Hierarchical Trajectory Planning for Quadrotor Flight in Unknown Environments

Furqan Ahmed, Matt Brymer, Sangita Sahu and Vandan Rao

***Abstract*— Trajectory planning for high speed quadrotor flight remains an open problem due to the requirements of fast processing time and accurate environment representations that are challenged by the extreme vehicle agility and limited compute capability. We investigated modifying an existing hierarchical trajectory planner to address the issue of time allocation with an alternate problem formulation and implemented each of its components in a set of ROS nodes. We found the proposed SQP formulation to be less robust to challenging environments than the original MIQP in the reference planner and thus fell back on the MIQP approach. Due to challenges in integration it was necessary to test our components separately. Our global planner implementation successfully navigated to goal locations in simulated forests. We found our convex decomposition and local planning components were able to support flight of up to 4.2 m/s at replanning rates of approximately 10 Hz in similar environments. We identified a number of challenges and areas for future work including simplifying the results of the convex decomposition, incorporating observability into the planning objective and a more robust time allocation methodology.**

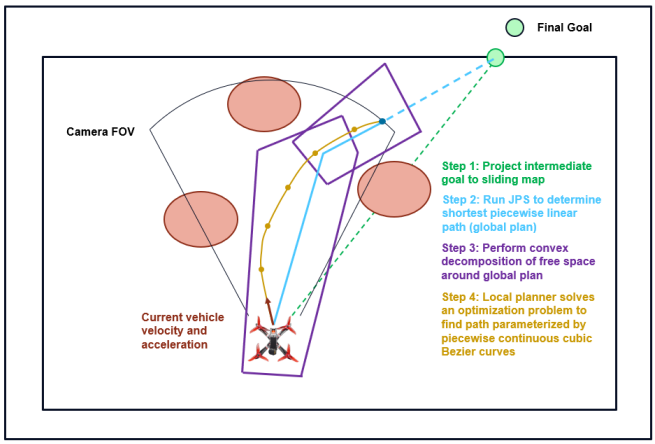

Fig. 1. Overall strategy of the hierarchical planning algorithm implemented for this project

I. Introduction

Autonomous UAVs, including quadrotors, have become commonplace in applications such as aerial photography, outdoor surveillance of industrial assets and package delivery. They are increasingly being tasked with operation in dense environments such as forests, urban or indoor spaces where the world map may not be known apriori. The combination of a larger number of obstacles and only partial observability of the world in these applications makes the problem of motion planning more difficult and requires methods that are capable of rapidly generating trajectories based on new information that are safe, smooth and dynamically feasible. Producing algorithms that can meet these needs while still being feasible for onboard computation is still an area of active research.

II. Literature Review

The quadrotor system dynamics possess the property of differential flatness, namely that the state and required control inputs can be directly computed from a set of flat output variables. For the quadrotor it can be demonstrated that the combination of the position and yaw angle satisfy this property, meaning that almost any given trajectory can be achieved given it lies within the control input limits[1]. This natural agility and flexibility has lead to a variety of approaches to the motion planning problem.

Some authors have explored sampling based methods, where at each replanning step an array of possible trajectories based on motion primitives are sampled and one is selected for execution based on some cost criterion. These motion primitives are in some cases based on optimal control theory results for minimum time or jerk motions[2][3]. These methods offer closed form solutions that enable very fast replanning times, but do not necessarily guarantee optimal behaviour over larger time horizons. Others have explored graph search based methods based on minimum time motion primitives, some even that fully plan in $SE(3)$ and perform collision checking based on the quadrotor's attitude and an ellipsoidal body[4][5]. While these approaches are powerful, they are often too expensive to be run onboard on computationally constrained platforms.

One promising class of methods that have emerged are hierarchical trajectory planners. These methods consist of a fast global planner that produces an initial path to the goal location based on some minimum cost heuristic that is then refined in a subsequent optimization step. Their inherent modularity has lead to a variety of different techniques being studied for each component in the literature. The global planning step is commonly performed with graph search techniques such as Jump Point Search (JPS)[6], A* [7] or sampling search methods such as RRT*[8]. Methods of environment representations for performing collision checking in both the global and local planning steps have included directly comparing against observed points in the world map via a Euclidean Distance Field[7] or a convex decomposition of the free known space[9][10]. To facilitate the optimization problem the trajectory is parameterized, often with a piecewise polynomial trajectory or Bezier curves, and an optimization problem is solved to minimize some form of motion derivative or trajectory time [6][8][9]. Constraints on the vehicle speed, acceleration and jerk to keep the trajectory dynamically feasible have been implemented by sampling

the trajectory[9] or by utilizing the convex hull property of b-splines[7]. Some techniques plan only in the known free space for safety[7], while others assume the unknown space is free to allow for faster flight speeds but also maintain a safe backup trajectory such as the FASTER algorithm[6]. Depending on the set of features used the optimization problem may end up being posed as a constrained QP[1], an unconstrained QP[8] or MIQP[6].

One challenging aspect of the quadrotor trajectory planning problem is that of coming up with an appropriate time allocation for the entire trajectory or individual segments. Too aggressive a time allocation can create an infeasible problem where the waypoints cannot be reached without exceeding vehicle dynamic limits. A too conservative time allocation however can force the quadrotor to take an unnatural circuitous route so that it reaches the goal location at the prescribed time. Some methods that have been explored in past works include performing a gradient descent optimization to adjust segment times set for an inner loop QP optimization[1], adaptively adjusting the time allocation for the overall trajectory between planning steps based on problem feasibility [6] or including the time in the objective function and removing it as a variable [8].

III. Methodology

Given the successes of hierarchical trajectory planners, we investigated the possibility of implementing a hybrid approach to leverage the strengths of various components in previous works. We adopt the approach of the FASTER algorithm of Tordesillas et al.[6] as a reference algorithm as it appears to strike a good balance between speed of execution, robustness of environment representation and overall flight speed. It consists of a fast global planner based on JPS, a polyhedral convex decomposition of the free space, a Bezier curve trajectory parameterization making use of the convex hull property and a backup trajectory to allow safely planning in the unknown space for higher flight speed. We studied the possibility of coupling its strengths with an alternate cost function formulation that puts time in the objective as studied by Richter et al. [8]. The overall aim is to leverage the use of a simple and fast yet robust environment representation and shift the focus of the optimization problem to time to generate faster trajectories. We also implemented the baseline MIQP formulation of FASTER as a basis for comparison.

Our overall algorithm architecture is illustrated in Figure 1 and is very similar to FASTER. We make the simplification of only planning in the free known space with a stopping condition at the end. This concession comes at the cost of the extra speed FASTER achieves by safely planning in both the known and unknown spaces. Despite the simplification this approach is still sufficient to compare the relative merit of the alternate optimization problem formulation. Our approach consists of four key components described below that we implemented and tested in a series of ROS nodes.

A. 3D Occupancy Map

The planning task for a quadrotor requires a 3D model of the environment to identify free and occupied spaces. In our implementation we use the open source OctoMap mapping library [11]. It produces a probabilistic 3D volumetric representation based on noisy sensor data.

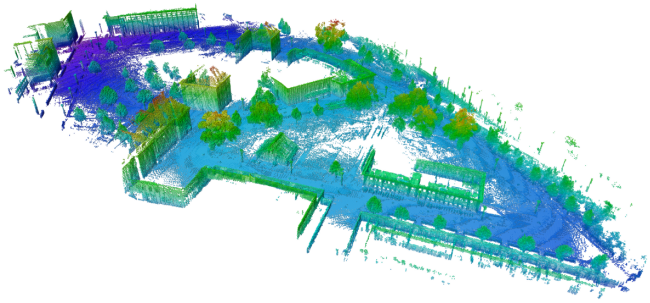

Fig. 2. Freiburg Campus Occupancy Map produced using OctoMap [11]

This mapping library uses an octree data structure for dividing the 3D space. Here each node of the tree represents the space contained in a cubic volume. Each volume is recursively subdivided into eight smaller sub volumes. This process is repeated till a given cut-off resolution is reached. Each leaf node stores the occupancy information of the space it represents. The occupancy information is modeled probabilistically to account for sensor noise and changing environments.

The probability of a leaf node being occupied is given by:

$$P(n|z_{1:t}) = \left(1 + \frac{1-P(n|z_t)}{P(n|z_t)} \cdot \frac{1-P(n|z_{1:t-1})}{P(n|z_{1:t-1})} \cdot \frac{1-P(n)}{P(n)}\right)^{-1}$$

Where,

$P(n|z_t)$ is the probability of a voxel being occupied given the sensor measurement z_t.

$P(n)$ is the prior probability.

$P(n|z_{1:t-1})$ is the previous estimate.

The occupancy grid output given by the OctoMap is shown in Figure 3 below.

For generating a global path using the Jump Point Search algorithm, a sliding map of fixed dimensions is required with the quadrotor at the centre. The sliding map includes free-known (F), occupied-known (O) and unknown space (U).

The occupancy information from the OctoMap, the quadrotor state and camera FOV information are used to generate a sliding map with F, O and U identified. Sample output from our mapper node with this information organized is shown in Figure 4. This information is published on the `/grid_publisher` topic for use by the JPS algorithm.

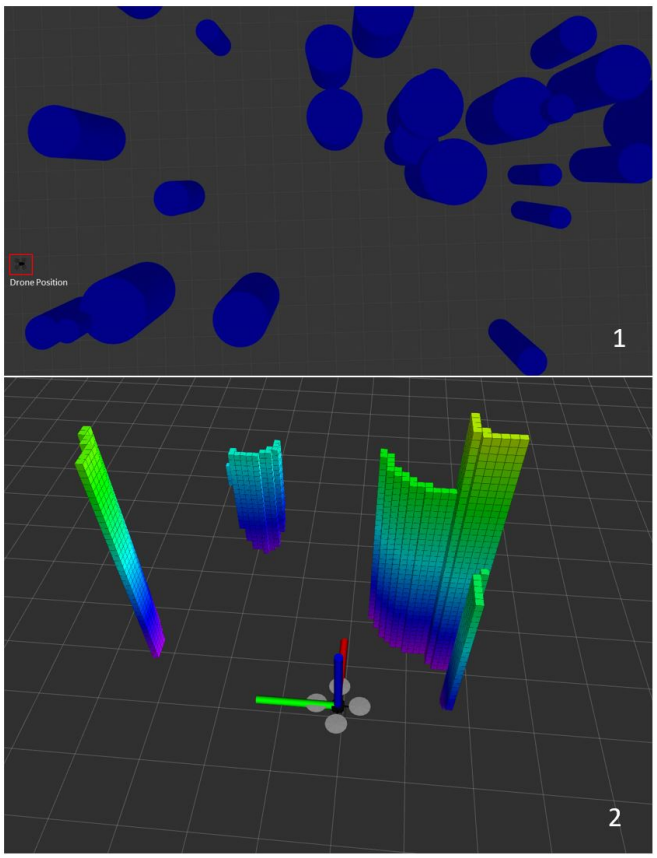

Fig. 3. Sample Occupancy Voxel Grid from OctoMap

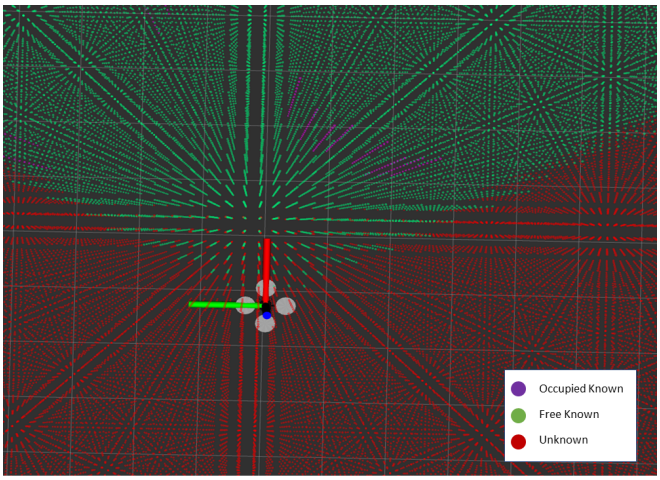

Fig. 4. Output from `/grid_publisher` topic

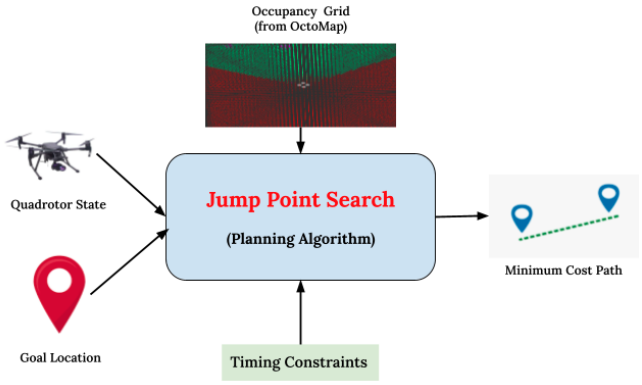

Fig. 5. Global planner setup

B. Global Planner

In the proposed framework, a global planner is implemented to provide shortest piece-wise linear paths from the start position to the goal using the Jump Point Search(JPS) algorithm. JPS models the world as a uniform cost grid with everything in the quadrotor's field of view treated as the known space and anything outside as the unknown space. Uniform grid maps allow pathfinding to be sped up by using symmetry reduction algorithms. Such algorithms eliminate many path symmetries and drastically reduce the number of paths to be searched during pathfinding [12].

The overall global planner setup is shown in Figure 5. The global planning starts with inflating the obstacles by the quadrotor's radius to avoid collision with the obstacles. The global goal is then projected onto the sliding 3D occupancy map. If the global goal is behind the quadrotor in the unknown space, the current position and the global goal form the global path, otherwise JPS is run to search the minimum cost path from the current location of the quad-rotor to the projected goal. The sliding map updates every planning step, and the JPS is run with the updated sliding map until the global goal is reached. The JPS input environment presented in Figure 6 shows the 3D Occupancy map, the inflated obstacles(white dots around colored obstacles), the global goal and the projected goal.

The JPS algorithm is an online symmetry breaking algorithm which speeds up pathfinding on uniform cost grid maps by "jumping over" many locations that would otherwise need to be explicitly considered [12]. It implements two simple pruning rules which are recursively applied during each search, one for the straight moves and the other for the diagonal moves as shown in upper half of Figure 7. In both the cases, we can immediately prune all the grey neighbours as these can be optimally reached from the parent of **x** without traversing through the node **x** at all. The set of neighbours shown in white, which are left out after applying the pruning steps are called as natural neighbours of the node **x**. These rules reduce the number of immediate neighbours around any node while expanding the nodes.

However, in the presence of obstacles around the node **x**, all the neighbours that can be optimally reached from the parent of the current node might not be eligible to be pruned. Thus, we can have forced neighbours as shown in lower half of Figure 7. For example, during a straight move if we have an occupied node right above or below the current node, we cannot prune the nodes ahead of the occupied nodes. Similarly, we cannot prune nodes 1 and 8 if nodes 4 and 7 are occupied respectively while moving diagonally. Thus

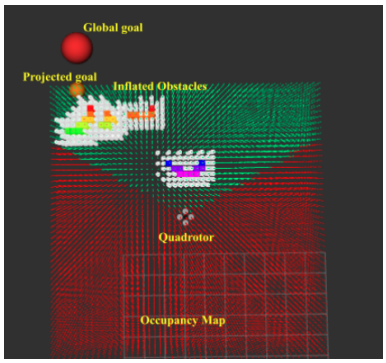

Fig. 6. JPS input environment

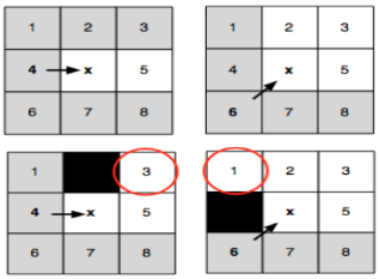

Fig. 7. JPS pruning rules (Left: Straight and Right: Diagonal) [12]

JPS searches for optimal paths by pruning the neighbours of the current node instead of looking for all the natural and forced neighbours, which eliminates all the symmetric paths from the parent of the current node to be expanded.

In addition to JPS we utilize a greedy Best-First Search(BFS) algorithm that uses the Euclidean distance of the current node to be expanded from the goal as a heuristic function. This means moves that take the quadrotor closer to the goal are favoured when deciding whether to take a straight or diagonal move next. JPS along with the BFS generates an optimal path with minimum path cost which is later used by the convex decomposition and local planning modules.

C. Convex Decomposition

Generating safe flight trajectories require defining regions where the quadrotor can plan a trajectory. These regions can be defined by bounded planes in an obstacle filled environment. We studied the ellipsoid convex decomposition methods for formation of safe flight corridors by Liu et.al [9] and implemented the same in our project.

We start by channeling global path info and the occupancy data from the previous modules. We then define a fixed bounding box around each segment of the JPS solution of Bx By Bz dimensions. The obstacles inside this box are obstacles of interest for that segment and are retained.

We now form obstacle free ellipsoids around the path segment. An ellipsoid is described as a deformation of a sphere given by:

$$\lambda(C,d) : p = (C \cdot p + d) | \, p <= 0$$

Where,

C is a 3x3 symmetric positive definite matrix representing the deformation of a sphere. This can further be decomposed intuitively into RAR^T. R is the 3D rotation from global frame to the ellipsoid axis and A is the semi axis lengths of the ellipsoid.

For a given segment defined by end points p_1 and p_2, we start by inflating the obstacles only in the direction of the segment by the radius of the quadrotor. Then we define a sphere around the segment centered at the midpoint of the segment. With the obstacles inflated, we consider the obstacles inside the sphere as obstacles of interest. Next we shrink the ellipsoid in the X-Y plane to the nearest inflated obstacle to the segment. The Z axis of the newly formed ellipsoid is reset to the radius of the sphere and re-shrunk to the nearest obstacle, if any in the Z-axis. These steps are continued recursively till an obstacle free ellipsoid is formed.The process is shown in detail in figure 8.

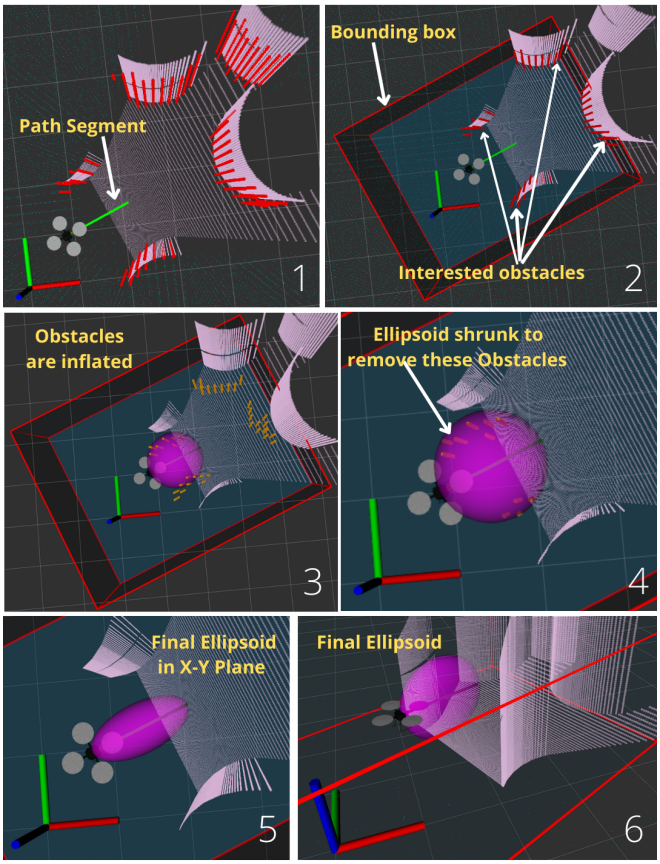

Fig. 8. Steps 1 through 6 showing an overview of how a sphere is shrunk forming an obstacle free ellipsoid around the segment

Next, we find the largest polyhedron that can be formed around the ellipsoid. The outward normal of a tangential half plane at point p on the ellipsoid $\lambda(C,d)$ can be defined by,

$$n = C^{-1} \cdot C^{-T}(p-d)$$

Where,

p is the closest obstacle to the ellipsoid.

A closest point where the inflated obstacle touches the ellipsoid is picked and a half plane is drawn. Then we discard all obstacles that lie outside the plane. Now the ellipsoid is dilated to touch the next obstacle inside the plane keeping its aspect ratio constant till the next nearest remaining obstacle is met. We form a new plane tangential to the ellipsoid there and remove all obstacles outside this half plane. The process is recursively continued till all obstacles are removed. The process of forming the polyhedron is shown in 9.

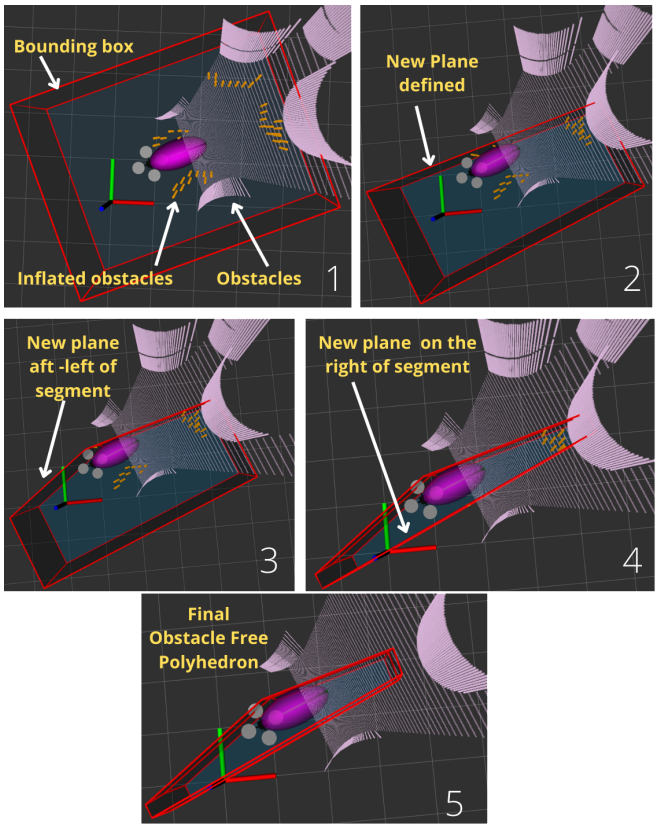

Fig. 9. Steps 1 through 5 showing an the obstacle free ellipsoid is dilated while forming planes that define the polyhedron

Finally, since the planes defined above do not guarantee a bounded polyhedron, the planes are clipped by the bounding box defined earlier, obtaining a bounded polyhedron.

The above steps are repeated for every segment of the JPS solution. A list of polyhedra in standard form defines a safe flight corridor around the JPS solution where the local planner can plan an optimized path trajectory.

D. Local Planner

In the final step we generate a smooth dynamically feasible path by parameterizing it as a set of cubic Bezier curves and solving an optimization problem to determine the set of control points that minimize a cost function while satisfying dynamic constraints. The expression of a cubic Bezier curve is given as follows:

$$\mathbf{P}(t) = \sum_{i=0}^{n} B_{n,i}(\frac{t}{\tau})\mathbf{P}_i \ , t \in [0,\tau]$$

Where $B_{n,i}(t)$ is the Bernstein basis polynomial of degree n. One useful property of a Bezier curve is the convex hull property, which gives that a Bezier curve lies within the convex hull of its control points[13]. As our polyhedron decomposition of the free space is convex, this means that if the control points for one segment all lie within a particular polyhedron then that segment is guaranteed to be collision free. This is an attractive property as it makes the process of collision checking much faster and allows it to be inexpensively incorporated into an optimization problem.

We thus divide the trajectory over our planning horizon into N Bezier curve segments with a certain number of segments allocated to each interval of the global path and its corresponding polyhedron. We select minimum jerk as a cost function and solve for a collision free trajectory that minimizes this cost without exceeding vehicle speed, acceleration and jerk limits.

$$J_{traj} = \int_0^{t_{plan}} \mathbf{J}_j^T(t)\mathbf{J}_j(t)\,dt \ , j \in [0, N-1]$$

We study two problem formulations to investigate the issue of time allocation highlighted earlier. We first evaluate the formulation of Tordesillas et al. in the FASTER algorithm[6]. This method assumes a fixed time allocation for the entire trajectory and hence each curve segment, but allows the curve segments to move between intervals through the use of binary variables that indicate in which polyhedron a segment lies. This interval allocation strategy creates additional flexibility in the path parameterization and a form of soft time allocation in that more segments can be placed in low speed regions and vice versa. The cost of this flexibility though is that the added binary variables promote the problem to a Mixed Integer Quadratic Problem(MIQP), which is more expensive than a QP and can in general require time proportional to 2^n for a problem with n binary variables[14]. The assumption of a fixed time allocation also means that this value must be computed externally.

As an alternate approach, we investigated using a fixed number of curve segments per interval but instead also adding the time of each segment as a decision variable. This both eliminates the binary variables and also allows adding time into the cost function to simultaneously derive a time allocation for the trajectory by minimizing a weighted sum of the jerk and time costs as follows:

$$J_{traj} = \int_0^{t_{plan}} (\mathbf{J}_j^T(t)\mathbf{J}_j(t) + \lambda)\,dt \ , j \in [0, n_{seg}]$$

We assume a state vector for the vehicle of $\mathbf{x}^T = [x^T, v^T, a^T]^T$, where x, v and a are the position, speed and acceleration of

the quadrotor respectively. Another useful property of Bezier curves is that their derivatives are themselves Bezier curves of one degree lower with control points that are a linear combination of the control points of the original curve[13]. To simplify the optimization problem when including time as a decision variable, we add the velocity, acceleration and jerk control points of the trajectory as decision variables as well. We can write the trajectories defined by our control point decision variables for each trajectory segment as:

$$\mathbf{P}_j(\tau) = \sum_{i=0}^{n} B_{n,i}(\frac{\tau}{\tau_j})\mathbf{P}_{i,j} \ , \mathbf{V}_j(\tau) = \sum_{i=0}^{n-1} B_{n-1,i}(\frac{\tau}{\tau_j})\mathbf{V}_{i,j}$$

$$\mathbf{A}_j(\tau) = \sum_{i=0}^{n-2} B_{n-2,i}(\frac{\tau}{\tau_j})\mathbf{A}_{i,j} \ , \mathbf{J}_j(\tau) = \sum_{i=0}^{n-3} B_{n-3,i}(\frac{\tau}{\tau_j})\mathbf{J}_{i,j}$$

Where $\mathbf{P}_j(\tau)$, $\mathbf{V}_j(\tau)$, $\mathbf{A}_j(\tau)$ and $\mathbf{J}_j(\tau)$ represent the position, velocity, acceleration and jerk respectively for the j^{th} trajectory segment and $\tau \in [0, \tau_j]$. The relation between the derivatives of Bezier curves additionally requires that the control points satisfy:

$$\mathbf{V}_{i,j} = \frac{3(\mathbf{P}_{i+1,j} - \mathbf{P}_{i,j})}{\tau_j} \ \forall \ i \in [0, n-1] \ , j$$

$$\mathbf{A}_{i,j} = \frac{2(\mathbf{V}_{i+1,j} - \mathbf{V}_{i,j})}{\tau_j} \ \forall \ i \in [0, n-2] \ , j$$

$$\mathbf{J}_{i,j} = \frac{(\mathbf{A}_{i+1,j} - \mathbf{A}_{i,j})}{\tau_j} \ \forall \ i \in [0, n-3] \ , j$$

The full optimization problem for both formulations is thus given as:

MIQP Formulation:

$$\min_{\mathbf{P}_{i,j}, \mathbf{V}_{i,j}, \mathbf{A}_{i,j}, \mathbf{J}_{i,j}, \mathbf{b}_{p,j}} \sum_{j=0}^{N-1} \tau_j \mathbf{J}_{0,j}^T \mathbf{J}_{0,j}$$

$$\text{s.t. } \mathbf{x}_0(0) = \mathbf{x}_{init} \ , \mathbf{x}_{N-1}(\tau_{N-1}) = \mathbf{x}_{final}$$

$$\mathbf{x}_j(\tau_j) = \mathbf{x}_{j+1}(0) \ \forall j \in [0, N-2]$$

$$\mathbf{F}_p \mathbf{P}_{i,j} \leq \mathbf{c}_p + M(1 - b_{j,p}) \ \forall i, j, p$$

$$3(\mathbf{P}_{i,j} - \mathbf{P}_{i+1,j}) + \tau_j \mathbf{V}_{i,j} = 0 \ \forall \ i \in [0, n-1] \ , j$$

$$2(\mathbf{V}_{i,j} - \mathbf{V}_{i+1,j}) + \tau_j \mathbf{A}_{i,j} = 0 \ \forall \ i \in [0, n-2] \ , j$$

$$\mathbf{A}_{i,j} - \mathbf{A}_{i+1,j} + \tau_j \mathbf{J}_{i,j} = 0 \ \forall \ i \in [0, n-3] \ , j$$

$$\|\mathbf{V}_{i,j}\|_\infty \leq v_{max} \ , \|\mathbf{A}_{i,j}\|_\infty \leq a_{max} \ , \|\mathbf{J}_{i,j}\|_\infty \leq j_{max} \ \forall i, j$$

$$\sum_{p=0}^{N_{int}-1} b_{j,p} \geq 1 \ \forall j \ , b_{j,p} \in \{0, 1\} \forall j, p$$

SQP Formulation:

$$\min_{\mathbf{P}_{i,j}, \mathbf{V}_{i,j}, \mathbf{A}_{i,j}, \mathbf{J}_{i,j}, \tau_j} \sum_{j=0}^{N-1} \tau_j \mathbf{J}_{0,j}^T \mathbf{J}_{0,j} + \lambda \tau_j$$

$$\text{s.t. } \mathbf{x}_0(0) = \mathbf{x}_{init} \ , \mathbf{x}_{N-1}(\tau_{N-1}) = \mathbf{x}_{final}$$

$$\mathbf{x}_j(\tau_j) = \mathbf{x}_{j+1}(0) \ \forall j \in [0, N-2]$$

$$\mathbf{F}_p \mathbf{P}_{i,j} \leq \mathbf{c}_p \ \forall i, p \ \forall j \in \text{interval}_p$$

$$3(\mathbf{P}_{i,j} - \mathbf{P}_{i+1,j}) + \tau_j \mathbf{V}_{i,j} = 0 \ \forall \ i \in [0, n-1] \ , j$$

$$2(\mathbf{V}_{i,j} - \mathbf{V}_{i+1,j}) + \tau_j \mathbf{A}_{i,j} = 0 \ \forall \ i \in [0, n-2] \ , j$$

$$\mathbf{A}_{i,j} - \mathbf{A}_{i+1,j} + \tau_j \mathbf{J}_{i,j} = 0 \ \forall \ i \in [0, n-3] \ , j$$

$$\|\mathbf{V}_{i,j}\|_\infty \leq v_{max} \ , \|\mathbf{A}_{i,j}\|_\infty \leq a_{max} \ , \|\mathbf{J}_{i,j}\|_\infty \leq j_{max} \ \forall i, j$$

Where $i \in [0, n]$ is the index of the Bezier curve control point, $j \in [0, N-1]$ is the index of the curve segment, $p \in [0, N_{int} - 1]$ is the interval or polyhedron index, n is the degree of the Bezier curve, N is the number of curve segments and N_{int} is the number of polyhedra received from the convex decomposition.

Note that $b_{j,p}$ is a binary variable indicating that curve segment j is assigned to polyhedron p. The constraint on the sum of binary variables across all polyhedra being greater than or equal to one forces a curve segment to be assigned to at least one interval. $\mathbf{F}_p$ denotes the matrix of normal vectors of the planes defining the p^{th} polyhedron and $\mathbf{c}_p$ their corresponding coefficients. The coefficient M is a large number that allows the polyhedron constraints to be trivially satisfied when the corresponding binary variable is inactive and was taken to be 1000 in our case. These binary variables are only present for the MIQP version as the alternate formulation assumes a fixed number of segments per interval.

In the case of the MIQP, as mentioned previously in the formulation of FASTER the time allocation for the entire trajectory is a fixed number and thus the trajectory time for each curve segment τ_j is a constant value. Between replanning iterations the time allocation is updated using a heuristic line search for a nondimensional time allocation scale factor f. For a given scale factor the time for each curve segment is calculated as:

$$\tau_j = f \cdot \max\{T_{v_x}, T_{v_y}, T_{v_z}, T_{a_x}, T_{a_y}, T_{a_z}, T_{j_x}, T_{j_y}, T_{j_z}\}/N$$

Where T_{v_i}, T_{a_i} and T_{j_i} are the solution times for the constant input motion equations from the initial to final condition for velocity, acceleration and jerk respectively achieved by applying the maximum allowable speed, acceleration and jerk in each axis independently. These values form a lower bound on the minimum possible time allocation based on the overall difference in state to be achieved and the scale factor can then account for the effect of the local environment.

This scale factor is adaptively updated between replanning iterations. At each replanning iteration the planner will try scale factors in a range of $[f_{k-1} - \gamma, f_{k-1} + \gamma']$ in ascending

order until it finds a value for which the problem is feasible. This new f_k value is then stored as a starting point for the next replanning iteration. This line search strategy allows the algorithm to progressively seek faster trajectory times based on the local environment conditions.

For the alternate formulation with segment times as a decision variable, the presence of a cubic term in the objective function and a product of terms in the spline derivative constraints makes the optimization problem no longer a QP. We solve the optimization problem iteratively by an SQP approach to find a trajectory that minimizes the time weighted cost subject to the same boundary conditions. The benefit of this approach is that it produces a time allocation as part of the solution process and does not require an outer loop line search.

E. ROS Implementation

The four major components mentioned above were implemented as a set of ROS nodes written in both Python and C++. The `Local_Planner` node interfaces with the MOSEK optimizer [15] to solve the optimization problem at each replanning iteration. Figure 10 below illustrates our architecture.

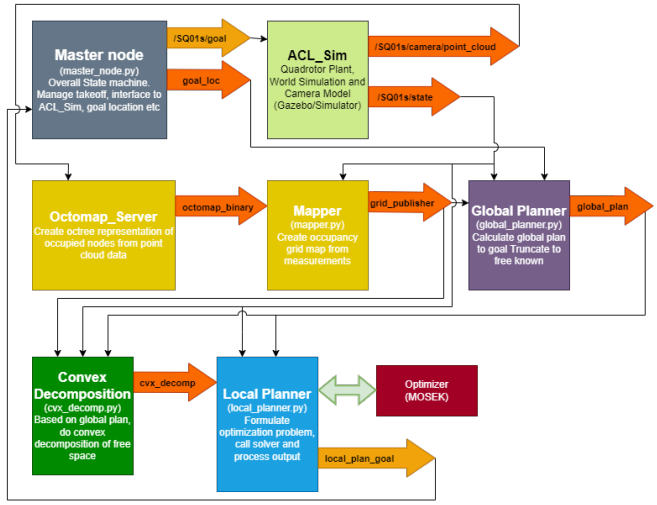

Fig. 10. ROS architecture of our hierarchical trajectory planner

For our project, the `octomap_server` ROS node was used to populate the occupancy octree from the point cloud data generated by the onboard stereo camera. This node is implemented as part of the `OctoMap` mapping project from [11].

We evaluate the performance of our planner in a Gazebo simulation environment. We made use of the `ACL_Sim` package forked from the FASTER GitHub repository[6], which provides a range of Gazebo simulation environments as well as vehicle and stereo camera models. Our code is available on our GitHub repository here: `https://github.com/sangitasahu/AER1516`

IV. Simulation Results

To evaluate the performance of our implementation we performed a variety of simulations. We first did a comparative study on the two proposed optimization problem formulations before performing full system flight tests in forest and office environments simulated using the `ACL_Sim` ROS package and Gazebo.

Due to challenges experienced in implementation of the global planner we could only test the full package in relatively simple environments. To push the capabilities of the convex decomposition and local planner we integrated the JPS3D implementation from the FASTER GitHub repository[6] to enable testing in more complex environments.

A. Optimization Problem Formulation

To compare the proposed optimization problem formulations for the local planner, we first tested both the SQP and MIQP formulations in a standalone Python script calling the MOSEK solver. We created three simple environments by selecting a start and goal location and then explicitly defining the polyhedron planes that would be produced by the convex decomposition. These consist of a simple right hand corner, a corner where the wall is close to the goal and an S shaped corridor. We then solved the respective optimization problems assuming zero velocity and acceleration at the start and goal locations as boundary conditions as well as vehicle limits of $v_{max} = 3m/s$, $a_{max} = 10m/s^2$, and $j_{max} = 50m/s^3$. The number of segments in the MIQP solution was fixed at 10 and runs were conducted at various time allocations. The number of segments per interval in the SQP formulation was varied between cases for reasons that will become apparent. Note that while the time allocation must be specified for the MIQP formulation, it is an output of the optimization for the SQP formulation.

The three cases are summarized in Table I below and shown graphically in Figure 11. Note that in Case 3 we have assumed three redundant intervals in the middle to study the robustness of both methods to variations in input conditions. Sample state trajectories for an optimal solution are shown in Figure 12 below.

TABLE I
Environments for Testing Problem Formulations

Case 1	Start: [0,0,0] m	Goal: [5,15,5] m
Interval 1	$x >= -2$	$x <= 1$
Interval 2	$y >= 10$	$z >= 3$
Case 2	Start: [0,0,0] m	Goal: [5,15,5] m
Interval 1	$x >= -2$	$x <= 1$
Interval 2	$y >= 14$	$z >= 3$
Case 3	Start: [0,0,0] m	Goal: [20,20,5] m
Interval 1	$x >= -2$	$x <= 2$
Interval 2-4	$y >= 7$	$y <= 12$
Interval 5	$x >= 15$	$z >= 3$

The first case represents a simple, shallow right hand corner. For this case we studied using 5 segments per interval for the SQP formulation and time allocations of 10 s and 7 s for the

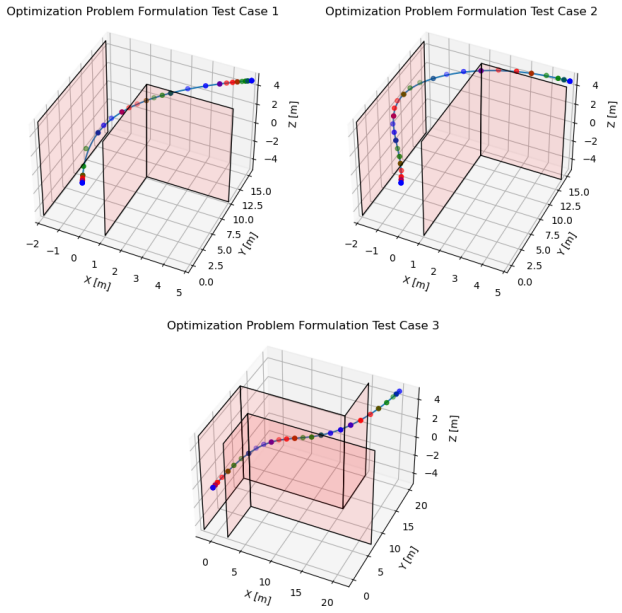

Fig. 11. Environments for testing problem formulations

MIQP. The resulting trajectory and solution times are given in Table II below.

TABLE II

CASE 1: TRAJECTORY AND SOLUTION TIMES

Problem	Trajectory Time [s]	Average Solve Time [ms]	Std Dev [ms]
SQP 5 segments	8.2	84.4	5.6
MIQP	10.0	43.6	0.7
MIQP	7.0	49.4	8.6

Both formulations successfully return a solution to the optimization problem in a reasonable timeframe. While the MIQP is faster at first glance, it should be noted that this is only a single solution at a given time allocation whereas the SQP formulation solves approximately 10 iterations and returns the time allocation. The MIQP would require multiple outer loop iterations to determine a suitable time allocation. The MIQP also finds a successful solution at a faster trajectory time of 7 s compared to the 8.2 s found by the SQP method. This difference is likely due to the required cost function simplications for the SQP approach described in the previous section.

The second case is a more severe corner where the wall is very close to the goal, requiring a sharp turn for a high speed trajectory. In this case we studied using both 5 and 15 segments per interval for the SQP approach and time allocations of 15 s and 8 s for the MIQP. The resulting trajectory and solution times are given in Table III below.

This more severe trajectory becomes challenging for the SQP approach. With 5 segments per interval the problem became infeasible despite the fact that the MIQP is able to solve

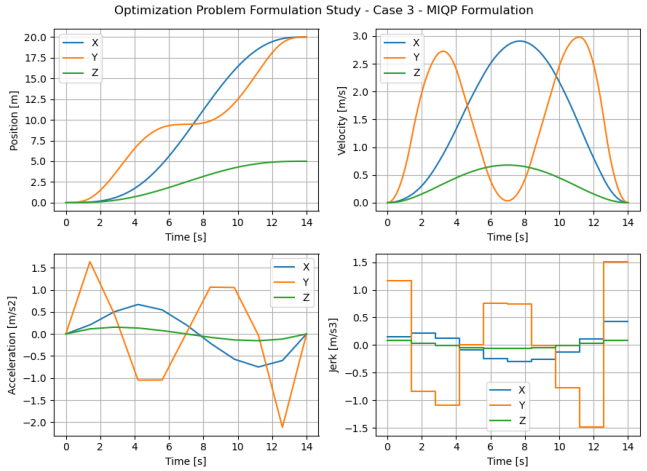

Fig. 12. State trajectories for the MIQP problem formulation on the S shaped corridor of Case 3

TABLE III

CASE 2: TRAJECTORY AND SOLUTION TIMES

Problem	Trajectory Time [s]	Average Solve Time [ms]	Std Dev [ms]
SQP 5 segments	Infeasible		
SQP 15 segments	15.0	307.8	15.0
MIQP	15.0	53.9	13.4
MIQP	8.0	25.1	5.3

it with the same number of segments. It was necessary to increase the number of segments to 15 per interval(for a total of 30) in order for it to find a feasible solution. This difference is due to the fact that the SQP method has a fixed number of segments per interval, whereas the MIQP approach can reallocate the segments between intervals as needed. The MIQP problem is also able to find a much faster feasible trajectory at 8.0 s compared to 15 s for the SQP at a significantly reduced solve time, even when considering that multiple MIQP iterations would need to be performed to determine the time allocation. The additional flexibility of interval allocation improves the robustness of this formulation given the limited expressiveness of a cubic curve.

The last case studied was an S shaped corridor with 5 intervals of which the three in the middle are duplicates. This case was studied both as a more difficult problem as well as to understand the robustness of the formulations to errant inputs. For this case we tried both 3 and 5 segments per interval for the SQP formulation and a time allocation of 14 s for the MIQP. The resulting trajectory and solution times are given in Table IV below.

TABLE IV

CASE 3: TRAJECTORY AND SOLUTION TIMES

Problem	Trajectory Time [s]	Average Solve Time [ms]	Std Dev [ms]
SQP 3 segments	12.7	167.0	1.2
SQP 5 segments	15.7	311.0	16.5
MIQP	14.0	132.7	5.9

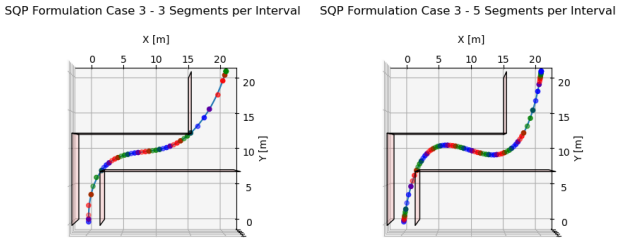

Fig. 13. SQP predicted trajectories for Case 3 with 3 and 5 segments per interval

It can be seen that again the SQP formulation struggles with this more complicated scenario. Increasing the number of segments per interval from 3 to 5 surprisingly increases the trajectory time from 12.7 to 15.7 s and leads to a different solution, even though it is fully capable of simply representing the 3 segment solution. Figure 13 below compares the two trajectories, where it can be seen that the trajectory with 5 segments per interval also has an unnecessary recurve shape to it and appears to be overparameterized.

The MIQP formulation on the other hand does not suffer from this issue. While its computation time does increase significantly from approximately 50 ms previously to 133 ms due to the redundant intervals, it is still able to robustly find a solution with the same parameters as in the previous cases. Interestingly, it was necessary to increase the trajectory time to 14 s to find a feasible solution, which is slightly higher than the SQP solution. It is still almost as expensive as the full SQP solution for a single iteration but does not return a time allocation.

Overall, while the SQP method is cheaper per iteration and returns a time allocation it shows poor robustness to more challenging environments. The sharp second corner case required 15 segments per interval to find a feasible solution, whereas we saw in the third case that increasing the segments per interval beyond 3 lead to higher time solutions with excessive curvature. Having a fixed number of segments per interval limits the expressiveness of the trajectories and makes it difficult to select an appropriate number given that all environments will not be known apriori.

The ability of the MIQP formulation to reallocate trajectories between intervals makes it significantly more robust to input conditions, even if the problem solution time is longer and it does not return a time allocation. For these reasons and due to time constraints we only implemented the MIQP in our ROS local planner node.

B. Global Planner Simulation Results

We tested the global planning algorithm in two forest environments of varying densities named as Forest 1 and Forest 2 shown in Figure 14. The start and goal locations for Forest 1 are [0,0,1] and [13,13,1] and the same for Forest 2 are [0,0,2] and [60,60,2] respectively. For all simulations, the 3D occupancy grid dimensions were set to 12m x 12m x 4m, the velocity of the quadrotor was set to 3 m/s and the global planner was run at 5 Hz.

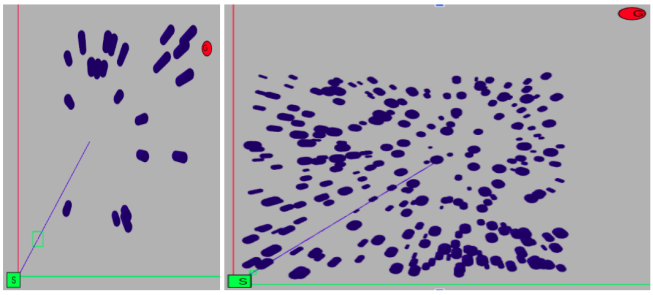

Fig. 14. Forest environments (Left : Forest 1 and Right : Forest 2)

Two parameters of JPS are of utmost importance, as they decide the quality of the JPS solution and its adherence to the timing constraints. One is the maximum number of node expansions allowed during each JPS search, which was set to 500 for Forest 1 and 1000 for Forest 2. The other parameter is the maximum number of successful paths the JPS is allowed to find before selecting the optimal path, and it was set to 100 for Forest 1 and 50 for Forest 2. The values of the parameters depend on the density of the environment and should be chosen carefully to ensure that the JPS always finds a feasible solution. To add, the larger the parameters are, the better the quality of the JPS solution is, and the slower the execution is.

TABLE V
PATH LENGTHS AND EXECUTION TIMES

Grid Resolution [m]	Path Length [m]	Average JPS Execution Time [s]	Total Flight Time [s]	Average Speed [m/s]
Forest 1				
0.5	20.7	0.56	12.0	1.72
0.25	40.8	6.81	40.3	1.02
0.125	54.2	28.39	64.8	0.84
Forest 2				
0.25	129.4	39.79	180.7	0.72

The performance of JPS also depends on the resolution of the occupancy grid. A finer resolution leads to a higher solution time and more segments in the global plan, which can increase the total path length. On the Forest 1 simulated environment, the effects of varying grid resolution were studied. The flight path lengths, execution times, total flight times and average flight speeds for both Forest 1 and Forest 2 are tabulated in Table V above. The best average speed is obtained for Forest 1 with a grid resolution of 0.5 m which is approximately 1.72 m/s and the average speed achieved for Forest 2 with a grid resolution is around 0.72. It can be seen in Table V that the average JPS execution time increases with a finer grid resolution. It is also observed that when the grid resolution increases, the average speed attained by the quadrotor decreases and the path lengths, execution times and flight times increase. It is mostly due to the increase in the density of the 3D occupancy grid map and the increase in the number of nodes searched before finding an optimal path.

Undoubtedly, the lower the resolution is, better the quality of the JPS solution is. However it comes with slower execution rates due to increase in the number of JPS computations.

Simulation results for the Forest 1 and Forest 2 environments are shown in Figure 15 and Figure 16 respectively. The blocks in the images are marker arrays that represent the occupied grid cells and the green curve extending from the start to goal is the executed global plan. For both the forests scenarios, the performance of the global planner is found satisfactory. The videos of some of the RViz simulations can be accessed **here**.

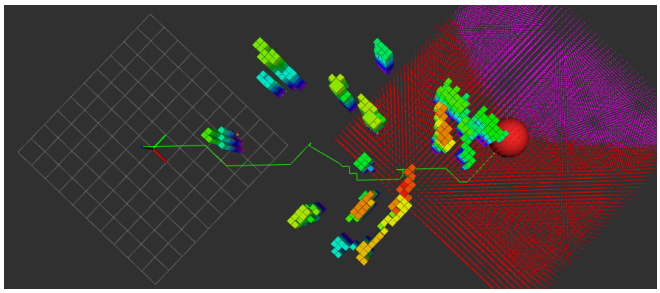

Fig. 15. Forest-1 simulation results

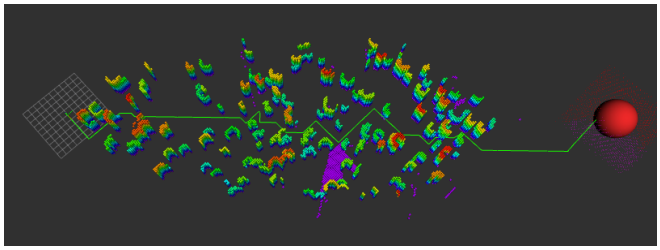

Fig. 16. Forest-2 simulation results

C. Convex Decomposition and Local Planner Results

The JPS3D [12] module is built in C++ and is a robust module that is fast enough to provide realtime JPS solutions for agile path planning applications. We imported JPS3D into our ROS environment to test the performance of the convex decomposer and the local planner. We tested the setup in a simple forest environment as well as a complex office environment with densely populated obstacles. We assumed vehicle dynamic limits of $v_{max} = 3\ m/s$, $a_{max} = 10\ m/s^2$ and $j_{max} = 50\ m/s^3$. We measured the travel time, path lengths, maximum speed attained by the quadrotor. The results are given in Table VI.

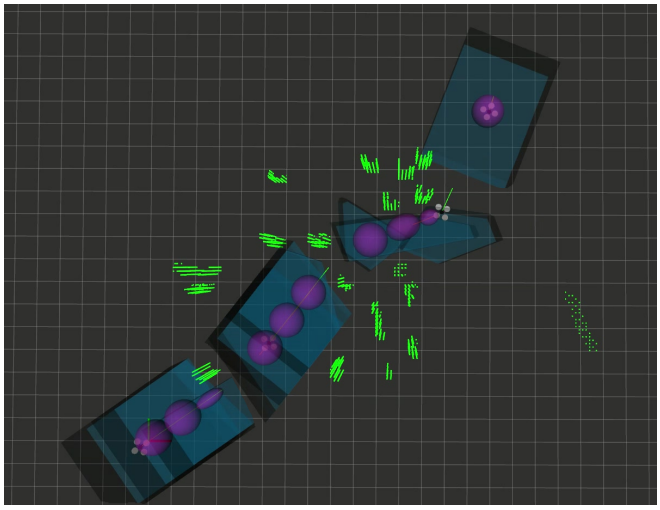

Fig. 17. Visualization of the full path taken by our planner through the forest environment

TABLE VI

LOCAL PLANNING SIMULATION RESULTS

Environment	Travel Time [s]	Path Length [m]	Maximum Speed [m/s]
Small forest	10.0	21.1	4.20
Office	34.1	38.4	3.84

The average time taken by the convex decomposition module to generate a safe flight corridor and the average re-planning time for the local planner are given in Table VII. These execution times were obtained running on an Intel i7-2600k desktop CPU.

TABLE VII

LOCAL PLANNING EXECUTION TIMES

Environment	Convex Decomposition [ms]		Local Planner [ms]	
	Average	Std Dev	Average	Std Dev
Small forest	108.2	79.2	95.0	80.3
Office	248.4	210.8	153.5	115.1

It is seen that our algorithms for convex decomposition and local planner perform well in simple environments like the small forest. However, with increasingly complex environments, we see a higher planning execution times both for the convex decomposer as well as the local planner. The with safe flight corridors not being generated fast enough and the local planner taking more time to generate an optimum flight plan, the quadrotor suffers a performance saturation in complex environments. Figure 17 shows a concatenated view of the path taken by the quadrotor in a forest environment and Figure 18 the path taken through the office environment.

V. DISCUSSION

The project required integration of all the modules shown in Figure 10. While individual modules were built independently, new insights on functionality and requirements were drawn during integration. While many functionalities were developed and included in the project, some key observations were noted and certain challenges are worth mentioning here.

Occupancy Map

The OctoMap package we used for this project worked very well for identifying occupied spaces. It however did not give a clear representation of the free and unknown spaces. This problem was mitigated by using the quadrotor state and camera FOV information to generate known and unknown spaces within the mapping grid.

We then used a ray cast operation to identify occluded areas behind obstacles as unknown. This implementation was

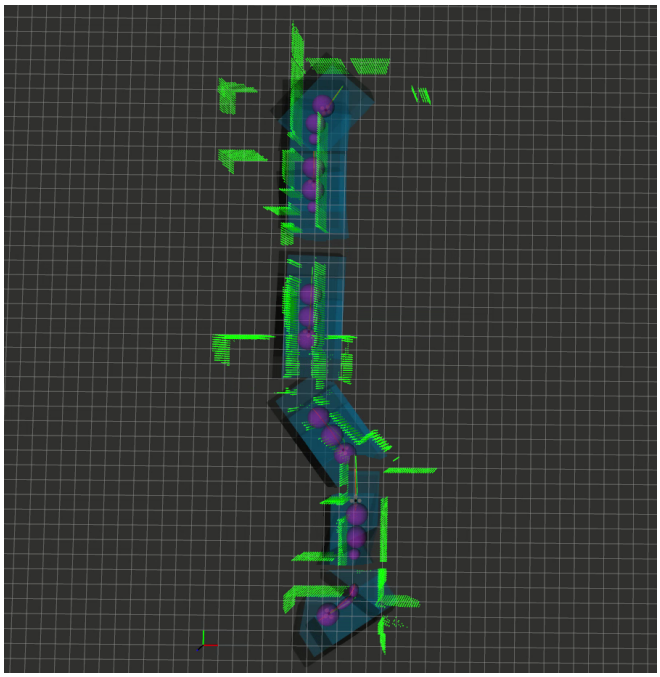

Fig. 18. Visualization of the full path taken by our planner through the office environment

however not very robust as it often led to an unknown region being classified as known. This could interfere with the JPS solution where the planner could end up planning in the unknown space where there could be obstacles. The risk of collision was however minimum as an obstacle would come into view as the quadrotor got closer and the JPS would re-plan to avoid the obstacle.

Global Planner

The Global Planner node was noticed to have cases where it could not consistently find JPS solutions at all time steps. The execution time of the global planner increased drastically when run on finer grid resolutions from the mapper due to limited computation power. For example, with a grid of 12x12x4 size, at 0.125 resolution, JPS has to search in a map containing around 100,000 nodes which is quite computationally expensive. In addition, a couple of parameters were to be tuned to successfully run JPS for different environments depending on their density in order to meet all the timing requirements. Thus, a faster and generalized implementation of the global planner was found necessary.

Convex Decomposition

The convex decomposition recursively dilates the ellipsoids to eliminate all the inflated obstacles around it. This method was found to form planes that have comparably similar normals as shown in Figure 19 . While this results in a robust decomposition, it increases the number of planes in the polyhedron. This increases re plan time for the local planner. The algorithm for defining the half planes could be softened. Similar planes with angle subtended below a preset limit could be merged to form a simpler plane that safely

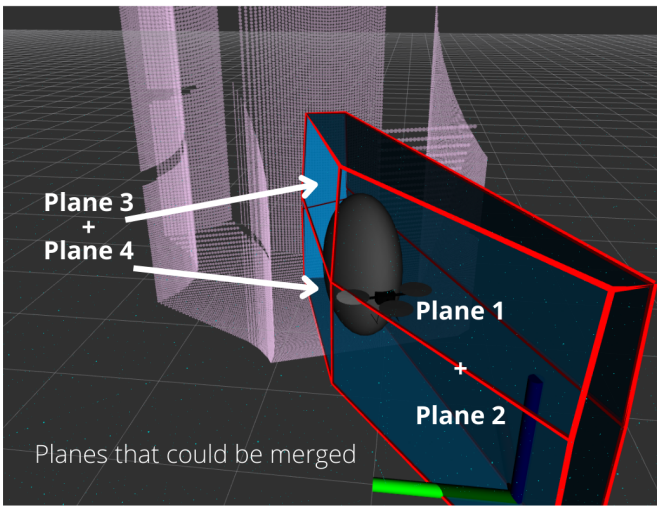

Fig. 19. Plane-1,Plane-2 could be merged to form a simpler plane. similarly Plane-3,Plane-4 could be merged too.

eliminates the obstacles in question while trading safe-fly-volume for computational simplicity of the polyhedron. This is a possibility that could not be fully explored in this project.

Local Planner

In our implementation of the local planner we found the use of infeasibility as a criterion for adapting the time allocation factor as proposed by the original FASTER paper to not be a robust solution. It was found that this strategy could lead to cases where the solver progressively makes the time allocation factor more and more aggressive until the velocity traces begin to saturate over long portions of the planning horizon as illustrated in Figure 20 below. At these points the vehicle is moving so quickly relative to its environment that eventually the optimization problem becomes infeasible without violating the speed constraints and requires large cutbacks on the time allocation factor. The result is an oscillatory mode where the vehicle speeds up before periodically stopping to remain within the vehicle limits.

The original authors did not mention this behaviour so while it may be an issue in our implementation, based on our reasoning here we believe it may be inherent to the method. We briefly investigated the use of soft constraints but could not find a robust implementation in the time allotted. This observation also supports the validity of our attempt to incorporate time into the objective function, which would solve this issue by simply allowing a small increase on cost.

Another challenge that arose was the effect of the local plan on observability of the world. The optimization problem as formulated does not consider observability and makes no attempt to keep the camera level or achieve low angular rates to reduce motion blur. As such we found that the resulting plans in some cases commanded large vehicle angles briefly that could cause the convex decomposition to fail due to

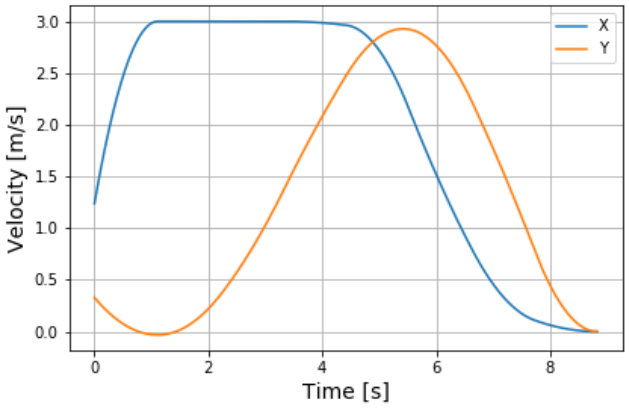

Fig. 20. Example of saturated vehicle velocities resulting from aggressive time allocation adaption strategy that eventually leads to infeasibility

poor point cloud data. An objective function formulation that considers these system level interactions could produce an overall more robust architecture.

For speed and ease of implementation most of our ROS nodes were written in Python. It is well known that Python programs can be much slower than those written in compiled languages such as C++. The global planner, convex decomposition and local planner nodes all produced high replanning times that could have been improved by implementing in a faster language such as C++.

VI. Conclusion and Future Work

In conclusion, we studied the use of hierarchical trajectory planners for the problem of fast quadrotor flight. We adopted FASTER as a reference algorithm due to its good balance of execution speed, environment representation and performance and proposed an alternate optimization problem formulation that added segment times as a decision variable to address the time allocation problem. We performed a comparative study of our proposed SQP formulation against the original MIQP of FASTER and found that while it performed similarly well on simple environments, on more aggressive corners and tight corridors the flexibility of the MIQP to reallocate curve segments between planning intervals made it more robust.

We implemented the four major components including the mapper, global planner, convex decomposition and local planner in a set of ROS nodes and evaluated them in Gazebo simulations using the ACL_Sim simulator[6]. Challenges with execution speed prevented us from testing all four components together. We tested flying the quadrotor based on the global plan alone and found that it was able to successfully navigate to the goal location in forest environments of varying density. We then integrated an existing JPS3D implementation[6] into our framework to test the performance of the convex decomposition and local planner. We simulated flight in forest and office environments and found we were able to achieve maximum speeds of up to 4.2 m/s and replanning rates nearing 10 Hz on average in the forest case. We experienced a number of challenges and identified areas for future work including simplifying the resulting polyhedra from the convex decomposition, incorporating observability into the planning objective function and a more robust time allocation methodology.

References

[1] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[2] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[3] M. Ryll, J. Ware, J. Carter, and N. Roy, "Efficient trajectory planning for high speed flight in unknown environments," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 732–738.

[4] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 2872–2879.

[5] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-based motion planning for aggressive flight in SE(3)," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2439–2446, 2018.

[6] J. Tordesillas and J. P. How, "FASTER: Fast and safe trajectory planner for navigation in unknown environments," *IEEE Transactions on Robotics*, 2021.

[7] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, 2019.

[8] C. Richter, A. Bry, and N. Roy, *Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments*. Cham: Springer International Publishing, 2016, pp. 649–666. [Online]. Available: https://doi.org/10.1007/978-3-319-28872-7_37

[9] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, 02 2017.

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

[11] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, 2013, software available at https://octomap.github.io.

[12] D. Harabor and G. Alban, "Online graph pruning for pathfinding on grid maps," in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011, pp. 1114–1119.

[13] C. Shene, "CS3621 Introduction to Computing with Geometry Notes," https://pages.mtu.edu/~shene/COURSES/cs3621/NOTES/, 2014, accessed: 09-Apr-2021.

[14] "MOSEK optimizer for mixed-integer problems," https://docs.mosek.com/latest/pythonapi/mip-optimizer.html, 2022, accessed: 09-Apr-2021.

[15] Mosek ApS, "The MOSEK optimization software." [Online]. Available: https://www.mosek.com/