2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Congress Center Hamburg
Sept 28 - Oct 2, 2015. Hamburg, Germany

Safe Receding Horizon Control for Aggressive MAV Flight with Limited Range Sensing

Michael Watterson[1] and Vijay Kumar[1]

***Abstract*— Micro Aerial Vehicles (MAVs) are becoming ubiquitous, but most experiments and demonstrations have been limited to slow flight except in open environments or in laboratories with motion capture systems. In this paper, we develop representations and algorithms for aggressive flight in cluttered environments. We incorporate the specific dynamics of the system and generate safe, feasible trajectories for fast navigation in real time. Specifically, we use a polyhedral decomposition of the visible free space and address the generation of safe trajectories that are within the space. Because of the limited field of view, we adopt a *receding horizon control policy* (RHCP) for planning over a finite time horizon, but with the guarantee that there exists a safe *stopping control policy* over a second time horizon from the planned state at the end of the horizon. Thus, the robot planning occurs over two horizons. While the robot executes the planned trajectory over the first time horizon, the map of obstacles is refreshed allowing the planner to generate a refined plan, once again over two time horizons. The key algorithmic contribution of the paper lies in the fast planning algorithm that is able to incorporate robot dynamics while guaranteeing safety. The algorithm is also optimal in the sense that the receding horizon control policy is based on minimizing the trajectory snap. Central to the algorithm is a novel polyhedral representation that allows us to abstract the trajectory planning problem as a problem of finding a path through a sequence of convex regions in configuration space.**

I. INTRODUCTION

For MAV applications in complex environments, planning relies on limited sensing and computation. A robot which flies quickly in cluttered environments requires quick planning. This work focuses on the problem of planning quickly under the constraints of the dynamics of the platform and the limited sensing horizon of a simulated onboard sensor. In this work, we assume the sensing, detection and localization of obstacles and the robot are solved.

Several works have taken inspiration for collision avoidance from a bird flying through a forest. [9] models a bird as a dynamical system moving in $\mathbb{R}^2$ at constant velocity in one dimension and direct control of velocity in the other dimension. Fixed wing aircraft, which are more dynamically constrained than multi-rotor MAVs, use motion primitives to plan trajectories through a forest [15]. Also using fixed wings, safety with receding horizon control has been explored with a planner which might not reach the goal [18].

General obstacle avoidance for fast moving mobile robots is addressed in [2]. If a MAV is equipped with 3D sensors, it can navigate around obstacles while moving at moderate speeds [14]. Some work uses a 2D Delaunay Triangulation to represent the free space as applied to mobile robots [3].

Trajectory generation for MAVs respects dynamics by minimizing snap of a $\mathcal{C}^4$ spline [13]. Mixed Integer Programs (MIP) can account for obstacle avoidance [6][13]. Generally, as the number of obstacles in the environment grows, the number of required integer variables grows as O(n), where n is the number of obstacles [13] or convex regions [6]. This growth in integer variables makes modest sized problems too slow for real time re-planning. Another approach is to use an RRT to seed an optimization routine [17].

[1] GRASP Lab, University of Pennsylvania, Philadelphia, PA, USA

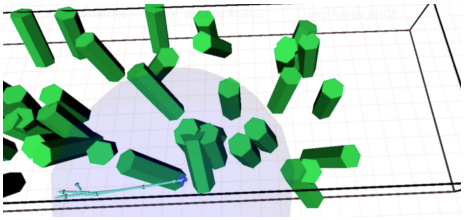


Fig. 1: Our goal is to generate safe trajectories (cyan) in a bounded environment with obstacles (green), using information in limited sensing radius (blue sphere). We maintain two receding horizon control policies, to plan a quickly as possible, but without sacrificing completeness by retaining memory of what the robot has observed.

In this paper, we first develop a novel, efficient representation the 3D environment using the Delaunay tetrahedralization [5]. Using the assumption of limited range information, we propose a Short Range Receding Horizon Control Policy (SRRHCP) to safely and quickly navigate the environment with real-time re-planning ability. We see that ensuring real time operation limits the completeness of this planner. Therefore it is necessary to use a Long Range Receding Horizon Control Policy (LRRHCP) to generate trajectories though an arbitrarily sized non-convex configuration space. This LRRHCP preforms well on hardware as a stand alone planner compared to [6] [13] in cases where the entire environment is known. Combining these policies, produces a planner which is fast, safe, and complete.

II. PROBLEM FORMULATION

We examine the problem of controlling a multi-rotor MAV modeled with some key assumptions. Firstly, we model the robot's collision geometry as a sphere. This is a conservative approximation and only prevents the robot from flying through tight spaces like windows. For the planning problem, the robot is assumed to be able to localize itself.

For the receding horizon control policy (RHCP), we use a simplified perception model which is solely dictated by a fixed sensing radius. The robot knows the exact position and geometry of an obstacle once the distance between it and the robot is less than the sensing radius. All obstacles in the environment are static, so their location will be the same for all future observations.

An MAV is modeled as a single rigid body, therefore its

position is described as an element of the special euclidean group $SE(3)$. Any derivatives of its position and orientation are isometric to $\mathbb{R}^n$. The control inputs to the robot can be decomposed into a force in the body z axis and a moment in the body frame m_b. Using the technique of [13], we can map the control inputs and the state of the robot to a differentially flat representation in $\mathbb{R}^{14} \times SO(2)$. Since this mapping exists and is smoothly invertible, we represent the state of the robot with the tuple $(\mathrm{X}, \Theta) \in \mathbb{R}^{14} \times SO(2)$:

$$\mathrm{X} = \begin{bmatrix} x & \dot{x} & \ddot{x} & \dddot{x} \end{bmatrix} \qquad \Theta = \begin{bmatrix} \psi & \dot{\psi} & \ddot{\psi} \end{bmatrix} \tag{1}$$

The position, velocity, acceleration, and jerk of the center of mass of the robot are represented as $x, \dot{x}, \ddot{x}, \dddot{x} \in \mathbb{R}^3$ respectively. The yaw of the robot and time derivatives are represented as $\psi \in SO(2)$, $\dot{\psi}, \ddot{\psi} \in \mathbb{R}$. The actuator limitations of the robot are modeled as confining the magnitudes of the acceleration and jerk. Due to the symmetry of the spherical model, we only need to consider the X part of the state when planning for obstacle avoidance. From here onwards Θ is assumed to be controlled to always be 0.

The set of all positions of the robot is the configuration space. In our system, this is the space of the zeroth order flat variables $\mathrm{C} = \mathbb{R}^3$. The free space $C^{free} \subset \mathrm{C}$ is the set of all positions where the robot is not in collision with obstacles. Using the spherical collision assumption, this space is the set of all points which are at least the radius of the robot r_q away from all obstacles $\mathcal{O}_i$ in an environment $\mathcal{N}$.

$$C^{free} = \{p \in \mathrm{C} | \; ||p - q|| > r_q \; \forall q \in \mathcal{O}_i, \; \forall \mathcal{O}_i \in \mathcal{N}\} \tag{2}$$

Choosing the representation of obstacles in the environment affects which methods a direct planner can use. There are two typical representations of obstacles, occupancy grids, and convex polyhedra. Occupancy grids discretize the world into many cubes with a binary free or occupied state. More sophisticated approaches use adaptive sized cubes. Alternatively, convex polyhedra represent obstacles as the intersection of half planes with the robot position restricted by these half planes [13] [6].

Occupancy grids create a large adjacency graph of the environment, which is costly to search through. Adaptive sized grids, shrink the size of this graph in some cases, but not in Figure 7. Searching directly on an environment graph with dynamics would require searching through a space of $\mathbb{R}^{12}$ ($\mathbb{R}^3 \otimes \mathbb{R}^4$) for our system, which is also too costly.

Instead we represent the environment as set of convex tetrahedrons. Any trajectory through the environment must pass through a sequence of these tetrahedrons. When trying to find a trajectory through the environment, we first find a sequence of adjacent tetrahedrons in C^{free}. With such a sequence of convex regions, we can formulate the trajectory generation problem as a convex optimization problem.

We assume that the environment is given as a set of non-intersecting convex obstacles $\mathcal{N} = \{\mathcal{O}_i\}$. Any non-convex or overlapping obstacles can be decomposed [12]. Each convex obstacle is approximated as a closed triangular mesh which is a set of vertices, edges, and faces $\mathcal{O}_i := \{\mathcal{V}, \mathcal{E}, \mathcal{F}\}$.

Let $P := \{\mathcal{V}_j \in \mathcal{O}_i, \mathcal{O}_i \in \mathcal{N}\}$ be the set of all vertices in the obstacles. The Delaunay triangulation of the points P produces a set of tetrahedrons $\mathcal{T}$ If enough points are added to obstacles [4], any tetrahedron in the triangulation will be either entirely in C^{free} or entirely in N.

In addition to the benefits on the planning side, this method uses an efficient amount of storage because it stores a number of region which scales linearly with the number of obstacles. We choose the Delaunay triangulation [5], which is defined by the property that the circumsphere of any tetrahedron does not contain the verticess of any other tetrahedron in the decomposition. The method is well suited to online systems because the decomposition can be incrementally built up [5].

The goal of the RHCP is to find a time parameterized trajectory $\tau(t) : [t_0, t_f] \to \mathbb{R}^{12}$ such that τ maps from a start state X_s to an end state X_g (3), the component x is always in the free space, the higher derivatives are bounded, and the τ is continuous and the derivatives have the appropriate relations (4). For this to be well posed, we also restrict τ to be in the set of functions whose snap is continuous $\mathcal{C}^4$.

$$\tau_0 = \mathrm{X}_s \quad \tau_{t_F} = \mathrm{X}_g \quad \tau(t) \equiv \begin{bmatrix} x_t & \dot{x}_t & \ddot{x}_t & \dddot{x}_t \end{bmatrix} \tag{3}$$

$$x(t) \in C^{free} \quad x(t) \in \mathcal{C}^4 \quad \tfrac{d^k x}{dt^k} \leq \max_k \quad k = 1..3 \tag{4}$$

III. TRAJECTORY OPTIMIZATION

Any trajectory $\tau : [t_0, t_f] \to \mathbb{R}^3$ through the world will intersect a set of the tetrahedrons in the triangulation. The usual trajectory generation problem with obstacles is non-convex because of the holes the obstacles create in the C^{free}. However if the trajectory segments are confined to be within a known corridor, finding a trajectory can be formulated as a convex problem. We find this set of tetrahedrons before trying to find a dynamically constrained trajectory.

In the fixed rotor MAV literature, it is common to minimize the integral of a time derivative of the robot's position [13] [6], or a weighted sum of time derivatives [17]. This choice of cost functional allows us to use a Quadratic Program (QP) solver, unlike other choices of functional such as minimizing power or time.

The optimization can be reformulated as a quadratic program (5) using a Legendre spline basis with coefficients α to represent $x(t)$ [13]. We use one spline segment for each tetrahedral region in the sequence returned by Algorithm 2. We can constrain a segment of the trajectory to be inside a convex region by inequality constraints on the basis coefficients using the method of [13].

$$\begin{array}{rl} \min & \int ||x^{(4)}(t)||^2 dt \\ s.t & (4) \end{array} \quad \Leftrightarrow \quad \begin{array}{rl} \min & \alpha^\intercal D \alpha \\ s.t & A\alpha = b \\ & C\alpha \leq d \end{array} \tag{5}$$

D is a diagonal matrix, A and b are constructed to match continuity of spline segments up to three derivatives, and C,d confine the spline segments to the tetrahedrons and limits on velocity, acceleration, and jerk. After we solve for the vector of coefficients α, we can explicitly recover $\tau(t)$.

Paramount to quickly generating a trajectory is to find times for the segments in (5). Using gradient descent [13]

[17] requires many iterations of the quadratic program. Instead, we find approximate values for these times by using a second optimization routine. For this we approximate the shortest path, made up of straight line segments, passing through the tetrahedrons. We represent this path by the end points of each line p_i. Each p_i must lie on the faces of the tetrahedrons in the sequence $\mathcal{F}_i$.

The sum of the length of each line segment $||p_i - p_{i-1}||_2$ cannot be directly formulated as a quadratic program because the square root is a concave function. Instead we used the 1-norm, which can be reformulated as a linear program.

$$\begin{array}{ll} \min & \sum ||p_i - p_{i-1}||_1 \\ s.t. & p_i \in \mathcal{F}_i \;\; i = 0..n_s \end{array} \tag{6}$$

The line segments each have a length $||p_i - p_{i-1}||_2$. To find times for each segment, we re-weight a trapezoidal acceleration profile across the total length to individual trajectory segments. In the event that this time allocation creates a trajectory which violates constraints, we can generate the times with a more conservative acceleration profile.

IV. RECEDING HORIZON CONTROL POLICY (RHCP)

The receding horizon control policy operates over a short range and a long range. They both make progress towards the goal based on new observations of obstacles. At the same time, we ensure safety by maintaining that the robot can come to a complete stop at all times.

A. Short Range Receding Horizon Control Policy (SRRHCP)

Like [6][8][16][17], the SRRHCP relies on a sampling based strategy to speed up computation. It plans from the current state to a set of waypoints inside the current sensing radius. We simultaneously generate plans to each of these waypoints which take Δt seconds and stopping trajectories starting at the waypoints and ending, with zero velocity, at some point inside the sensing radius after T_s seconds. After checking the feasibility of these primitive trajectories, we execute the one that brings us closest to the goal X_g with the greatest velocity towards the goal.

We define the SRRHCP at planing step k to generate trajectories from X_k to candidates for state X_{k+1}. Each of these candidate trajectories Φ is referred to as a motion primitive. They are maps from two states and a transition time Δt to a function from an input range to state space:

$$\begin{array}{l} \Phi(\mathrm{X}_k, \mathrm{X}_{k+1}, \Delta t) : \mathbb{R}^{12} \times \mathbb{R}^{12} \to \mathcal{C}^4(\mathbb{R}) \\ \Phi(\mathrm{X}_k, \mathrm{X}_{k+1}, \Delta t)(0) = \mathrm{X}_k \\ \Phi(\mathrm{X}_k, \mathrm{X}_{k+1}, \Delta t)(\Delta t) = \mathrm{X}_{k+1} \end{array} \tag{7}$$

A stopping policy Ψ is one which slows down as quickly as possible in a straight line from X_k with time T_s.

$$\begin{array}{c} \Psi(\mathrm{X}_k, T_s) : \mathbb{R}^{12} \to \mathcal{C}^4(\mathbb{R}) \\ \Psi(\mathrm{X}_k, T_s)(T_s) = \begin{bmatrix} x_f & 0 & 0 & 0 \end{bmatrix} \end{array} \tag{8}$$

For brevity will now suppress parameters of the primitives:

$$\Phi_k \equiv \Phi(\mathrm{X}_k, \mathrm{X}_{k+1}, \Delta t) \tag{9}$$

$$\Psi_k \equiv \Psi(\mathrm{X}_k, T_s) \tag{10}$$

The set of velocities we plan to at the waypoints is sampled from a set of velocities $V_{k+1}\{v_1, v_2, ..., v_\beta\}$ such that $||v_i|| < ||v_{max}||$ and $v_i/||v_i||$ is sampled from some subset of the 2 dimensional sphere $\mathcal{S}^2$. We choose $||v_i||$ to be uniform across 0 to v_{max} and $v_i/||v_i||$ to be parametrized by a distribution over a heading angle and an inclination angle.

To find the waypoints, we sample adjacent tetrahedrons to find candidate points which are collision free. From tetrahedron which the robot is in, we find other tetrahedrons which are connected through at most m_δ faces. The centroids of these tetrahedrons are selected as sample points because they form a discretization which scales with the complexity of the free space. We call this set of centroids K_{k+1}.

The candidate end points X_{k+1} are chosen from the cross product of the velocity and centroid samples $\mathrm{S}_{k+1} := \{\begin{bmatrix} x & v & 0 & 0 \end{bmatrix} | x \in K_{k+1}, v \in V_{k+1}\}$. From the robot location to each end point, we generate a primitive Φ_k. We avoid using a QP solver by algebraically solving (5)(7) without inequality constraints. We keep Φ_k only if it stays within the ignored inequality constraints.

$$\mathrm{S}'_{k+1} = \{\mathrm{X}_{k+1} \in \mathrm{S}_{k+1} | \Phi_k \in \mathcal{C}^{free}\} \tag{11}$$

With each primitive in S'_{k+1}, we generate Ψ_{k+1} with T_s such that the robot stops as quickly as possible while respecting (4). Using the same trick as Φ_k, we solve for Ψ_{k+1} in closed form. From only the valid stopping policies we then have a set of plans:

$$\mathrm{P}_{k+1} := \{(\Phi_k, \Psi_{k+1}) | \Phi_k \in \mathrm{S}'_{k+1}, \Psi_{k+1} \in C^{free}\} \tag{12}$$

Algorithm 1 Short Range Receding Horizon Control Policy

From state X_k
$\mathrm{S'}_{k+1} \leftarrow \emptyset$
$\mathrm{P}_{k+1} \leftarrow \emptyset$
for $\mathrm{X}_{k+1} \in \mathrm{S}_{k+1}$ **do**
  if Φ_k is Valid **then**
    Add X_{k+1} to $\mathrm{S'}_{k+1}$
for $\mathrm{X}_{k+1} \in \mathrm{S'}_{k+1}$ **do**
  if Ψ_{k+1} is Valid **then**
    Add (Φ_k, Ψ_{k+1}) to P_{k+1}
if P_{k+1} is empty **then**
  return Failure
else
  return Success

B. Short Range Limitation

We also seek to address the completeness problems with only planning within a local horizon. Figure 3 shows an environment were a myopic planner will fail as a path to the goal requires backtracking beyond the range of the sensor.

A planner which uses the whole observed map does not have this short sight. However, the completeness comes at the cost of computation efficiency. As the observed map grows, so does the time required to synthesis a plan. Since a short

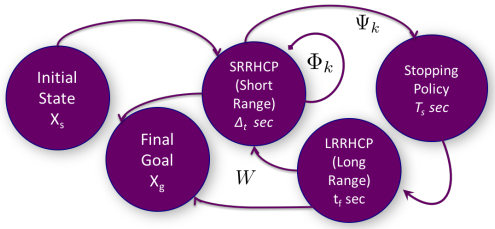


Fig. 2: Receding Horizon Control Policy

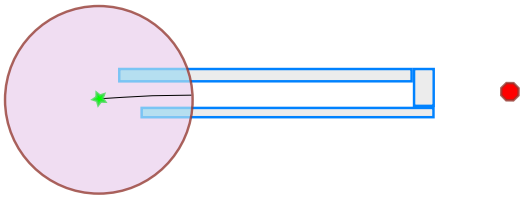


Fig. 3: A plan from start (green star) to goal (red octagon) with information only inside burgundy circle will get stuck in the corridor (blue) if it continues to plan with local information

sighted planner works most of the time, we use an approach which switches between a LRRHCP and a SRRHCP to plan quickly without sacrificing completeness.

C. Long Range Receding Horizon Control Policy (LRRHCP)

The LRRHCP plans inside the entirety of free space which has been observed by the robot. It takes in two points, and plans a trajectory between them using the planning method in Section III. To ensure safety within our computational budget, we require that the start and end points of the LRRHCP have zero velocity, acceleration, and jerk.

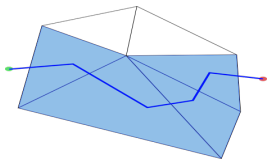


Fig. 4: 2D projection of finding convex path through environment. LRRHCP selects a path of tetrahedrons which lead from green start to red goal to confine the trajectory.

Algorithm 2 Get Tetrahedron Path

Given x_s, $x_g \in \mathbb{R}^3$
Initialize empty graph $G(V, E)$
for all $T_i \in T$ **do**
    if T_i is not in collision **then** $T_i \to V$
for all $\mathcal{F}_j \in \mathcal{F}$ **do**
    $[T_1, T_2] \leftarrow$ find tetrahedrons adjacent to $\mathcal{F}_j$
    if $T_1 \in V$ And $T_2 \in V$ **then**
        $\mathcal{F}_j \to E$
Find Shortest Path from $x_s \to x_g$ in G

All possible sets of paths through free space can be represented as a graph with the nodes being the tetrahedrons and the edges being their faces. To get a nominal edge weight, we use the distance between the centroids of the tetrahedrons. When a desired goal and initial position are given to the planner, the corresponding cells in the triangulation are both found. Now a path between these nodes can be found using Dijkstra's search [11]. This results in a sequence of tetrahedrons $\{T_j\}$ for some $j = 1..m$ which connect a start and goal in $\mathbb{R}^3$. From here we use a QP solver to solve (5).

D. The complete planning algorithm

The global planning method alternates between the SRRHCP and the LRRHCP using algorithm 3. The SRRHCP is not complete, as it can get stuck in local minimum. The LRRHCP is complete, but cannot handle re-planning from a from a way point with non zero velocity. However alternating between the two can be used to fly through an unknown environment without getting stuck.

The robot can explore its environment locally using the SRRHC, but may not be able to reach its goal due to local minimums (Figure 6a). As it moves, it updates an explored region $R \subset \mathbb{R}^3$ which is the union of all sensed space. This can be efficiently represented with the Delaunay tetrahedrons. When the robot can no longer progress with the SRRHC, it executes its stopping policy.

Whenever the SRRHC stops, either the goal is inside the reachable subset of the explored region $R \subset E$, or the goal is outside this region. In the first case, the LRRHC can find trajectory it if one exists. In the second case, we look at the intersection of the boundary of the reachable set and the boundary of the explored region $\partial R \cap \partial E$. If this set is empty then the reachable set is strictly inside the explored region, thus there is no way to expand the reachable region. In this situation no path can exist because the robot has started in a compact region which does not contain the goal. If $\partial R \cap \partial E$ is not empty, the LRRHC finds a path to a point on the boundary of the reachable set and explored set. Traveling to this point will increase the explored region R, and then the planner switches back to the SRRHCP. Assuming that the environment is finite, this procedure will always terminate with success, or failure because the explore region is limited to the size in which it can grow.

V. VALIDATION

For the simulation results, we used a desktop workstation with 24 GB of RAM and a quad core 3.4 GHz processor. All software is written in C++ using the Robotics Operating System (ROS) to handle all of the sensor communication. The optimization is done by the Gurobi package [7] using its C++ interface. We use one thread each for the dynamics simulation, control, SRRHCP, LRRHCP, and full RHCP.

The obstacles in simulation are cylinders with a fixed radius and height modeled by regular polygon prisms and inflated by the radius of the robot to avoid collision. We built environments in two different ways using rectangular meshes: firstly criss-crossing pole like obstacles as in Figure 5 and the other with similar obstacles distributed randomly in both position and orientation (Figure 1). The robot was commanded to fly to a point 20 m away using the full RHCP, only the SRRHCP, and only LRRHCP planners. The timing results of the Delaunay decomposition and graph population

in algorithm 2 of the entire environment is shown in Table I. This operation only needs to be done once for large horizon planning, and can be done incrementally during exploration. This table shows the representation scales well to handle large maps of many obstacles.

Algorithm 3 Full Receding Horizon Planner

Given X_s and X_g
Explored Region $E \leftarrow$ Current Sensing
$\mathrm{X} \leftarrow \mathrm{X}_s$
while $\mathrm{X} \neq \mathrm{X}_g$ **do**
  while Short Range RHCP is Sucessful **do**
    $(\Phi_k, \Psi_k) \leftarrow \arg \min_{(\Phi_k,\Psi_k)\in P_k} ||\Phi_k(\Delta_t) - \mathrm{X}_g||_2$
    Execute Φ_k
    $\mathrm{X} \leftarrow \Phi_k(\Delta_t)$
    $E \leftarrow E \cup$ Current Sensing
  Execute Ψ_k
  $\mathrm{X} \leftarrow \Psi_k(T_s)$
  Find $R \subset E$ such that R is reachable from X
  if $\mathrm{X}_g \in R$ **then**
    $W \leftarrow$ Long Range RHCP from X to X_g
  else
    $Y \leftarrow \arg \min_{y\in\partial R\cap\partial E} ||y - \mathrm{X}||_2$
    $W \leftarrow$ Long Range RHCP from X to Y
    if $W = \emptyset$ **then**
      return No Path Exists
  Execute W
return Success

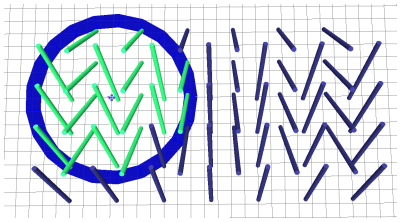


Fig. 5: Criss-crossed Environment

The planning and execution time for the trajectories in a prototypical criss-crossed environment with size of 20 m by 10 m were tested with using the SRRHCP, the LRRHCP replaning with a 5m sensing radius, and the LRRHCP with the infinite sensing radius. In this example, the SRRHCP never got stuck, so the full RHCP reduced to using just the SRRHCP. The results for this are tabulated in table II. The LRRHCP with the infinite sensing radius generated a plan through 30 convex cells. The full RHCP was run on the random environment (Figure 1), which has the same dimensions as the criss-crossed environment.

We also validated the LRRHCP using an AscTec Pelican MAV [1], see the video for the robot planning and executing a sequence of 3D trajectories in the criss-cross environment.

VI. COMPUTATIONAL COMPLEXITY

The 3D Delaunay triangulation has worst case $O(p)$ computational complexity with respect to the number of points p when all the points are not coplanar [5]. Since our obstacles have the same number of points, the triangulation complexity is $O(b)$, with respect to the number of obstacles b.

The number of tetrahedrons is worst case linear with respect to the number of points and obstacles. Dijkstra's search is worst case $O(|E|\log|V|+|V|\log|V|)$. In our case, the number of vertices and edges are proportional, so the search complexity is $O(b \log b)$

Type	Num Obs	Num Verts	$t_{Decom}(s)$	$t_{Graph}(s)$
Criss-cross	6	120	0.094	0.023
Criss-cross	816	16320	11.40	4.64
Criss-cross	88	1760	1.24	0.47
Criss-cross	24	600	0.83	0.16
Criss-cross	3264	26112	16.34	7.25
Random	1000	4000	2.83	2.96
Random	120	480	0.48	0.28
Random	30	480	0.26	0.13
Random	10	120	0.072	0.024
Random	1000	12000	6.26	3.81

TABLE I: Computation Time Representation Generation

Type	Radius	SRRHCP Iter	LRRHCP Iter	Total Time
SRRHCP	5 m	15	0	24.08
LRRHCP	5 m	0	6	26.14
LRRHCP	∞	0	1	11.29 s
Full RHCP	5 m	11	2	41.84 s

TABLE II: Total time in criss-cross environment (SRRHCP, LRRHCP) and random environment (Full RHCP)

The complexity of the quadratic program depends on the number of segments l in the path returned by Dijkstra. Worst case, it is always possible to construct a "maze-like" environment of which the only feasible path is through all the tetrahedrons to which the path length is $O(b)$. For each path segment there will be $O(1)$ variables and $O(1)$ constraints on the program. Therefore there will be $O(b)$ variables and $O(b)$ constraints total. From [10] the QP will take $O(b^{3.5})$ arithmetic operations.

Other methods which require a MIP solver [6] [13] [18], are at best, operating with $O(2^C)$ complexity with respect to the number of constraints C on the problem. C grows linearly with respect to the complexity of the environment.

It is hard to compare the worst case complexity of our algorithm to those [17] which rely on probabilistically complete planning methods such as RRT* [8]. It is possible for us to construct examples with very narrow configuration space which have constant worst case complexity with respect to a configuration space volume $\epsilon > 0$. Despite having constant complexity for our planner, as $\epsilon \to 0$, the RRT* planner needs $N \to \infty$ samples to find an optimal plan.

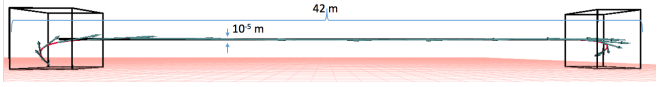
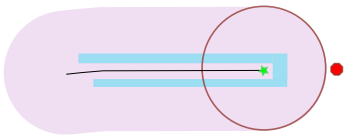
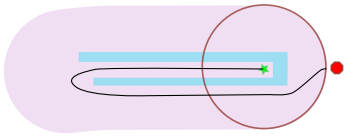


Fig. 7: Narrow configuration space where our planner vastly outperforms sampling based methods. Volume of the central sliver is $2.8 \cdot 10^{-9}\%$ of C^{free}. Our LRRHCP takes 55ms to do the decomposition and calculate this trajectory.

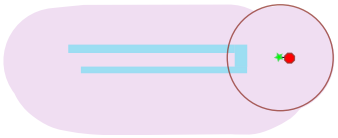
For the SRRHCP, the computation complexity scales with



(6a) SRRHCP has failed to find a new trajectory and has executed stopping policy Ψ_k. During execution, the explored region E has grown to lavender region.



(6b) All of the explored area is reachable $R = E$. The LRRHCP plans a trajectory to the closed point on the boundary of the reachable region ∂R to the goal.



(6c) After LRRHCP trajectory is executed, the SRRHCP regains control and is able to go directly to the goal since it is within the sensing radius.

the number of sampled parameters. The number of sampled centroids $|\mathrm{K}_{k+1}|$ is exponential in local depth searched $O(2^{m_\delta})$. Since the search space is the Cartesian product of K_{k+1} with $\mathrm{V}_{\mathrm{k+1}}$, the total complexity is proportional to $O(2^{m_\delta} \cdot |V_{k+1}|)$. For operation, we fix these parameters, so the SHRRCP has constant complexity with respect to the environment complexity.

VII. CLOSING REMARKS

We have presented a planning and trajectory generation method to enable fast navigation of a system with high order dynamics through an 3D environment. Using a geometric decomposition of the environment, we are able to quickly form and solve a quadratic program to generate smooth trajectories which avoid obstacles.

Using a short range receding horizon control policy, we are able to generate safe stopping policies concurrently with minimum snap trajectories. Using this myopic approach cannot completely navigate the environment, but we combine it with our Delaunay triangulation based trajectory generator to generate a full receding horizon control policy.

In contrast to the comparable obstacle avoidance methods using mixed integer programming [6] [13], our LRRHCP can generate and execute a plan in an environment (Figure 5) with 44 obstacles in much less time than the other methods take in problems with fewer obstacle regions. Our RHCP works with a limited sensing range unlike the similar RRT-QP method [17] that requires a full map of the environment.

We have tested these policies in simulated environments with cylindrical obstacles distributed in multiple ways. The planned LRRHC trajectories were tested on actual hardware to demonstrate that they are dynamically feasible on a real system. Our results demonstrate a system that navigate 3D obstacle filled environments with up to average planning and execution speeds of about 2 m/s and peak velocities of 4 m/s.

There are several ways to extend this work to more general applications. We can think about using probabilistic models of the environment, but then safety and completeness guarantees need to be relaxed into probabilistic completeness and a probabilistic measure of safety. We can model non-symmetric sensors by adding dimensions to our search space for the orientation of the robot. For planning in more complicated manifolds, like SE(3), these techniques can be extended by using higher dimensional Delaunay decompositions with more general metrics.

VIII. 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. We thank Dr. Philip Dames for his editorial comments and others in the MRSL lab for their help with the software and hardware infrastructure.

REFERENCES

[1] Ascending Technologies, GmbH.

[2] Johann Borenstein and Yoram Koren. Real-time obstacle avoidance for fast mobile robots in cluttered environments. In *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pages 572–577. IEEE, 1990.

[3] Michel Buffa, Olivier D. Faugeras, and Zhengyou Zhang. Obstacle Avoidance and Trajectory Planning for an Indoors Mobile Robot Using Stereo Vision and Delaunay Triangulation.

[4] Paulo Roma Cavalcanti and Ulisses T. Mello. Three-Dimensional Constrained Delaunay Triangulation: a Minimalist Approach. In *IMR*, pages 119–129, 1999.

[5] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. Delaunay triangulations. *Computational Geometry: Algorithms and Applications*, pages 191–218, 2008.

[6] Robin Deits and Russ Tedrake. Efficient Mixed-Integer Planning for UAVs in Cluttered Environments. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015.

[7] Inc. Gurobi Optimization. *Gurobi Optimizer Reference Manual*. 2015.

[8] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[9] Sertac Karaman and Emilio Frazzoli. High-speed flight in an ergodic forest. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2899–2906. IEEE, 2012.

[10] Mikhail K. Kozlov, Sergei P. Tarasov, and Leonid G. Khachiyan. The polynomial solvability of convex quadratic programming. *USSR Computational Mathematics and Mathematical Physics*, 20(5):223–228, 1980.

[11] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.

[12] Khaled Mamou and Faouzi Ghorbel. A simple and efficient approach for 3d mesh approximate convex decomposition. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 3501–3504. IEEE, 2009.

[13] Daniel Mellinger. *Trajectory generation and control for quadrotors*. 2012. Copyright - Copyright ProQuest, UMI Dissertations Publishing.

[14] Matthias Nieuwenhuisen, David Droeschel, Marius Beul, and Sven Behnke. Obstacle Detection and Navigation Planning for Autonomous Micro Aerial Vehicles. In *Vision-based Vehicle Guidance*, pages 268–283. Springer, New York, 1992.

[15] A. A. Paranjape, K. C. Meier, X. Shi, S.-J. Chung, and S. Hutchinson. Motion primitives and 3d path planning for fast flight through a forest. *The International Journal of Robotics Research*, February 2015.

[16] Mihail Pivtoraiko, Daniel Mellinger, and Vijay Kumar. Incremental micro-UAV motion replanning for exploring unknown environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2452–2458. IEEE, 2013.

[17] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2013.

[18] Tom Schouwenaars, Jonathan How, and Eric Feron. Receding horizon path planning with implicit safety guarantees. In *American Control Conference, 2004. Proceedings of the 2004*, volume 6, pages 5576–5581. IEEE, 2004.