 \\ \Phi_1(t - t_1) & t_1 \le t < t_2 \\ \vdots & \\ \Phi_{n-1}(t - t_{n-1}) & t_{n-1} \le t < t_n \end{cases} \tag{4}$$

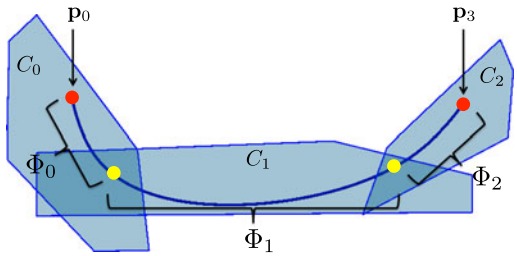

Fig. 10. Example trajectory has three polyhedra C_i and each segment Φ_i is confined to be inside its corresponding polyhedron. The red start and end points are confined to be at those locations and the yellow knot points are only constrained to be continuous and are allowed to vary within the intersection of adjacent pairs of polyhedra.

Here we use piece-wise polynomials as [17] to describe the trajectory $\Phi(t)$. Thus, $\Phi_i(t)$, $\frac{d}{dt}\Phi_i(t)$, $\frac{d^2}{dt^2}\Phi_i(t)$, $\frac{d^3}{dt^3}\Phi_i(t)$ indicate the desired position, velocity, acceleration and jerk at time t, which are the input for the non-linear controller [15] to calculate desired force and momentum for controlling the quadrotor. Δt_i in above equations refers to time of each polynomial as $\Delta t_i = t_{i+1} - t_i$. Fig. 10 shows an example of piece-wise polynomial trajectory that is confined by the SFC.

The estimation of Δt_i or *Time Allocation* significantly affects the resulting trajectories. As every SFC contains a valid path P, the naive *Time Allocation* method is to map this P into time domain using trapezoid velocity profile. Solving (3) with initial *Time Allocation* may result in trajectories with large velocity, acceleration or jerk that exceed the maximum thresholds of the MAV. Similar to [6], we modify Δt_i according to the (5) to adjust the *Time Allocation* such that the final trajectory generated using $\Delta t'_i$ can be followed by the robot. Denote the maximum velocity, acceleration and jerk of the generated trajectory as $v_{\max}, a_{\max}, j_{\max}$ and the corresponding thresholds as $\bar{v}_{\max}, \bar{a}_{\max}, \bar{j}_{\max}$. The unit 1 is used to prevent modifying a proper *Time Allocation*.

$$\Delta t'_i = \max\left\{1, \left(\frac{v_{\max}}{\bar{v}_{\max}}\right), \left(\frac{a_{\max}}{\bar{a}_{\max}}\right)^{\frac{1}{2}}, \left(\frac{j_{\max}}{\bar{j}_{\max}}\right)^{\frac{1}{3}}\right\} \Delta t_i \tag{5}$$

In our optimization process, we use a sample-based method to confine each polynomial, the details for which can be found in [2]. Also, we always assume a static end state with zero velocity, acceleration and jerk to ensure the flight safety.

D. Receding Horizon Planning

For navigation of the MAV in an unknown environment with local sensing, we use *Receding Horizon Planning* (RHP) to continuously generate trajectories until the robot reaches the

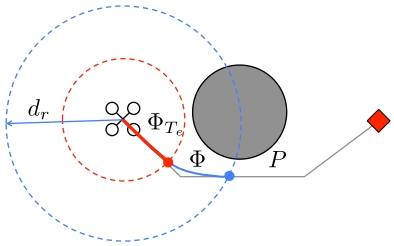

Fig. 11. *Receding Horizon Planning*. The planned path P goes to the goal (red diamond) directly. However, for generating trajectory Φ, we only plan to the boundary point according to the *planning horizon* d_r. The execution trajectory $\Phi(T_e)$ is bounded in the red circle according to the *execution horizon* T_e.

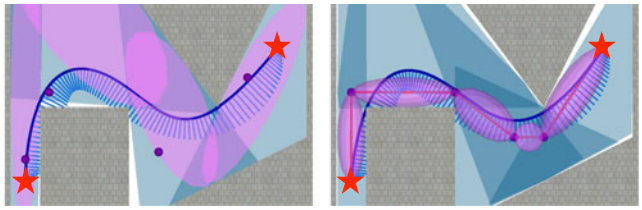

Fig. 12. Comparing our convex decomposition approach with IRIS. Red stars point out the start and goal. The generated trajectories are very similar, even though using two different *Safe Flight Corridors*. The light blue short lines that are perpendicular to the trajectory show the speeds at corresponding positions. (a) IRIS. (b) SFC.

final goal. As mentioned in the Section. I, the RHP is a variant of *Receding Horizon Control* where people solve a optimal control problem over a fixed future time interval [18], [19]. Instead of solving for a fixed time interval, we define the *planning horizon* to be the longest distance d_r that is restricted by sensing range in our receding horizon framework. Once we plan a path from start to goal, we only use a portion of this path with radius d_r of the robot to generate the trajectory Φ. The robot only executes Φ for a short period which we call the *execution horizon* T_e and thus the starting state for trajectory generation in next re-planning epoch is determined by $\Phi(T_e)$. We select T_e such that the time for generating a trajectory is guaranteed to be less than T_e so that the robot is able to follow a new trajectory once it finishes executing the current trajectory Φ. In other words, we start generating the trajectory for the next epoch when the robot is executing the trajectory at the current epoch. Since the execution time T_e is bigger than the time it takes for generating a trajectory, the robot is always able to transit to track a new trajectory when it finishes executing the current one. Fig. 11 shows the example of RHP.

In certain cases, if the planner is not able to find a path or the trajectory optimization fails due to a bad *Time Allocation*, the trajectory in the next re-planning epoch is not achievable. We utilize the stopping policy as described in [13] to make the robot come to a stop if the failure happens and after the robot stabilizes itself, we continue searching for a new trajectory using the same trajectory generation pipeline. We are able to plan trajectories in either a global or local map but for our experiments we use a local map. A local map is built using the last few sensor readings while a global map requires a full SLAM solution to correct for drift in state estimation. Compared to the global map, a local map is easier to achieve and is sufficient for obstacle avoidance, but the lack of global information makes the planner globally incomplete and susceptible to dead-end like environments.

III. Analysis

A. Comparison with IRIS

The existing algorithms for generating the collision-free convex region [5], [14] requires a proper selection of seeds and a geometric representation of obstacles which is hard to get from real sensor data. In their process (IRIS), solving the maximum ellipsoid through convex optimization takes a long time. For the map shown in Fig. 12, the IRIS algorithm takes around 110 ms while our algorithm only requires 4.8 ms. In fact, the selection of seeds for growing ellipsoids in IRIS is non-trivial, which also makes it harder to run IRIS for decomposition in real-time.

B. Run Time Analysis

We use four different maps to test the run time of our algorithm by generating hundreds of trajectories through them. The four maps are named as 'Random Blocks', 'Multiple Floors', 'The Forest' and 'Outdoor Buildings'. We sample goals at certain density in each map and manually select a start. Fig. 13 shows these maps and generated results. These maps are selected because they are typical for different environments encountered in the real world (namely 2.5-D, fully 3-D, randomly scattered complex obstacles and real-world data).

To evaluate the computational expense of our algorithm, we split the whole trajectory generation into three parts: path planning, convex decomposition and trajectory optimization. Table I indicates the time cost for each component when generating trajectories as shown in Fig. 13 on an i7-4800MQ processor. For path planning, we compared two different methods: A* and JPS to show the impact on run time by using JPS. As can be seen from the results, we are able to generate trajectory under a few hundred milliseconds which is sufficient fast for re-planning at 2–3 Hz.

C. Completeness

In this subsection, we discuss the algorithmic completeness within the local map: whether a trajectory will be found if one exists up to the resolution of the map. Since construction of SFC starts with line segments, it will at least produce a set of convex regions that includes those line segments. In this case, the feasible set of the optimization always contains the solution where the trajectory Φ is polynomial with static starting and ending states. For other cases where the initial non-static dynamics cause the failure of the trajectory optimization, the vehicle will either follow the existing collision-free trajectory or execute a stopping policy. Eventually, the vehicle will stop in a hover mode and from that static state we can always generate a trajectory if there exists a path to the final goal. In sum, our algorithm is complete since the path planning algorithm we use is complete. When using the global map (for example, Fig. 13), the completeness is guaranteed. However, if we only have local maps, the global completeness is impossible to achieve and the robot may get trapped in dead-end.

D. Flight Speed

In this section, we analysis the speed of the autonomous flight through non-dimensional parameters. We describe an MAV model by the maximum acceleration $\bar{a}_{\max}$ (constrained by the vehicle thrust to weight ratio) and the maximum velocity $\bar{v}_{\max}$

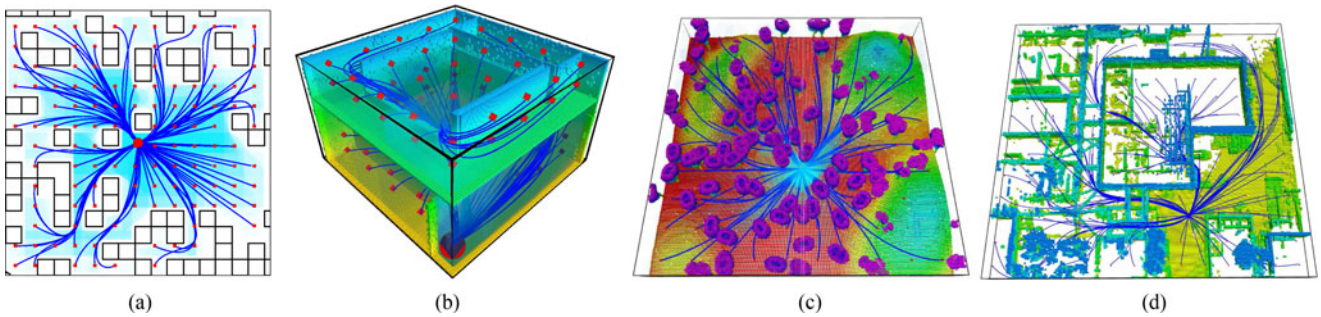

Fig. 13. Generate trajectories from a start (big red ball) to sampled goals (small red balls) in different maps. The blue curves are generated trajectories, cyan region is the overlapped SFC. (a) Random blocks. (b) Multiple floors. (c) The forest. (d) Outdoor buildings.

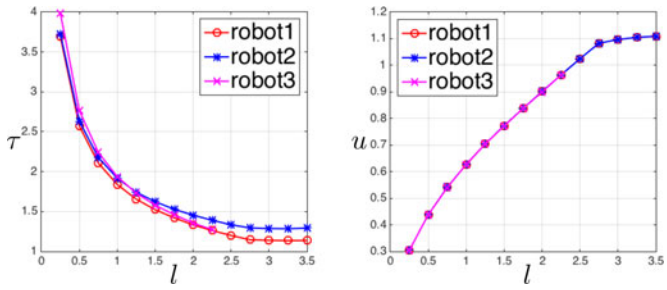

Fig. 14. Non-dimensional analysis for 3 different quadrotors while keeping t_e fixed at 0.1 and changing l: for robot 1, $\bar{v}_{\max} = 20\,\mathrm{m/s}, \bar{a}_{\max} = 10\,\mathrm{m/s^2}$; for robot 2, $\bar{v}_{\max} = 10\,\mathrm{m/s}, \bar{a}_{\max} = 5\,\mathrm{m/s^2}$; for robot 3, $\bar{v}_{\max} = 5\,\mathrm{m/s}, \bar{a}_{\max} = 5\,\mathrm{m/s^2}$. The total time for reaching the goal τ and the maximum speed u goes to 1.1 with increasing l (due to sample-based method we use for trajectory optimization, the maximum speed will exceed the actual bound by a small amount), which means the longer *planning horizon* leads to a faster flight.

(bounded by air drag). These two parameters reflect how fast an quadrotor can travel. For different platforms, we usually have different $\bar{a}_{\max}, \bar{v}_{\max}$ values due to their various hardware configurations. The planning horizon d_r (limited by the sensing range) and execution horizon T_e (limited by the on-board computation power) are two independent variables that affect the flight speed of the vehicle in RHP. They can be non-dimensionalized through normalization as:

$$l = \frac{2\bar{a}_{\max}}{\bar{v}_{\max}^2} d_r, \quad \tau_e = \frac{\bar{a}_{\max}}{\bar{v}_{\max}} T_e \tag{6}$$

The flight speed can be evaluated using two parameters: total time for reaching a goal T and the max speed $v_{\max}$. Suppose the total distance is d_{goal}, we are able to evaluate the nominal flight time and maximum speed using the notation as:

$$\tau = \frac{\bar{v}_{\max}}{d_{\text{goal}}} T, \quad u = \frac{v_{\max}}{\bar{v}_{\max}} \tag{7}$$

We plot the test results from using three different robots in simulation using these non-dimensional parameters (Fig. 14). We can conclude that fast flight can be achieved through setting a large planning horizon. However, in the actual experiments, the planning horizon is limited by the sensing range and won't increase the flight speed after a certain threshold. The execution horizon is also limited by the on-board computation, for example in Table I the max time cost for re-plan takes up to $0.47\,\mathrm{s}$ which places a lower bound on T_e.

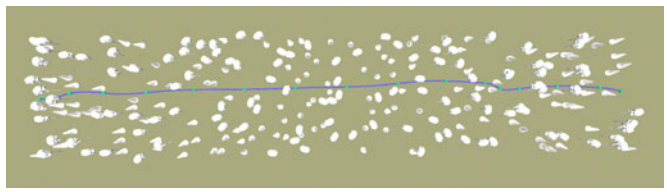

Fig. 15. 400 trees are randomly placed onto a 200×40 m square. The RHP planning horizon is 50 m, execution horizon is 1 s. Blue curves show the robot trajectory from one end to the other. Green dot show the start position of each re-plan.

To test high speed obstacle avoidance, we simulate environments by randomly scattering N convex obstacles inside a region. A typical environment is shown in Fig. 15. With a simulated Velodyne Puck VLP-16 of $40\,\mathrm{m}$ sensing range, the robot is able to achieve a max speed of $19.2\,\mathrm{m/s}$ in this forest and reach the goal 200 m away in 14.3 s.

IV. Experimental Results

We apply the proposed navigation pipeline on the quadrotor platform shown in Fig. 1. We use a stereo version of the MSCKF algorithm [20] for state estimation and a Velodyne VLP-16 to build a local map. All the computation is performed on an on-board Intel NUC computer (dual-core i7). Fig. 17 shows several experiments in the outdoor scenario where the robot has zero prior knowledge about the environment. Given a goal with respect to initial robot position, our system can successfully reach the goal and come back without hitting any obstacle. The vehicle travels at speeds up to $5\,\mathrm{m/s}$ for the runs shown in Fig. 17. In test 1, the robot successfully avoids trees and bushes with complicated 3-D geometries. In test 2, since the forest is dense the robot decides to fly around it instead of flying through it. In test 3, the robot avoids trees, forests and buildings, the total distance traveled by the robot is around 1 km. Our trajectories are smooth and constrained by thresholds on velocity, acceleration and jerk which helps to decrease the error in vision-based state estimation: the general drift in position after coming back to the start position is less than 1%.

As we set the maximum acceleration to be relatively small ($3\,\mathrm{m/s^2}$), the robot is able to closely track the generated trajectories. Fig. 16 shows the performance of the controller during test 1 and we can see that the errors are smaller than 0.2 m in position.

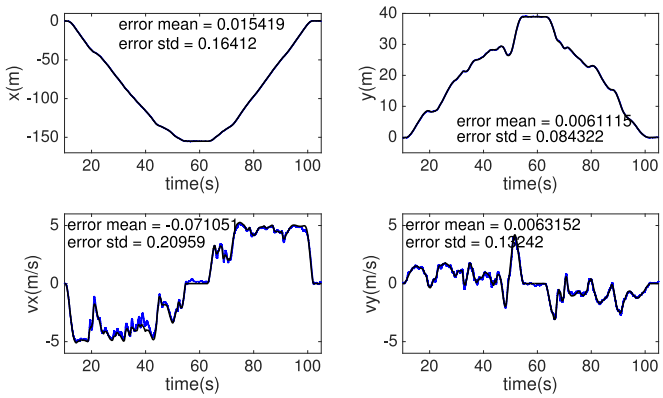

Fig. 16. State estimation *vs* desired command for test 1. The actual robot state is marked blue, while the desired command is marked black.

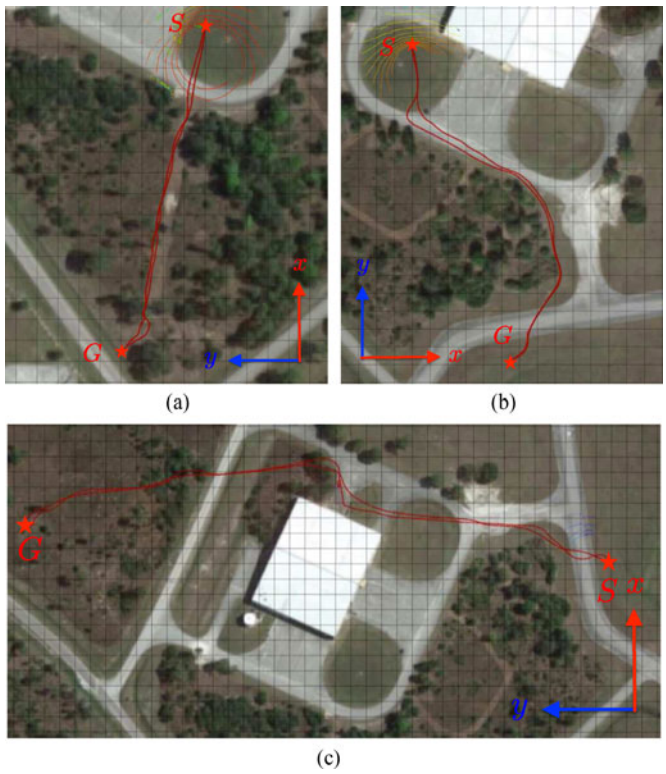

Fig. 17. Outdoor experiments. The grid cell size is $10\ \mathrm{m} \times 10\ \mathrm{m}$. The maximum speed is set to be $5\ \mathrm{m/s}$ while we also limit the maximum acceleration as $3\ \mathrm{m/s^2}$. The 2-D axes shows the direction of $x - y$ axes, the origin is located at the start point marked as a read star denoted by S. (a) Test 1. Goal $(-155, 39)$. (b) Test 2. Goal $(46, -184)$. (c) Test 3. Goal $(25, 384)$.

V. Conclusion

High-speed autonomous navigation is a challenge for MAVs because of (a) the constraints on dynamics that have to be incorporated into motion planning; (b) the limited computational resources for planning; (c) the limited sensor sensing range because of which the robot only has access to a local map of the world. In this letter, we described a trajectory generation algorithm that derives dynamically-feasible, collision-free trajectories in real time based only on on-board sensing and computation, and updates these trajectories in real time as fresh information becomes available from its sensors under the framework of *Receding Horizon Planning*. We studied the trade-offs between speed and safety and the effects of such parameters as (a) T_e, the *execution horizon*, (b) l, the sensor sensing range in the environment for different quadrotors. The study of the whole system and balance between the subsystems will identify the limiting factors hindering the speed of navigation and guide future research towards mediating these factors.

References

[1] M. J. Van Nieuwstadt and R. M. Murray, "Real time trajectory generation for differentially flat systems," 1997.
[2] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. 2011 IEEE Int. Conf. Robot. Autom.*, 2011.
[3] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Proc. 2012 IEEE Int. Conf. Robot. Autom.*, 2012.
[4] K. F. Culligan, "Online trajectory planning for UAVs using mixed integer linear programming," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2006.
[5] R. Deits and R. Tedrake, "Efficient mixed-integer planning for UAVs in cluttered environments," in *Proc. 2015 IEEE Int. Conf. Robot. Autom.*, 2015.
[6] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Proc. Int. Symp. Robot. Res.*, 2013.
[7] J. Chen, T. Liu, and S. Shen, "Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments," in *Proc. 2016 IEEE Int. Conf. Robot. Autom.* IEEE, 2016, pp. 1476–1483.
[8] F. Gao and S. Shen, "Online quadrotor trajectory generation and autonomous navigation on point clouds," in *Proc. 2016 IEEE Int. Symp. Safety, Security, Rescue Robot.*, Oct. 2016, pp. 139–146.
[9] S. Liu, M. Watterson, S. Tang, and V. Kumar, "High speed navigation for quadrotors with limited onboard sensing," in *Proc. 2016 IEEE Int. Conf. Robot. Autom.* IEEE, 2016, pp. 1484–1491.
[10] 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.
[11] S. Karaman and E. Frazzoli, "High-speed flight in an ergodic forest," in *Proc. 2012 IEEE Int. Conf. Robot. Autom.,* IEEE, 2012, pp. 2899–2906.
[12] K. M. Seiler, S. P. Singh, S. Sukkarieh, and H. Durrant-Whyte, "Using lie group symmetries for fast corrective motion planning," *The Int. J. Robot. Res.*, vol. 31, pp. 151–166, 2011.
[13] M. Watterson and V. Kumar, "Safe receding horizon control for aggressive MAV flight with limited range sensing," in *Proc. 2015 IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2015.
[14] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic Foundations of Robotics XI*. Berlin, Germany: Springer, 2015, pp. 109–124.
[15] T. Lee, M. Leoky, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE (3)," in *Proc. 49th IEEE Conf. Decis. Control*. IEEE, 2010, pp. 5420–5425.
[16] D. D. Harabor *et al.*, "Online graph pruning for pathfinding on grid maps," in *Proc. 25th AAAI Conf. Artif. Intell.*, 2011.
[17] D. W. Mellinger, "Trajectory generation and control for quadrotors," Ph.D. dissertation, Univ. Pennsylvania, Philadelphia, PA, 2012.
[18] J. Bellingham, A. Richards, and J. P. How, "Receding horizon control of autonomous aerial vehicles," in *Proc. 2002 Am. Control Conf.*, vol. 5, 2002, pp. 3741–3746.
[19] T. Schouwenaars, É. Féron, and J. How, "Safe receding horizon path planning for autonomous vehicles," in *Proc. Annu. Allerton Conf. Commun. Control Comput.*, vol. 40, no. 1, The University, 2002.
[20] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *Proc. 2007 IEEE Int. Conf. Robot. Autom.* IEEE, 2007, pp. 3565–3572.