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

High Speed Navigation For Quadrotors With Limited Onboard Sensing

Sikang Liu, Michael Watterson, Sarah Tang, and Vijay Kumar[1]

Abstract— **We address the problem of high speed autonomous navigation of quadrotor micro aerial vehicles with limited onboard sensing and computation. In particular, we propose a dual range planning horizon method to safely and quickly navigate quadrotors to specified goal locations in previously unknown and unstructured environments. In each planning epoch, a short-range planner uses a local map to generate a new trajectory. At the same time, a safe stopping policy is found. This allows the robot to come to an emergency halt when necessary. Our algorithm guarantees collision avoidance and demonstrates important advances in real-time planning. First, our novel short range planning method allows us to generate and re-plan trajectories that are dynamically feasible, comply with state and input constraints, and avoid obstacles in real-time. Further, previous planning algorithms abstract away the obstacle detection problem by assuming the instantaneous availability of geometric information about the environment. In contrast, our method addresses the challenge of using the raw sensor data to form a map and navigate in real-time. Finally, in addition to simulation examples, we provide physical experiments that demonstrate the entire algorithmic pipeline from obstacle detection to trajectory execution.**

I. Introduction

Micro Aerial Vehicles (MAVs), in particular quadrotors, have become promising platforms for many applications. As a result, many control, planning, and perception methods have been proposed for these systems. However, these problems have mostly been approached in isolation. Planning and control algorithms for aggressive maneuvers are often demonstrated within a motion capture space where the perception challenge is eliminated, and many perception solutions have only been validated with low-speed flight in environments with sparse obstacles. In contrast, this paper addresses the problem of fast flight through unknown environments, where the vehicle must both sense and localize obstacles as well as plan and execute trajectories in real-time.

The design of geometric controllers [1] for quadrotors has allowed for the precise tracking of aggressive trajectories, and as a result, a number of planning algorithms have been proposed and successfully validated on experimental platforms. For example, Mellinger et al. [2] [3] and Deits and Tedrake [4] formulate the problem as a Mixed Integer Quadratic Program (MIQP). However, the computation time required to find a solution makes this approach impractical for real-time planning. Computationally faster techniques include those of Hehn and D'Andrea [5], which generates minimum time trajectories using optimal control techniques, and Richter et al. [6], which formulates the trajectory generation problem as a Quadratic Program (QP) that minimizes the integral of the trajectory's snap squared. However, these methods have only been validated in scenarios where the environment is completely known *a priori*. In particular, Hehn and D'Andera [5] assume the environment is obstacle-free and Richter et al. [6] assume an OctoMap [7] representation of the environment is available.

Past works have also addressed the problem of navigation in unknown or partially known environments. Pivtoraiko et al. [8] generate a set of motion primitives, which they use to incrementally re-plan towards a goal. Bellingham et al. [9] solve a Mixed Integer Linear Program over a receding horizon to find trajectories that incrementally move towards a goal, incorporating a collision avoidance heuristic. Watterson and Kumar [10] propose a receding horizon control policy (RHCP), which offers guarantees of algorithm completeness and collision avoidance. While these algorithms address demonstrate real-time generation of new trajectories as information about the environment becomes known, they assume the obstacle detection problem is solved and the robot can instantaneously query the properties of any obstacle within in a given sensing radius, such as location and geometry. In reality, creating and maintaining a map from raw sensor data is a nontrivial step.

In this work, we present an algorithm for fast navigation through unstructured and unknown environments. We define fast navigation as finding a high-speed trajectory for the MAV under dynamic constraints, limited sensor range, and limited computational capabilities. In particular, our algorithm address three novel challenges. First, we detect obstacles and create a map representation of the environment on-the-fly, explicitly incorporating the computational demands of translating raw sensor data to a map for trajectory planning. Second, we propose a novel short range planning policy that includes a frontier-based method for finding promising paths towards the robot's final goal and fast convex segmentation of a provided map that allows for real-time generation of optimal trajectories which accommodate the vehicle's dynamics. Finally, we present simulation examples and experimental results that demonstrate the complete algorithmic pipeline from perception of obstacles to execution of a designed trajectory.

This paper will proceed as follows. Section II introduces the overall algorithm. Section III describes the perception, trajectory generation, and control protocols. Section IV discusses the algorithm's safety and optimality guarantees. Section V presents simulation examples and experimental validation. Conclusions are presented in Section VI.

[1]S. Liu, M. Watterson, S. Tang, and V. Kumar are with the General Robotics, Automation, Sensing & Perception (GRASP) Laboratory, University of Pennsylvania, Phildadelphia, PA 19104, USA {sikang, wami, sytang, kumar} @seas.upenn.edu

II. Algorithm Overview

A. Problem Formulation

We model the quadrotor as a rigid body whose configuration is an element of $SE(3)$. Since the quadrotor is a differentially flat system [2], we can use its flat outputs, $\mathbf{x} = [x \quad y \quad z \quad \psi]^T$, ψ is the yaw of the robot, and their derivatives to completely represent the robot's state. Assuming ψ remains constant while executing a trajectory, we denote the robot state at time t:

$$\mathbf{X}_t = [\mathbf{x}^T \quad \dot{\mathbf{x}}^T \quad \ddot{\mathbf{x}}^T \quad \dddot{\mathbf{x}}^T]^T = [\mathbf{x}^T \quad \mathbf{v}^T \quad \mathbf{a}^T \quad \mathbf{j}^T]^T \qquad (1)$$

Here, $\mathbf{v},\ \mathbf{a},\ \mathbf{j} \in \mathbb{R}^3$ represent the robot's velocity, acceleration and jerk, respectively. A goal is defined as a position $\mathbf{g} \in \mathbb{R}^3$. Given a goal in an unknown environment, our problem is to navigate the robot to this goal as fast as possible under imposed maximum velocity, acceleration and jerk limits, denoted $v_{max},\ a_{max},\ j_{max}$, respectively. We model both the vehicle dynamics and perception sensor while aiming to achieve real-time computation. As a result, the speed of our robot's trajectories depends on the available computation time. Therefore "as fast as possible" does not necessarily mean computing globally time optimal trajectories.

B. Control Policy

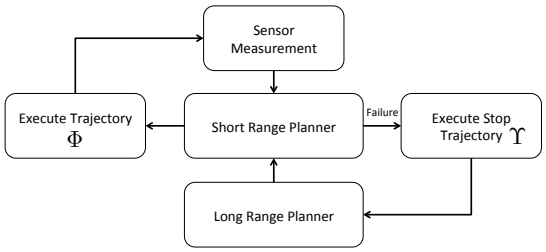

Fig. 1. Illustration of planning paradigm. The short range planner takes most recent sensor measurment of the environment and is running repeatedly until the robot reached the goal or a failure is detected. The failure triggers execution of stop trajectory Υ and the long range planner.

In this work, we adopt the planning paradigm similar to [10], which is illustrated in Fig. 1 and 2. Under this paradigm, planning in two different time horizons guarantees algorithm completeness while moving as fast as possible. A short range planner can find local trajectories, denoted Φ_τ in Fig. 2, that incrementally move the quadrotor towards its goal. When the short range planner fails, we can execute a stopping policy, denoted Υ_τ, to safely stop the quadrotor before it collides with an obstacle and switch to a long range planner. In our algorithm, we use a novel short range planning technique that processes raw sensor data and generates safe trajectories in real time.

III. Short Range Planner

When navigating in unknown environments with limited onboard computation, it is infeasible to use an entire map to plan at each time step. It is also often unnecessary to do so, because we only require avoidance of local obstacles in the immediate horizon. However, it is also easy to construct environments in which a local planner will fail.

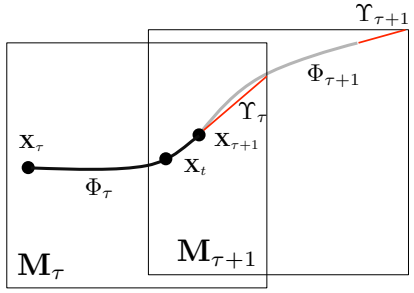

Fig. 2. Illustration of the planning paradigm. Black rectangle indicates the map boundary in each planning epoch. Let $\mathbf{X}_t$ denotes the state of robot at time t, $\mathbf{X}_\tau$ and $\mathbf{X}_{\tau+1}$ indicate the intial state of trajectory Φ_τ (black spline) and $\Phi_{\tau+1}$ (gray spline) respectively and $\mathbf{X}_{\tau+1}$ is the end state of Φ_τ too. During exectuing current trajectory Φ_τ, the robot starts searching for $\Phi_{\tau+1}, \Upsilon_{\tau+1}$ at $\mathbf{X}_t$ for next iteration before it finishes Φ_τ. If $\Phi_{\tau+1}, \Upsilon_{\tau+1}$ are not found, it will execute Υ_τ (denoted by the red line) to stop.

Traditional planning approaches have combined exploration and SLAM [11] to allow the robot plan collision-free paths with a temporary goal and an accumulated map. The proposed short range policy uses a similar idea, but instead of having the robot explore the entire environment, we define a temporary goal in current known map which is most likely to lead the robot to the final desired goal. Based on the local map and a path found with a graph search algorithm, we construct a convex decomposition of the local map and use that to generate a safe trajectory. An overview of this short range planning process is illustrated in Fig. 3 and detailed in the following subsections.

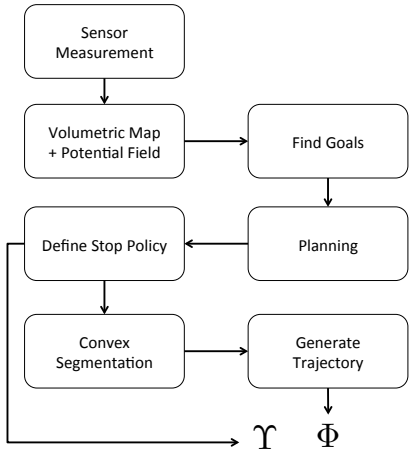

Fig. 3. Short range policy overview. This diagram shows the data flow inside short range planner in Fig. 1. In the end, the trajectory Φ and corresponding stop trajectory Υ are generated.

We illustrate the algorithm using the sample problem in Fig. 4. The desired goal $\mathbf{g}$ is outside of M, and robot's current location is at p_1. For clarity, we describe the algorithm with planar illustrations, but the algorithm applies to and is implemented in 3D.

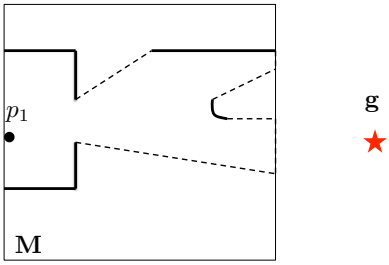

Fig. 4. Robot planning to a goal using its current local map $\mathbf{M}$. We define the current robot position p_1 as the origin in $\mathbf{M}$, the length and width of $\mathbf{M}$ are fixed. Solid black lines indicate obstacles detected by the onboard sensor while dashed lines indicate the frontier edge.

A. Environment Representation and Sensor Model

In order to generate collision-free paths in obstacle-filled spaces, we require a suitable and practical representation of the environment. Previous works [3] [10] [12] require a geometric representation of obstacle surfaces to represent obstacle avoidance constraints as linear inequalities of the form $Ax \leq b$. While it is straightforward to use these inequality constraints to optimize a safe trajectory, it is non-trivial to find the normal and boundary of obstacle surfaces from real sensor data. Thus, these works either assume the environment is known or abstract perfect perception capabilities that can automatically detect obstacles' geometries. As a result, their methods cannot be directly used in the real-world settings with a noisy sensor.

An alternative is to create volumetric representations of the environment, such as an occupancy grid map and its variants. This type of map can be built by LIDAR, vision, or depth sensors. Their structure inherently provides a partition of the free space of the environment along with a connected graph through which we can search. In past work, an OctoMap has been used when creating a large scale or high resolution map to save memory for storage [7]. However, it is slow to build OctoMap and since we plan in the local map, we won't run into the memory storage issue. As a result, we use a uniform resolution volumetric occupancy grid map in our algorithm. Such a map will be referred to as an occupancy grid map when it is a 2D map and a voxel map when it is a 3D map.

Both LIDAR and vision sensors can generate a point cloud which can be used to create such a volumetric map. For each cell of the volumetric map, there exists three states: occupied, free, or unknown. In our algorithm, the robot can only travel through free cells. While this restriction is conservative, it will guarantee the vehicle's safety. Rather than accumulating consecutive sensor measurements to form a global map, we only use the latest sensor measurement to create a local map. This eliminates the need to handle mapping errors from bad localization in state estimation and decreases the map computation time to allow for real-time re-planning. Despite the use of a local map, our method for choosing frontier points allows us to reach the goal in most circumstances. In randomly generated environments with cylindrical obstacles, a local planner can be used almost everywhere [13].

To make the planned trajectories safer, we calculate a discretized collision cost map corresponding to the volumetric map. For each cell $\mathbf{p}$ in the volumetric map, we find the distance to the obstacle closest to the cell, $h(\mathbf{p})$, and assign the cell a collision cost $\phi(\mathbf{p})$ according to Eq. 2. Here, R_{robot} is the robot radius and R_d is a constant coefficient and k_{rep} is a predefined constant to scale the value of collision cost. A typical occupancy grid map and its corresponding collision cost map are shown in Fig. 5

$$\phi(\mathbf{p}) = \begin{cases} \infty & , h(\mathbf{p}) < R_{robot} \\ k_{rep}(\frac{R_d - h(\mathbf{p})}{R_d - R_{robot}})^2 & , R_{robot} < h(\mathbf{p}) < R_d \\ 0 & , h(\mathbf{p}) > R_d \end{cases} \quad (2)$$

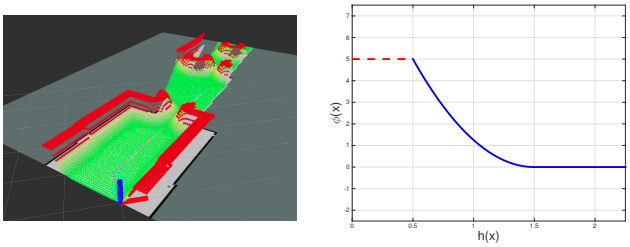

Fig. 5. Left: Occupancy grid map and collision cost map. The robot travels through the free cells in the occupancy grid map in planning, so we ignore the unknown area and only assign collision cost ϕ for free space. Right: Plot of collision cost $\phi(\mathbf{p})$ versus distance to obstacle $h(\mathbf{p})$.

B. Frontiers

Since we plan using only a local map from the most recent sensor measurement, in many cases, the desired goal $\mathbf{g}$ will be outside of the current map. To guarantee safety, we only allow the robot to pass through the free space of the current volumetric map. Consequently, when the desired goal is in the unknown space, we need to find a set of candidate goals gs. On the other hand, if the goal $\mathbf{g}$ is in the current local map, we simply set $gs \rightarrow \mathbf{g}$.

Given a current 2D occupancy grid map, we identify all frontiers by searching for connected regions of cells which border free and unexplored space [14]. The typical frontier points are shown in Fig. 6. We group the cells defined as frontier along the same edge, the average of those cells in frontier group i is selected as a candidate goal g_i for planner according to its cost value $\phi(g_i)$ and the distance to robot position $\|g_i - p_1\|$.

In 3D, the current voxel map has many frontier voxels, making it more difficult to find frontier groups. Thus, instead of finding frontiers in the 3D voxel map directly, we slice the voxel map into multiple 2D occupancy grid maps at different heights and find the frontiers of each 2D map using the method previously described.

C. Path Search

We consider each voxel in the voxel map as a node and build a graph representation of the environment. We preform an $A^\star$ search on this graph to find a path through the voxels from the current robot location to each candidate goal $g \in gs$. In anticipation of real world noise, we want to minimize the

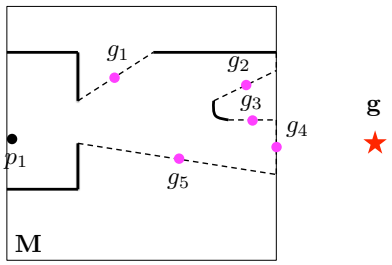

Fig. 6. Frontiers of the example occupancy grid map. $gs = \{g_{i,\ i=1,\ldots,5}\}$, where g_i is indicated by the purple dot and they are average point of each frontier groups (dashed edges).

times at which the robot moves close to obstacles. To this end, we use the constructed collision cost map to bias the robot from prolonged movement in free space close to walls by defining edge weights as given in Eq. 3. Here, $d(\mathbf{p}_i, \mathbf{p}_j)$ is the Euclidean distance between two connected cell $\mathbf{p}_i$ and $\mathbf{p}_j$. The effect of the collision cost ϕ is demonstrated in Fig. 7.

$$c(\mathbf{p}_i, \mathbf{p}_j) = d(\mathbf{p}_i, \mathbf{p}_j) + \phi(\mathbf{p_j}) \tag{3}$$

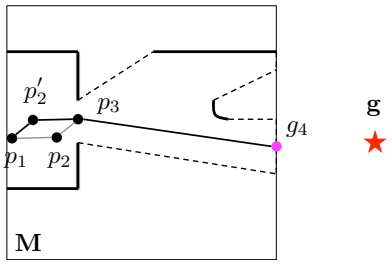

Fig. 7. To go to g_4, path $p_1 - p_2' - p_3 - g_4$ is considered to be safer than $p_1 - p_2 - p_3 - g_4$ since p_2' is at a better location compared to p_2 such that the robot is farhter away from walls.

We generate paths to each candidate goal in gs and sort the output paths in order of increasing distance between the candidate goal and the desired goal $\mathbf{g}$. We will later generate trajectories from each path in this order until we find a feasible one.

D. Stopping Policy

The path we obtain from the previous planning step goes directly to the best candidate goal g. However, there could be an obstacle close to the frontier point g that is not seen in the current map. The region behind frontier g will be explored when the quadrotor arrives at g, after executing its planned trajectory. In the worst scenario, an obstacle could be revealed to be right behind g. Thus, for guaranteed collision avoidance, we would have to set the desired velocity at g to be zero. However, having the robot come to a full stop at every short range planing iteration is too conservative and does not allow for fast flight to a given goal.

Instead, as illustrated in Fig. 8, we generate our trajectory to an alternate point, g', with final velocity v_f. Such that g' in the intersection of the circle of radius ds centered at g and the path P Where ds is defined as:

$$ds = R_{robot} + \frac{v_{max}^2}{2a_{max}} \tag{4}$$

When robot reaches g' with any $v_f \leq v_{max}$, it will be able to decelerate to a halt at a position g'' before g along a straight line. We refer to this stopping maneuver as a stop policy. To generate this stop policy, Υ, we use g' and predict a final position g'' according to velocity at g'. We adopt the fast algorithm in [6] to generate Υ with respect to g', g'', v_f.

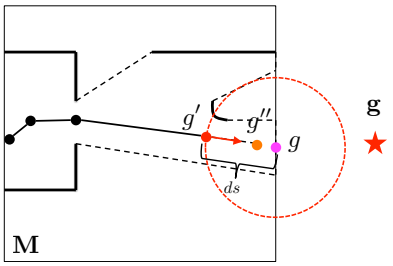

Fig. 8. Reducing path length by distance needed to stop. The robot needs length ds to stop, so the section $\overline{g'g}$ is removed from the path.

E. Convex Segmentation

Before generating the trajectory for a given path, we need to find a sequence of convex volumes to define the free space. We find these convex spaces by inflating the selected path with polygons in the free space such that the region inside these polygons guaranteed to be collision-free (Algorithm 1).

To do this, we first sample voxels along a line segment l of the path P. Along each of of voxels in this line, we use ray tracing to find the maximum interval which is known and collision free. The boundary of this each of these intervals S_i is a pair of points $v_i^{\pm}$. For pairs of adjacent voxels, we form a polygon which the convex hull of the four points $v_i^{\pm}, v_{i+1}^{\pm}$. We then check collinearity of corners of adjacent polygons and merge them if possible. Otherwise we add the first polygon to segmented list. Similar idea is developed into 3D to generate 3D polyhedrons.

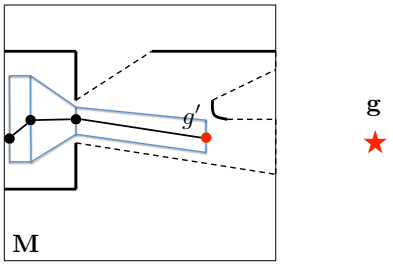

Fig. 9. The convex space, blue lines are the boundaries for each polygon.

F. Trajectory Generation

We use a similar trajectory optimization as [10] to find a minimum jerk trajectory within the convex corridor found in above section by using a basis (Eq. 5) to formulate (Eq. 6) as a quadratic program (Eq. 7). We first optimize without

Algorithm 1 Convex Segmentation

Input: collision cost map ϕ, path P
Output: polygons Bs
for Each line segment $l \in P$ **do**
  voxels: $vs \leftarrow$ Raytrace(l)
  for Each voxel $v_i \in vs$ **do**
    Raytrace line $r(v_i) \perp l$ which intersects v_i
    $S_i \leftarrow \{v_j \in r(v_i) \mid \phi(v_j) < \phi_{thr}$ and $\phi(v_j) \neq \phi_{unkown}\}$
    $v_i^{\pm} \leftarrow \partial S_i$
    $B_i \leftarrow$ ConvexHull$(v_i^{\pm}, v_{i+1}^{\pm})$
    if $v_{i-1}^{\pm}, v_i^{\pm}, v_{i+1}^{\pm}$ colinear **then**
      Combine B_{i-1} and B_i
    else
      $Bs \leftarrow \{Bs, B_i\}$
    end if
  end for
end for

any inequality limits imposed on the velocity, acceleration, and jerk. We impose the additional inequality constraint that the velocity of the end of the trajectory is within the cone of the sensor's view.

$$\tau_i(t) = \sum_j \alpha_{ij} \cdot b_j(t) \tag{5}$$

Starting from the initial trapezoidal segment time allocation in [10], we analytically compute the gradient of the cost function with respect to segment times to iteratively refine the time allocations as in [3]. Once the optimal trajectory times are found, we can re-weight the total time of the trajectory $T = \sum_i \Delta t_i$ to be aggressive as possible, while respecting the constraints on the velocity, acceleration, and jerk.

$$\begin{aligned} \arg\min_{\tau_i \ \forall i} \quad & \sum_i \int_0^{\Delta t_i} || \tfrac{d^3}{dt^3} \tau_i(t) ||^2 dt \\ s.t \quad & \tfrac{d^k}{dt^k} \tau_i(\Delta t_i) = \tfrac{d^k}{dt^k} \tau_{i+1}(0) \quad k = 0..3 \\ & \tau_i \in P_i \\ & \tau_0(0) = x_0 \\ & \tau_N(\Delta t_N) = x_f \end{aligned} \tag{6}$$

The cost function is non-convex with respect to the segment time allocation vector Δt. We use a trapezoidal time allocation with an iterative adjustment as in [2] to set times. The original time allocation plot is shown in Fig. 11

$$\begin{aligned} \min_{\alpha} \quad & \alpha^{\intercal} D \alpha \\ s.t \quad & A\alpha = b \\ & C\alpha \leq d \end{aligned} \tag{7}$$

Each trajectory segment is confined to be inside a convex polyhedron. We use either a uniform sampling of points along each trajectory segment to confine instead of continuum of the trajectory [2] or a Bezier basis for the $b_j(t)$ in Eq. 7 to confine the convex hull of the trajectory segments [15] .

The whole processes from Section III-A to III-F generates a collision-free trajectory in the safe corridor of current map. Through theses steps, we can find a valid trajectory to execute, otherwise the short term planner will throw a failure and trigger the stopping trajectory. This framework is depicted in Algorithm 2. A typical trajectory output is shown in Fig. 10.

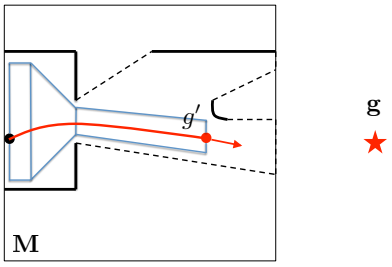

Fig. 10. Red spline is the final trajectory Φ we get from this process. Its end velocity is denoted by the red arrow. The stop trajectory is generated according to the end state of Φ.

Algorithm 2 Short range planner. In planning epoch τ, we are generating the trajectories for next epoch $\tau + 1$ (Fig. 2). Define the set of valid paths as $\mathcal{P}$. $\mathcal{P}$ is sorted according to section III-C.

Input: map $\mathbf{M}$, collision cost map ϕ, desired goal $\mathbf{g}$, initial position $\mathbf{p}_s$
Graph $\mathcal{G} = (\mathbf{M}, \phi)$
$\mathcal{P} \leftarrow \emptyset$
if $\mathbf{g} \in \mathbf{M}_{free}$ **then**
  $\mathcal{P} \leftarrow A^{\star}(\mathbf{p}_s, \mathbf{g}, \mathcal{G})$
else
  $gs \leftarrow$ Find Frontiers$(\mathbf{M})$
  for $g \in gs$ **do**
    P = $A^{\star}(\mathbf{p}_s, g, \mathcal{G})$
    $\tilde{P} \subset P$ is the path P shortened by the distance needed to stop by g
    $\mathcal{P} \leftarrow \{\mathcal{P}, \tilde{P}\}$
  end for
  $\mathcal{P} \leftarrow$ sort $\mathcal{P}$ according to Cost$(\tilde{P})$
end if

for $\tilde{P} \in \mathcal{P}$ **do**
  $\mathcal{B}_s \leftarrow$ Find Convex Segmentation$(\tilde{P})$ (Algorithm 1)
  $\Phi \leftarrow$ Trajectory Solver$(\mathbf{X}_s, \tilde{P}, \mathcal{B}_s)$
  $\mathbf{X}_e \leftarrow$ Find End State(Φ)
  $\Upsilon \leftarrow$ Find Stop Policy$(\mathbf{X}_e)$
  if Φ, Υ exist **then**
    $\Phi_{\tau+1}, \Upsilon_{\tau+1} \leftarrow \Phi, \Upsilon$
    break
  end if
end for

IV. Analysis

In this section, we will analyze the sub-optimality we trade for computational speed and discuss the safety of our system.

A. Sub-optimality

The problem of finding the optimal trajectory in the local map is not computationally tractable, because it requires computation of the exact convex segmentation of the available free space and optimal trajectory of every path in each homotopy class of paths from the current location to each frontier point. Instead, we compute a suboptimal trajectory that holds the following properties:

1) The trajectory's final state is the closest to the goal.
2) The trajectory's path is the safest.
3) The trajectory takes the least time to execute.

The first property ensures that the robot goes in the best possible direction towards the goal. The details of this process are described in Section III-B.

To satisfy the second property, we utilize planner described in Section III-C to find the path, where we take into account the cost of the path's cells to maximize safety.

Finally, the time allocation plotted in Fig. 11 may not be optimal because the time is assigned based on the path returned by $A^\star$. Even though the times are iteratively refined, the objective is non-convex with respect to the time allocation. Thus this optimization cannot be guaranteed to converge to a globally optimum solution.

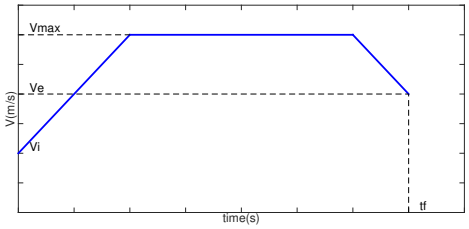

Fig. 11. Time allocation for path, denote robot initial speed as v_i, max speed as v_{max} and end speed as v_e.

B. Safety

System safety is maintained by several conservative assumptions. Firstly, by always generating a stopping policy during each planning step, the robot can always stop in the case that the short range planner cannot find a valid trajectory. Secondly, during the convex segmentation step, unknown space is treated as obstacles, so in the event that everything the robot cannot see is obstacles, it can still stop.

We also limit the maximum velocity based on limits on the acceleration of our platform and sensor range. Under the assumption that the robot must stop within distance d, can decelerate at $a_{\max}$ and takes δt seconds to process the sensor data, it can travel up to velocity v.

$$v = a_{\max}\left(\sqrt{\delta t^2 + 2\frac{d}{a_{\max}}} - \delta t\right) \tag{8}$$

The lateral acceleration of our platform is proportional to the sine of its pitch in the direction of travel. For control purposes we limit this angle and thus the acceleration. With this relation, we limit our velocity based on our sensor range.

V. Experimental Results

A. Simulation Results

We construct several Gazebo worlds with different obstacles to test our algorithm. In order to verify system's performance and safety guarantees, we construct three types of environments.

First, we demonstrate the ability of the stopping policy to guarantee safety. In Fig. 12, the final goal is behind a wall at the end of a long corridor. The quadrotor, initially sensing free space in front of itself, accelerates to its maximum velocity v_{max}. However, as soon it detects the wall, it executes its stop policy to safely halt in front of the wall before a collision occurs. This result can be seen in the video. With a sensor range of $4.5m$, maximum acceleration of $5m/s^2$, and prediction time $\delta t = 0.15s$, the quadrotor reaches a maximum velocity of $4m/s$. This is very close to the theoretical maximum velocity we can achieve according to Eq. 8, which in this case is $5.6m/s$.

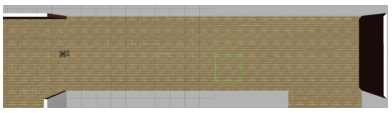

Fig. 12. Gazebo simulation environment type 1: long corridor with a wall in the end.

In the second type of environment, shown in Fig. 13, we place various 3D obstacles including table, cabinet, vertical pillar, window and horizon bar. The robot is able to avoid all these obstacles to successfully reach the goal at the end of corridor. In particular, the algorithm is able to adjust the speed of the robot to around $1m/s$ to accommodate the high density of the obstacles. The simulation results can be found in submitted video.

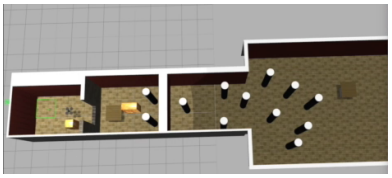

Fig. 13. Gazebo simulation environment 2. It verifies our algorithm's capability to avoid all kinds of obstacles.

Finally, we want to benchmark system performance with different obstacle densities. We create a total of 11 environments with varying obstacle densities, three examples of which are shown in Fig. 14, and test various maximum velocity and acceleration constraints. The required times to reach the given goal are plotted in Fig. 15. As expected, the flight time increases with the increasing obstacle density. However, from the second figure, we can see that maximum accelerations in a reasonable range result in similar flight times.

B. Hardware Experiments

Fig. 16 illustrates our experimental platform. We use an AscTec Pelican quadrotor with a 3.4 GHz dual-core i7 Intel

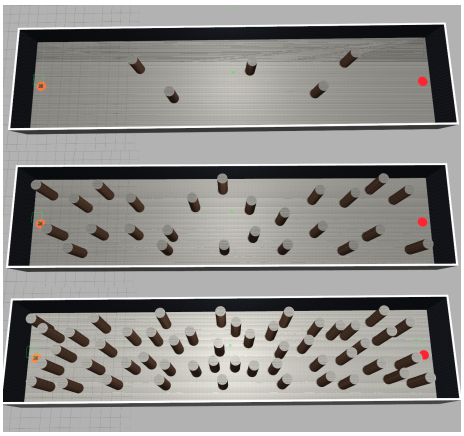

Fig. 14. We fill a $40m \times 10m \times 4m$ space with cylinders. From top to bottom, the number of pillars in the space are 5, 25, 50. The orange dots on the left indicate the robot start positions and the red dots on the right denote goal positions.

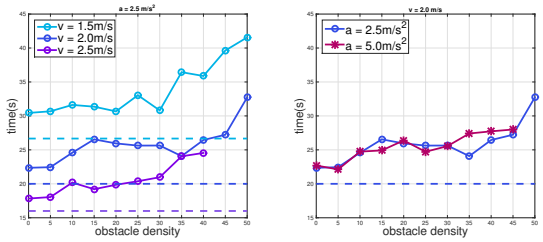

Fig. 15. Left figure: desired (maximum) speed at $1.5, 2.0, 2.5m/s$ respectively, the dashed line is the lower bound on time for traveling with corresponding speed for 40m and a maximum acceleration of $5.0m/s^2$. Right figure: maximum acceleration at $2.5, 5.0m/s^2$ respectively with desired speed of $2.0m/s$

NUC. A Primesense RGBD sensor is mounted in front for map creation and obstacle detection in 3D environments. The total weight of this platform is 1.5kg, giving it a thrust to weight ratio of 2.4.

Fig. 16. Asctec Pelican with Primesense RGBD sensor.

We use a non-linear controller in $SE(3)$ [16] to control the quadrotor and use a Qualisys [17] motion capture system for localization. Note that the motion capture system is used only for odometry measurements of the robot. All obstacle detection and map creation is done using the on-board Primesense RGBD sensor. The range of depth sensor is $4.5m$, and δt is set to be $0.15s$.

We demonstrate the capabilities of our system through two experiments, shown in Fig. 17 and Fig. 20.

Like the first simulation environment, the first experiment demonstrates the robot coming to a halt in front of a wall-like obstacle. We use this to demonstrate safety, even when the robot is moving near its maximum velocity. The maximum velocity is $3m/s$, with a maximum acceleration of $5m/s^2$. With these values, it theoretically takes $0.9m$ to stop, which is within the sensor range of 4.5m. The robot was able to successfully stop before hitting a wall created by a white curtain (Fig. 19), and its position and velocity profiles throughout the flight are shown in Fig. 18. During this experiment, the trajectory and sensor are visualized in Fig. 19.

Fig. 17. Desired goal is labeled as the red star in the left image, the quadrotor is facing forward to the goal, but in between there is a wall. The quadrotor cannot detect the wall until it gets close enough.

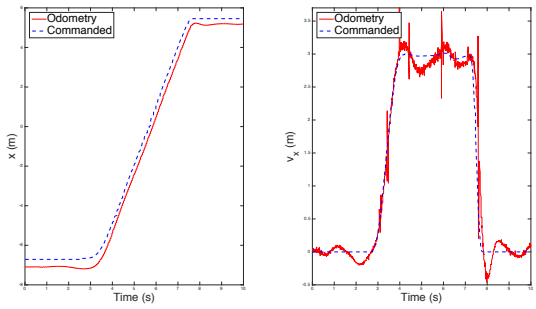

Fig. 18. The plot of trajectory command and quadrotor odometry in x-axis which is the direction picture from right to left in Fig. 17. The solid blue line is the desired, and the dashed read line is that measured by the Qualisys Sytem [17].

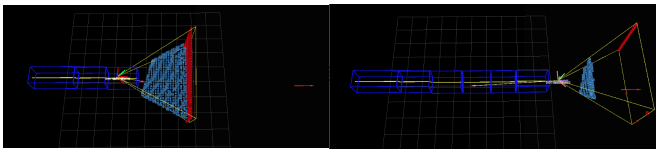

Fig. 19. Left image: Blue small boxes are the ground detected by depth sensor, red boxes are virtual points, red arrow shows the position of desired goal. Right image: the robot stopped in front the the wall, since it cannot find a valid trajectory to go around the obstacle to reach the goal.

We place two pillars in the second environment to show the collision-avoidance behavior of our algorithm. The robot is able to fly around the obstacles and reach the desired goal. The maximum speed in this experiment is $1m/s$.

VI. Conclusion

Our planning algorithm is able to generate, execute, and re-plan trajectories on computationally constrained hardware

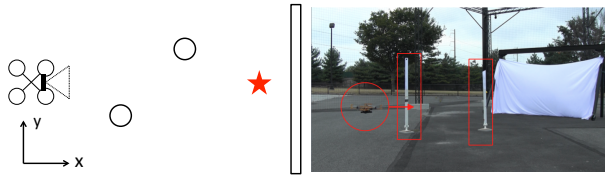

Fig. 20. Desired goal is labeled as the red star in the left image, the quadrotor is facing forward to the goal, but in between there are two pillars.

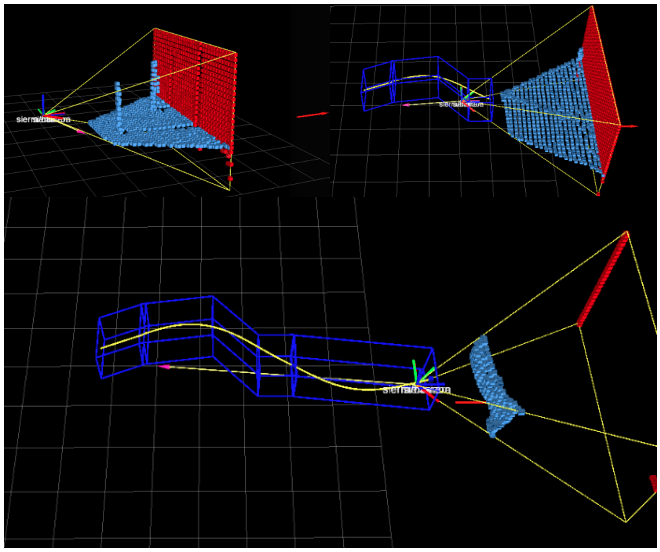

Fig. 21. Left upper image: initial state, the pillars are detected by depth sensor. Right upper image: the robot is tracking the trajectory (yellow spline). Bottom image: the robot follows the S-curve and reaches the goal, the wall is in front of the robot.

in real-time using simulated and real sensor data. We propose and implement a planning algorithm which imposes a collision-avoidance cost map on top of a probabilist occupancy representation of an incrementally observed environment. From this cost map, we find a frontier and a convex corridor to generate a optimal trajectory. Over a short range we also generate a stopping policy which can be executed in case of future failure of the local planner. In this failure mode, we use a long range planner to navigate around obstacles can trap any short range planner. We validate this algorithm in simulated environments, and outdoor test flights.

VII. Acknowledgments

We gratefully acknowledge the support of ARL grant W911NF-08-2-0004, ONR grants N00014-07-1-0829, and N00014-09-1-1051, NSF grant IIS-1426840, ARO grant W911NF-13-1-0350 and a NASA Space Technology Research Fellowship. Sarah Tang is supported by NSF Research Fellowship Grant No. DGE-1321851.

References

[1] T. Lee, M. Leok, and N. H. McClamroch, "Stable manifolds of saddle points for pendulum dynamics on s^2 and $so(3)$," in *IEEE Conference on Decision and Control*, Dec 2011, p. 39153921.

[2] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 2520 – 2525.

[3] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*, 2012, pp. 477–483.

[4] R. Deits and R. Tedrake, "Efficient mixed-integer planning for uavs in cluttered environments," in *Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 42–49.

[5] M. Hehn and R. D'Andrea, "Real-time trajectory generation for interception maneuvers with quadrocopters," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 4979–4984.

[6] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Proceedings of the International Symposium of Robotics Research (ISRR 2013)*, 2013.

[7] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[8] M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-uav motion replanning for exploring unknown environments," in *Proceedings of the 2013 IEEE International Conference on Robotics and Automation (ICRA)*, 2013, pp. 2452–2458.

[9] J. Bellingham, A. Richards, and J. P. How, "Receding horizon control of autonomous aerial vehicles," in *Proceedings of the 2002 American Control Conference (ACC)*, vol. 5, 2002, pp. 3741–3746.

[10] M. Watterson and V. Kumar, "Safe receding horizon control for aggressive mav flight with limited range sensing," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[11] B. Charrow, S. Liu, V. Kumar, and N. Michael, "Information-theoretic mapping using cauchy-schwarz quadratic mutual information," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2015.

[12] J. Israelsen, M. Beall, D. Bareiss, D. Stuart, E. Keeney, and J. van den Berg, "Automatic collision avoidance for manually tele-operated unmanned aerial vehicles," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6638–6643.

[13] S. Karaman and E. Frazzoli, "High-speed flight in an ergodic forest," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 2899–2906. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6225235

[14] B. Yamauchi, "Frontier-based exploration using multiple robots," in *Proceedings of the second international conference on Autonomous agents*. ACM, 1998, pp. 47–53.

[15] M. E. Flores, "Real-time trajectory generation for constrained nonlinear dynamical systems using non-uniform rational B-spline basis functions," Ph.D. dissertation, California Institute of Technology, 2007. [Online]. Available: http://core.ac.uk/download/pdf/11809687.pdf

[16] T. Lee, M. Leoky, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *Decision and Control (CDC), 2010 49th IEEE Conference on*. IEEE, 2010, pp. 5420–5425.

[17] "Qualisys motion capture system." [Online]. Available: http://www.qualisys.com/