drones

MDPI

Article

Quadrotor Path Planning and Polynomial Trajectory Generation Using Quadratic Programming for Indoor Environments

Muhammad Awais Arshad, Jamal Ahmed and Hyochoong Bang *

Department of Aerospace Engineering, Korea Advanced Institute of Science and Technology, Daejeon 34141, Republic of Korea
* Correspondence: hcbang@kaist.ac.kr

Abstract: This study considers the problem of generating optimal, kino-dynamic-feasible, and obstacle-free trajectories for a quadrotor through indoor environments. We explore methods to overcome the challenges faced by quadrotors for indoor settings due to their higher-order vehicle dynamics, relatively limited free spaces through the environment, and challenging optimization constraints. In this research, we propose a complete pipeline for path planning, trajectory generation, and optimization for quadrotor navigation through indoor environments. We formulate the trajectory generation problem as a Quadratic Program (QP) with Obstacle-Free Corridor (OFC) constraints. The OFC is a collection of convex overlapping polyhedra that model tunnel-like free connecting space from current configuration to goal configuration. Linear inequality constraints provided by the polyhedra of OFCs are used in the QP for real-time optimization performance. We demonstrate the feasibility of our approach, its performance, and its completeness by simulating multiple environments of differing sizes and varying obstacle densities using MATLAB Optimization Toolbox. We found that our approach has higher chances of convergence of optimization solver as compared to current approaches for challenging scenarios. We show that our proposed pipeline can plan complete paths and optimize trajectories in a few hundred milliseconds and within approximately ten iterations of the optimization solver for everyday indoor settings.

Keywords: path planning; quadratic programming; trajectory generation; optimization; obstacle-free corridors

Citation: Arshad, M.A.; Ahmed, J.; Bang, H. Quadrotor Path Planning and Polynomial Trajectory Generation Using Quadratic Programming for Indoor Environments. *Drones* **2023**, *7*, 122. https://doi.org/10.3390/drones7020122

Academic Editors: Yu Wu and Liguo Sun

Received: 29 December 2022
Revised: 31 January 2023
Accepted: 6 February 2023
Published: 9 February 2023

1. Introduction

A generic motion planning framework for mobile robots can be divided into map acquisition, path planning, trajectory generation, trajectory optimization, and trajectory tracking. Map acquisition is the process of acquiring and representing the spacial environment (workspace) of robots in a structured way. Commonly used structured maps include occupancy grid maps [1], octomaps [2], point cloud maps [3], Voronoi diagram maps [4], Euclidian signed distance fields [5,6], etc. The quality and accuracy of these maps directly influence the path planning performance. The goal of path planning is to find piecewise linear segments from a current point to a target point in a robot's configuration space (C-Space) while avoiding obstacles. This path can be represented by a set of points (called waypoints or knot-points) joining those linear segments. We may broadly classify path planning algorithms into Graph-Search-Based Algorithms (GSBAs) and Sampling-Based Algorithms (SBAs) [7]. Commonly used algorithms for both graph-search-based algorithms, as well as sampling-based algorithms, are shown in Figure 1. Path planning algorithms usually consider the geometric constraints of the robot's workspace but ignore the dynamical limitations of the robots. Therefore, the final segmented paths suggested by the path planner might not be executable by the actual robot. The trajectory generation and optimization process considers the dynamics of the mobile robot along with other constraints (e.g., safety constraints, kino-dynamic constraints, etc.) and incorporates

the time-allocation mechanism to empower the motion planner with a better executable and optimal trajectory. Therefore, by cascading path planning with trajectory generation and optimization, a motion planner can finally generate kino-dynamic-feasible, collision-free, executable, and trackable trajectories that satisfy several optimization constraints and objectives.

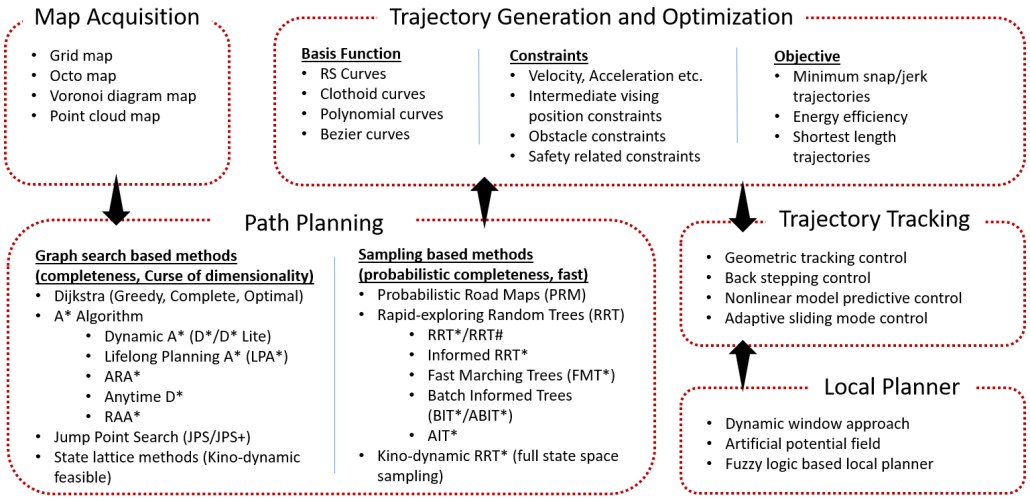

Figure 1. Motion planning framework for mobile robots. Symbols ($*$, #) are part of the name of the method. RRT-Star (RRT$*$) is an improved version of RRT.

Motion planning of quadrotors for indoor environments is a challenging problem in terms of mapping (obstacles of different sizes and shapes), path planning (relatively limited free spaces), and trajectory optimization (challenging constraints). We consider the indoor environment as an enclosed Euclidean space in $\mathbb{R}^3$ with possible partitioning of its interior into multiple sections (rooms) by thin walls. Nieuwstadt et al. [8] considered the problem of real-time trajectory generation and tracking for nonlinear differentially flat systems and formulated the problem as a Quadratic Program (QP). Mellinger et al. [9] also formulated the problem with QP by parameterizing the trajectory as a kth order polynomial. Earlier works focused on modeling the obstacles in the environment as convex objects by imposing integer constraints and solving the problems with Mixed Integer Linear Programming (MILP) or Mixed Integer Quadratic Programming (MIQP) solvers [10–12]. These mixed integer-based approaches are impractical because we need to impose multiple constraints for every single obstacle in the environment. We also need to consider that all the obstacles in the environment can be represented with single or multiple convex polyhedra. Moreover, integer constraints placed in optimization problems require MILP/MIQP solvers which are relatively slow for real-time performance requirements [10–12]. These limitations led to the expansion of quadrotor motion planning work in two directions. First, new approaches were developed to remove the integer constraints and solve the resulting optimization problems as QPs [13–15]. Second, researchers preferred modeling free spaces instead of modeling every single object in the environment and used those models of free spaces as inequality constraints in a quadratic programming solver [14–16].

Richter et al. [13] built upon the work of Mellinger et al. [9] and solved for the optimal trajectory in closed form by imposing constraints on higher derivatives of trajectories at fixed locations (waypoints) but their method does not provide any guarantees for trajectory generation in fixed time-span in complicated map scenarios. Chen et al. [14] modeled free spaces through the environment with axes-aligned cubes. This formulation may lead to an excessively large number of constraints because of the unnatural modeling of free space and reduces the chances of convergence during the optimization step. This approach may only be suitable for environments where all obstacles are rectangular parallelopipeds. Inspired by birds flying through cluttered environments such as dense forests, Karaman et al. [17]

analyzed the high speed motion planning through a random obstacle field but they did not consider non-trivial robot dynamics of quadrotors and their results may not be applicable to quadrotors. Liu et al. [18] built upon the work of Deits et al. [16] and formulated trajectory generation as a quadratic program using the collection of convex overlapping polyhedra that model free space. They used the Jump Point Search (JPS) [19] algorithm as a guide to generate convex overlapping polyhedra (Safe Flight Corridors) to model free space. The JPS algorithm forces the path to pass in close proximity to obstacles which may impose tight constraints in an optimization problem. JPS algorithm may also lead to generating paths with very small segments near thin obstacles as shown in Figure 2. Polyhedra generated by short segments again impose very challenging constraints on the optimizer and considerably reduce the chances of convergence. Zhou et al. [20] tried to solve the problem associated with the close proximity to the guided path to the obstacles by modifying the optimization objective function using Euclidean Distance Field as a guide but ended up converting the problem into a non-quadratic one, which is much slower to solve.

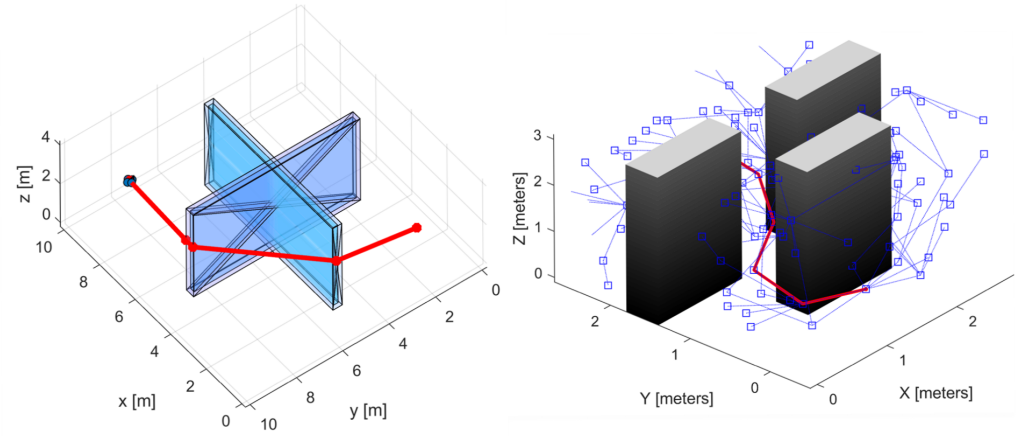

Figure 2. Robotic path planning with the GBSA (JPS-3D) algorithm on the left and the SBS (RRT∗) algorithm on right. The planned path through the environment is comprised of piece-wise line segments shown in red. Note the considerably small length of one segment in the path planned with the JPS-3D algorithm in close proximity to the thin wall.

In QP-based methods described above, safety guarantees are usually obtained by having a local planner that plans a trajectory with a stop condition in the free-known local space and replans the upcoming trajectory after more free space is known by following the previously generated trajectory. This, however, leads to slow and conservative trajectories. Tordesillas et al. [21] tried to address this problem by proposing FASTER (Fast and Safe Trajectory Planner), which is based on the MIQP formulation again. FASTER obtains high-speed trajectories by enabling the local planner to optimize in both the free-known and unknown spaces and provides safety guarantees by always having a feasible, safe backup trajectory in the free-known space at the start of each replanning step. Now, it is important to mention some recent work on trajectory planning for multiple quadrotors in dynamic environments. Tordesillas et al. [22] proposed MADER to plan trajectories for multi-agents in dynamic environments. A distinctive feature of MADER, in comparison to previous approaches, is that it performs real-time collision avoidance with other dynamic obstacles by performing outer polyhedral representations of every interval of the trajectories and then includes the plane that separates each pair of polyhedra as a decision variable in the optimization problem. Since they used outer polyhedral representations of every interval of the trajectories instead of some control points on the trajectory to formulate the optimization problem, their method provides better guarantees but, again, at the cost of non-quadratic formulation of the problem which is slower and less likely to converge as compared to the quadratic problem. Another recent work on the swarm of micro flying robots in the wild is proposed by Zhou et al. [23], and is based on their previous work

on EGO-Planner [24]. EGO-Planner is unique from other mainstream methods because it formulates the collision term in an objective function of the optimization problem by comparing the colliding trajectory with a collision-free guiding path. This objective function of EGO-Planner is free from ESDFs (Euclidean Signed Distance Fields) used previously by the same authors. An anisotropic curve fitting algorithm is introduced to adjust higher-order derivatives of the trajectory while maintaining the original shape. Like other high-end multi-agent techniques, this method does not formulate the optimization problem as a quadratic programming problem.

We adopt some ideas from related works [9,13,16,18] and improve upon the techniques to generate optimal, kino-dynamic-feasible, obstacle-free, and executable trajectories for quadrotors with applications to indoor settings. We limit ourselves to the quadratic programming formulation of the trajectory optimization problem as opposed to Gao et al. [15] and Tordesillas et al. [21,22], so that we may benefit from the convergence and computational efficiencies of quadratic programming formulation. We propose an algorithm that improves upon the limitations of current techniques with an emphasis on improving optimization performance in complex scenarios. Our pipeline uses a linear piecewise path from the path planning algorithm and improves this path further with our path relocation algorithm to guide the convex decomposition of the map to find an Obstacle-Free Corridor (OFC) from the current configuration to the goal configuration of a robot. We formulated the trajectory generation problem as a quadratic program and treated OFCs as inequality constraints in QP. By minimizing the snap (a fourth derivative of position) of trajectories, we made sure that the fourth-order dynamical system is able to follow our generated trajectory through a nonlinear controller. The total time for trajectory generation using our trajectory planning pipeline is sufficiently small and we may use it in real time for a navigation system with mapping and state estimation. We tested our procedure in simulation with diverse indoor environments. Four distinguishing features of our work are summarized below.

1. We formulate the trajectory planning problem as a quadratic program, which is more likely to converge in real time as compared to non-quadratic approaches.
2. Our path relocation algorithm along with the obstacle-free-corridor construction procedure provides better optimization performance as compared to mainstream techniques.
3. The safety and completeness of the algorithm make it attractive for real-world applications
4. We provide a comprehensive control pipeline along with the trajectory planning pipeline and test complete architecture in simulations.

The outline of this article is as follows: in Section 2, we describe the system model; in Section 3, we explain and compare different path planning schemes; in Section 4, the trajectory planning pipeline is discussed; in Section 5, we present our control strategy; simulation results are discussed in Section 6; conclusions follow in Section 7.

2. System Model

We consider a development drone (VOXL m500 drone manufactured by ModelAI, San Diego, CA, USA) as the reference platform in this research work with its coordinate system as shown in Figure 3. Body frame, B, and camera frame, C, are fixed to the body of the quadrotor while the world frame, W, is an inertial frame. We use ZXY sequence of rotation to define Euler angles yaw, ϕ, roll, θ, and pitch, ψ. We define the rotation matrix from the body frame, B, to the world frame, W, as $\mathbf{R}_{WB} = \mathbf{R}_{WC}\mathbf{R}_{CB}$, where $\mathbf{R}_{WC}$ defines the orientation of the camera frame, C, in world frame, W, while $\mathbf{R}_{CB}$ defines the orientation of the body frame, B, in the camera frame, C. Our system uses four motors $[m_1, \cdots, m_4]$ with propellers rotating in specified directions with angular velocities $[\omega_1, \cdots, \omega_4]$. Forces $[F_1, \cdots, F_4]$, and momenta $[M_1, \cdots, M_4]$ produced by the propellers are related to angular velocities $[\omega_1, \cdots, \omega_4]$ by $F_i = k_F\omega_i^2$, $M_i = k_M\omega_i^2$, where k_F and k_M are force coefficients and momentum coefficients of the propeller, respectively. We denote the angular velocity of the robot by $\omega_B^W = p\mathbf{x}_B + q\mathbf{y}_B + r\mathbf{z}_B$, where p, q, and r are the components of velocity

along the x-axis, y-axis, and z-axis of the body frame B. We define the kinematics of the quadrotor as:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} cos(\theta) & 0 & -cos(\phi)sin(\theta) \\ 0 & 1 & sin(\phi) \\ sin(\theta) & 0 & cos(\phi)cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{1}$$

We may exert different forces and moments on quadrotors with the help of propellers and derive the dynamics of the quadrotor with the help of Newton–Euler equations as:

$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -m\mathbf{g} \end{bmatrix} + \mathbf{R}_{WB} \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \tag{2}$$

$$\mathbf{I} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} L(-F_1 + F_2 + F_3 - F_4) \\ L(-F_1 + F_2 - F_3 + F_4) \\ \sqrt{2}(-M_1 - M_2 + M_3 + M_4) \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \mathbf{I} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3}$$

where m denotes the mass of the vehicle, $\ddot{\mathbf{r}}$ represents the acceleration of the center of the mass of the quadrotor in the inertial frame of reference, and $\mathbf{g}$ denotes acceleration due to gravity. The distance between the axis of rotation of the propeller and the center of mass of the vehicle is denoted by L. The constant $\mathbf{I}$ represents the moment of inertia matrix of the quadrotor referenced to the center of the mass and along the body-fixed frame of reference.

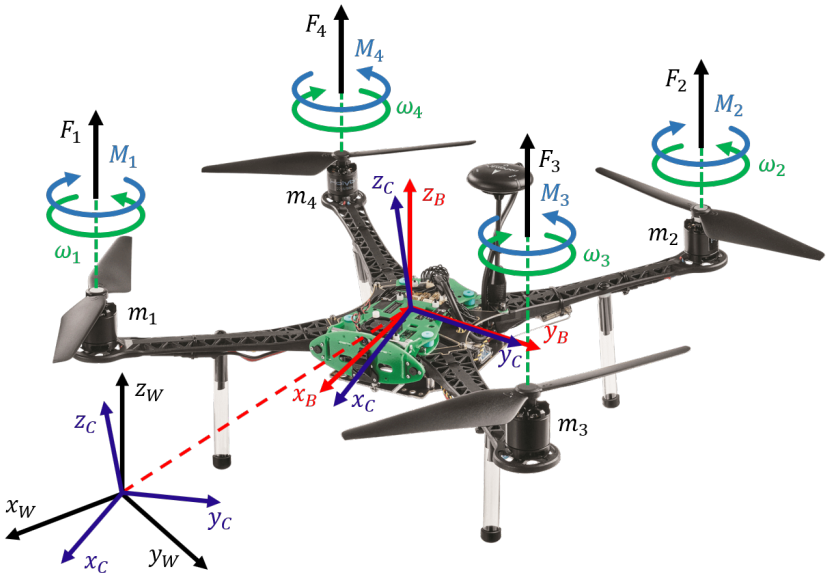

Figure 3. Development drone (Manufactured by ModelAI, San Diego, CA, USA) with our Coordinate system. We consider three frames of reference namely World Frame, W (shown in black), Body Frame, B (shown in red), and a Camera Frame, C (shown in blue), with relevant axes $[x, y, z]$. Forces, $[F_1, \cdots, F_4]$, moments, $[M_1, \cdots, M_4]$, and angular velocities, $[\omega_1, \cdots, \omega_4]$, produced by rotors are shown with black, blue, and green arrows respectively.

When we apply external force or torque to a quadrotor, it is applied with the help of motors in the body-fixed frame of reference. We control the angular velocity of the motors with the help of the pulse-width-modulated signal from the controller and it correspondingly applies forces and torques on the robot in the body-fixed frame of reference. Therefore, it is important to represent forces and moments in the body-fixed frame of reference in terms of the angular velocities of rotors. We represent control input to the system as a 4×1 vector $\mathbf{u} = [u1, u2, u3, u4]^T$, where u_1 is the net body force and u_2, u_3, u_4

are the body moments in the body-fixed frame of reference. This can be expressed in terms of rotor speeds as:

$$\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} k_F & k_F & k_F & k_F \\ -k_F L & k_F L & k_F L & -k_F L \\ -k_F L & k_F L & -k_F L & k_F L \\ -\sqrt{2}k_M & -\sqrt{2}k_M & \sqrt{2}k_M & \sqrt{2}k_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{4}$$

Now, we may further simplify our Newton–Euler force equation as well as angular acceleration equation as shown in Equations (5) and (6). $\mathbf{z}_W$ and $\mathbf{z}_B$ represent the z-axis of the world frame W and the body frame B, respectively.

$$m\ddot{\mathbf{r}} = -mg\,\mathbf{z}_W + \mathbf{R}_{WB}\,u_1\,\mathbf{z}_B \tag{5}$$

$$\dot{\omega}_B^W = \mathbf{I}^{-1} \left[-\omega_B^W \times \mathbf{I}\omega_B^W + \begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} \right] \tag{6}$$

The state vector of the quadrotor system is usually defined with position, velocity, Euler angles, and the angular velocity $[x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, p, q, r]^T$ for the purpose of vehicle control. We can show that the quadrotor is a flat system as its dynamics have flatness property [8]. Quadrotor dynamics with four inputs $[u_1, u_2, u_3, u_4]^T$ may be represented with four flat outputs $\sigma = [x, y, z, \psi]^T$, which can be used to explicitly express all states and inputs. In other words, the states and the inputs can be written as algebraic functions of four carefully selected flat outputs $\sigma = [x, y, z, \psi]^T$ and their derivatives. This facilitates the representation of the state vector and input vector of the under-actuated quadrotor with just four parameters. Therefore, the trajectory generation problem can be thought of as a generation of a considerably smooth (with reasonably bounded derivatives) four-element time-varying vector in the space of flat outputs as:

$$\sigma(t) : [t_0, t_m] \rightarrow \mathbb{R}^3 \times SO(2) \tag{7}$$

3. Path Planning

Path planning is a process of finding a feasible piece-wise linear path that consists of a series of waypoints in a robot's configuration space (C-Space) from the current configuration to the goal configuration. Different approaches are used for robotic path planning depending on the type of environment, planning goals, accuracy requirements, and timing requirements. Path planning algorithms may be broadly classified into Graph-Search-Based Algorithms (GSBA) and Sampling-Based Algorithms (SBA) [7]. GSBAs are resolution-optimal and resolution-complete but suffer from the curse of dimensionality. Therefore, GSBAs are very inefficient in large, high-dimensional settings. SBAs, on the other hand, are non-optimal, directionless, and only probabilistically complete algorithms. The main advantage of SBAs comes in their relatively efficient implementation in large, high-dimensional configuration spaces. Prominent graph-search-based planning algorithms include Dijkstra [25], A$*$ [26], Jump Point Search (JPS) [19], and State Lattice methods [20,27,28]. We also have more efficient variants of these prominent graph-search-based algorithms such as Dynamic A$*$ (D$*$) [29], Lifelong Planning A$*$ (LPA$*$) [30], D$*$ Lite [31], etc. Probabilistic Road Maps (PRM) and Rapid-exploring Random Trees (RRT) are two fundamental types of sampling-based path planning methods [32]. As in the case of graph-search-based algorithms, we have different variants of sampling-based methods as shown in Figure 1.

The quality and execution time of a planned path are highly dependent upon the choice of the path planning algorithm. We experimented with both GSBAs and SBAs in this study and applied these algorithms to different scenarios. In SBAs, we experimented with the 3D version of the asymptotically optimal RRT planner (RRT-Star) [33]. This method provides probabilistic completeness and we can apply this method to a map of a relatively larger

size. Though this method does not provide any guarantee concerning the time it takes to find a path, it is mostly comparable to other GSBA-based methods in moderately dense and relatively small environments. For GSBAs, we experimented with a 3D version of the Jump Point Search algorithm (JPS-3D) [18]. Although search-based algorithms such as Dijkstra and A$*$ are resolution complete, their computation time for finding an optimal path is a limitation when used with large maps. The 3D version of the Jump Point Search (JPS-3D) algorithm solves this problem to some extent by planning in uniform-cost grid maps. A graphical comparison of these two methods is presented in Figure 2 and the difference in the execution times of these two methods for varying scenarios can be seen in Table 1. Once we find a path through the path planning algorithm, we can apply our trajectory pipeline to generate optimal, kino-dynamic-feasible, and obstacle-free trajectories in cluttered indoor environments as shown in Figure 4.

Table 1. Execution times for different parts of our algorithm. Since the Path Relocation (PR) step is the unique feature of our algorithm, we test our system with PR and without PR.

Map Scenario	Map Size	Voxel Size	Execution Times (sec)						
			Path Planning		Path Relocation	Obstacle-Free Corridor		Trajectory Optimization	
	(m)	(cm)	JPS-3D	RRT$*$	PR	with PR	w/o PR	with PR	w/o PR
(a) Vertical Boxes	$3 \times 3 \times 3$	$1 \times 1 \times 1$	0.242581	2.257435	0.060654	0.244080	0.346878	0.549249	0.583510
(b) Vertical Columns	$3 \times 3 \times 3$	$2 \times 2 \times 2$	0.081071	2.236969	0.069489	0.590176	0.624557	0.571728	0.604901
(c) Multiple Floors	$6 \times 6 \times 10$	$1 \times 1 \times 1$	2.891760	6.154565	0.097741	1.691436	1.700918	0.457560	Failure [1]
(d) Multiple Rooms	$10 \times 10 \times 4$	$1 \times 1 \times 1$	1.141364	1.915849	0.079010	0.594684	0.718829	0.563237	Failure [1]
(e) Long Corridor	$10 \times 6 \times 2.5$	$1 \times 1 \times 1$	1.192246	2.445646	0.073292	1.355371	2.086071	0.469419	Failure [1]

[1] Solver stopped prematurely after reaching the maximum iteration limit of 2000 iterations. A probable cause for solver failure is the presence of at least one extra-small segment in the path close to the edge of the thin wall.

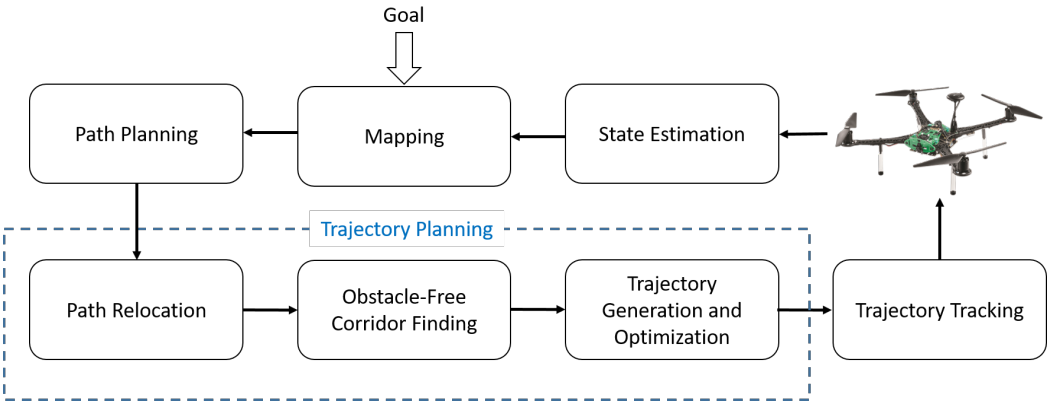

Figure 4. Block diagram for our system.

4. Trajectory Planning

Our trajectory planning pipeline is comprised of three steps as shown in Figure 4. In the first step of trajectory planning, namely path relocation, we relocate the path generated by the path planning algorithm such that we assist our next step of obstacle-free corridor generation and improve the optimization performance of the algorithm. In the second step, we try to find an obstacle-free corridor in the free space that can be used to plan a quadrotor trajectory. In the final step of the trajectory pipeline, we represent our trajectory with polynomials and optimize that trajectory within the constraints provided by the obstacle-free corridor (OFC). These three steps of the trajectory planning pipeline are explained below.

4.1. Path Relocation

We experimented with GSBAs and SBAs for path planning as explained in the previous section and found that paths planned through these algorithms usually pass in very close proximity to obstacles. This behavior is particularly dominant in graph-search-based techniques because GSBAs try to minimize path lengths. Though paths with minimum length are good for energy-efficient navigation, the close proximity of these paths to the obstacles makes convergence very difficult with respect to the trajectory optimization problem. We have observed in our simulations that convergence of the optimization problem takes considerably more iterations when applied to paths in close proximity to obstacles. Sometimes, the optimization solver may even fail to produce a feasible result because the constraints for the optimization solver are very tight in close proximity to obstacles. Therefore, we have a situation where we need a compromise because if we try to generate shorter paths for energy-efficient navigation, the optimization solver is less likely to converge. Conversely, if we try to move the path away from the obstacle, we generate long trajectories that may not be energy-efficient.

The second problem associated with path planning algorithms is the presence of very small segments in a generated path (segments smaller than 0.25 m in length). This phenomenon is particularly dominant in graph-search-based algorithms near thin obstacles because these algorithms try to minimize the path length as shown in Figure 2. These small segments might also be present in SBAs because of their random nature. These extra-small segments impose very tight constraints on the optimization problem in challenging robotic environments. We observed in our experiments that even one segment of a small length in a path usually results in solver failure. Finally, we sometimes experience the problem of locked segments in generated paths. A locked segment is one that is locked among obstacles such that it gives very little freedom to rotate or relocate this segment within free space. One such segment is shown in Figure 5a. This problem particularly arises when we have extra long segments in our generated path (segments greater than 10 m in length). Locked segments also tighten the constraints and make the optimization problem difficult to solve. With these observations, it is not suitable to use path planning results directly for trajectory generation and optimization.

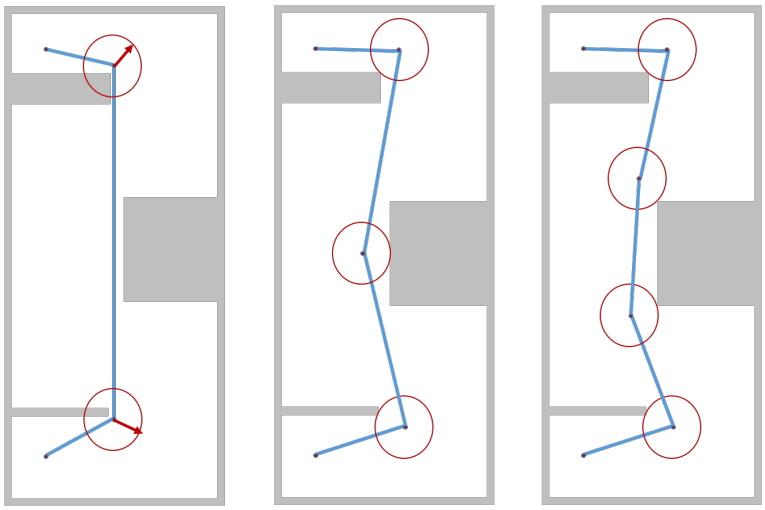

Figure 5. Left (**a**): Path planned with the planning algorithm comprised of three segments with its middle segment locked among obstacles. Waypoints of the original path have been superimposed with margin spheres shown in red. **Center** (**b**): Middle segment of the planned path in (**a**) is bisected into two sub-segments and the new waypoint is defined at the cut point. Now we have more freedom to relocate the path. **Right** (**c**): Middle segment of the planned path in (**a**) is divided into three sub-segments and new waypoints are defined at cut points. We have even more freedom to relocate the path in (**c**).

We came up with a novel solution to handle these problems and propose a path relocation algorithm that relocates path segments such that they are sufficiently far from obstacles but not far enough to affect the overall lengths of paths. Moreover, it solves the problem of extra-small segments as well as locked segments. It is pertinent to mention that Zhau et al. [20] also tried to solve the problem associated with the close proximity to generated paths to obstacles by using the collision cost term in the objective function of the optimization problem. The collision cost term in the optimization problem is formulated as a repulsive force of obstacles acting on each control point using Euclidian Distance Fields (EDF). This way of solving the close proximity problem led to the non-quadratic formulation of the objective function which is not that efficient to compute. Secondly, it may also take the original path far away from obstacles and considerably increase the overall length of the path. Our method tries to solve this problem along with the other two problems presented above without any compromise concerning the quadratic formulation of the problem and does not let the path be relocated far beyond the limits. We have not come across any work that solves the problem of extra-small segments as well as locked segments. Our path relocation algorithm is described below.

4.1.1. Define Margin Spheres

In the first step of path relocation, we define multiple margin spheres along the length of the path generated by path planning algorithms. Based upon our observation of paths generated via the planning algorithm, we make sure every junction between segments (waypoints) has a margin sphere defined on it as shown in Figure 6. We chose to define margin spheres on every waypoint because waypoints lie in close proximity to obstacles in graph-search-based planning algorithms. We choose the dimensions of spheres considering the size of free space openings (such as window/door opening sizes) in our environment. Our work considers a diameter of 1.5 m for margin spheres because we assume that our environment has a minimum opening of 1.5 m $\times$ 1.5 m between the starting point and the target point in the robotic configuration space. By doing so, we try to have a minimum distance of 0.75 m between our waypoints and obstacles.

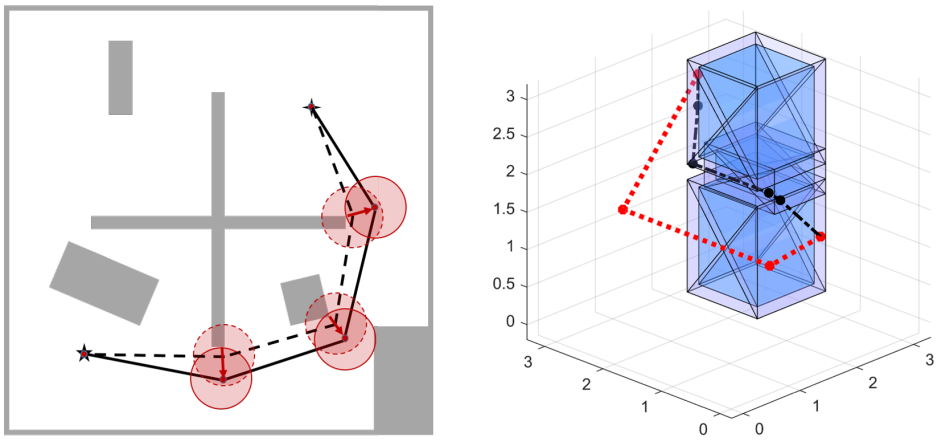

Figure 6. Left: The Path Relocation (PR) concept in a 2D map, where obstacle space is represented with gray area. The black dotted line shows the original path planned by the planning algorithm and the black solid line shows the relocated path. Margin spheres along with relocation direction arrow are shown in red color. **Right**: Path relocation results in 3D map, where obstacle space is represented with blue cubes. The black dotted line show original path planned by the planning alogrithm. Red dotted line shows relocated path produced by Path Relocation (PR) step. Note the removal of extra-small segments and increase in the spacing between relocated path and obstacles.

4.1.2. Margin Spheres Update

Once we define margin spheres at the waypoints of our path, the next step is to relocate those margin spheres to a new location such that we minimize the overlap of margin spheres with the obstacles. Assuming we have an onboard sensor that provides us with point-cloud data of obstacles around that sphere, we select the points inside the boundary of the margin sphere to take further steps. In this step, we relocate our margin spheres based on the number and distribution of point-cloud data inside the boundaries of margin spheres. To relocate the margin sphere to a feasible point, we need to find the direction of relocation and the step size of relocation. For the direction of relocation, we find the average of the k-nearest point inside the boundary of the sphere, $s_{k,avg}$, as a guide. Our direction of relocation for the jth sphere is from $s_{k,avg}$ towards the center of the sphere s_c. The step size of relocation is a tuneable parameter and we can vary its step length depending on the obstacle density of the map, and the required timing performance of the algorithm. The primary purpose of the margin sphere relocation is to gradually move a margin sphere away from obstacles such that its overlap with obstacles decreases in every iteration. We stop the margin sphere relocation at a point where we may not reduce the obstacle overlap any further. If we keep our step size very small, we may need numerous iterations to assure that we have the minimum possible overlap of relocated margin spheres with obstacles. Therefore, decreasing step size to extra-small lengths may negatively affect the computational efficiency of the algorithm. On the other hand, if we make our step size very large, the accuracy of our algorithm suffers. If we increase the step size beyond the diameter of the margin sphere, we may jump from one obstacle to a totally different obstacle and our algorithm may produce totally inaccurate results. Our work uses a step size of 0.15 m, which is ten percent of the margin sphere diameter. Once we know the direction and step size of the relocation for each margin sphere, we may relocate margin spheres to new locations in the environment. At the end of this step, the center points of our margin spheres do not necessarily lie exactly over the planned path waypoints.

4.1.3. Path Update

Once we relocate our margin spheres, we need to check if we can update our path waypoints with the center points of relocated margin spheres or not. To guarantee the feasibility of relocated margin spheres as new waypoints, we must check if the new configuration of path segments formed by the center points of relocated margin spheres is free from obstacle collisions or not. If the result is true, we may safely update segments and waypoints in the path. We may repeat the margin sphere update step and the path update step until the number of point-cloud data in the boundary of the margin spheres is decreasing as shown in the while-loop of the pseudocode shown in Algorithm 1. Once we exit the while-loop in Algorithm 1, we have a modified path that is sufficiently far away from the obstacles but not far enough to considerably affect the overall length of the planned path. Spacing of the modified path from obstacles depends upon the radius of margin spheres as well as the volume of free spaces in the map around the initial path planned with the path planning algorithm.

Algorithm 1 Given planned path $P = \langle p^0, p^1, \ldots, p^N \rangle$, find modified path $S = \left\langle p^0, s^1, s^2, \ldots, s^k, p^N \right\rangle$ for better optimization performance. The set of obstacle points is denoted by O.

function $PATH\_RELOCATION(P, O)$
for $iteration = 1, 2, \ldots (N-1)$ **do**
  $s^j(R, d) \leftarrow DefineWaypointSpheres(p^j)$
  $O_{inside} \leftarrow CalculatePointsInside(s^j, O)$
  while $\Delta O_{inside} > 0$ **do**
    $s^j_{temp} \leftarrow RelocateSpheres(s^j, O_{inside})$
    $s^j_{flag} \leftarrow CheckCollisions(P, s^j_{temp})$
    if $s^j_{flag} = False$ **then**
      $s^j \leftarrow s^j_{temp}$
    end if
    $O_{new} \leftarrow CalculatePointsInside(s^j, O)$
    $\Delta O_{inside} \leftarrow CalculateDeltaO(O_{new}, O_{inside})$
    $O_{inside} \leftarrow O_{new}$
  end while
end for.
$RemoveExtraSmallSegments(s^j)$
return $S = \left\langle p^0, s^1, s^2, \ldots, s^k, p^N \right\rangle$
end function

4.1.4. Extra-Small Segment Removal

An important step of the path relocation algorithm is the removal of extra-small segments from the path because these extra-small segments also impose tight constraints for the optimization solver. We use a very intuitive strategy in removing extra-small segments from the modified path. For every segment in the path, we check if its length is smaller than a certain threshold. If it is, we simply remove that segment from our path and join the surrounding segments together. Though we check for path collision with an obstacle after removing extra-small segments, we observe that the removal of extra-small segments from the path does not affect the feasibility of the path in most cases. We present the concept of path relocation along with results in simulation in Figure 6. We may note that the path relocation step changes the paths in two ways. First, it relocates the paths sufficiently far from the obstacles. Second, it removes extra-small segments from the path. These two effects can be observed in Figure 6 (Right). The black piece-wise dotted line segments show the original path planned by the planning algorithm. We can see that the original path has five segments and two of those segments are extra-small. Moreover, the original path is planned in very close proximity to the obstacles. The modified path (shown as a red piece-wise dotted line) solves both of these problems by moving the path farther from the obstacles and handling extra-small segments. The modified path has only three segments as compared to the five segments in the originally planned path.

4.1.5. Handling Locked Segments

Our path relocation algorithm is comprised of four steps: (a) define margin spheres, (b) margin sphere update, (c) path update, and (d) extra-small segment removal. These four steps of the path relocation algorithm solve only two of the three problems associated with the original path planned with the path planning algorithm. Our original path planning algorithm is designed to solve the problem of close proximity to the planned path to obstacles as well as the problem of extra-small segments. However, these steps do not solve the problem of locked segments. It is pertinent to mention that we cannot know in advance if our generated path has a locked segment or not. We can only find the locked segment by comparing the relocated path with the original path once the path relocation algorithm is complete. Therefore, we might need to run the path relocation algorithm multiple times

while resolving the locked segments before each iteration. We may resolve the locked segments by dividing every locked segment into multiple sub-segments and defining new waypoints at the cut points as shown in Figure 5. By doing so, we give more freedom to sub-segments to relocate to feasible locations as compared to the original single locked segment. Different approaches may be used to select the location of cut points at the locked segment. A simple and intuitive approach is to cut a long locked segment into two or three sub-segments of equal length and run the path relocation algorithm once again. However, a more robust approach is to find one or more cut points in close proximity to locking obstacles by using Euclidean Distance Fields (EDF) as used by Zhou et al. [20] to compute the collision cost term in the objective function. Once we resolve the locked segment problem in our path by multiple iterations of Algorithm 1, we may use this modified path to define better corridor constraints for the optimization problem as described in the next section.

4.2. Obstacle-Free Corridor Construction

The second step of the trajectory planning pipeline is the Obstacle-Free Corridor (OFC) finding which we adopted from the work of Liu et al. [18]. In this step, we construct a polyhedron around every segment of the piece-wise linear modified path such that those polyhedra define a free space in which we may generate the optimal trajectory for quadrotors to follow. Every polyhedron is defined with multiple hyperplanes, which represent different faces of polyhedra. Those hyperplanes may be used as inequality constraints in our optimization problem. The process of finding hyperplanes representing polyhedra of obstacle-free corridors is comprised of three steps, namely (1) find ellipsoid, (2) find polyhedron, and (3) cuboidal bonding. The cuboidal bonding step in our approach is slightly different from [18] in the sense that we use the size of free space openings (e.g., door openings in walls) in our environment to decide the dimensions of the cuboid instead of the maximum speed and the maximum acceleration of quadrotors. We assume that the free space openings are wider than the quadrotor size and the quadrotor may easily pass through free space openings. The reason for this change in approach is to make the corridor finding step independent of quadrotor types.

4.2.1. Find Ellipsoid

In this step, we try to find the empty space surrounding the path segments with the help of ellipsoids. Our goal is to find an ellipsoid with its major axis aligned with the path segment. The other semi-axes of ellipsoids are such that they maximize the volume contained by ellipsoids while avoiding all obstacles in the environment. An ellipsoid in $\mathbb{R}^3$ can be represented with a matrix E and a vector d where E is a 3×3 symmetric positive definite matrix representing the deformation of the sphere and d represents the center of the ellipsoid.

$$\xi(E, d) = \{p = E\bar{p} + d \mid \|\bar{p}\| \leqslant 1\} \tag{8}$$

To find a feasible ellipsoid, we need point cloud data representing obstacles in the vicinity of the path segment and a line representing the path segment itself. We first start with a spheroid around the line segment as shown in Figure 7a. Now considering the line segment as the semi-major axis of the ellipsoid, we shrink the initial spheroid to obtain a maximal spheroid (a spheroid with two axes of equal length) such that it does not contain any point-cloud data representing obstacles within its boundaries. We shrink the ellipsoid step by step by searching for the closest obstacle point from the ellipsoid center. At the end of the shrinking process, we have a maximal spheroid with one closest obstacle point lying at its boundary. This process is depicted in Figure 7. With the closest point at the boundary of the maximal ellipsoid and a line segment, we form the xy-plane of the ellipsoid. Later we deform the maximal ellipsoid further and dilate it along the z-axis. We find the right length of the third axis in the z-axis direction by a similar procedure by first expanding the ellipsoid along the z-axis and then shrinking step by step until we have no obstacle point

inside the boundary of the ellipsoid as explained in Algorithm 2. Multiple segments of the obstacle-free path along with the corresponding ellipsoid are also depicted in Figure 8d.

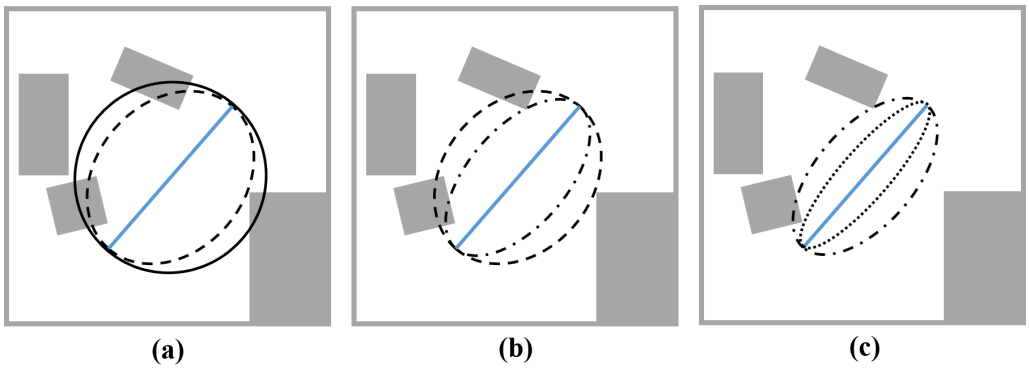

Figure 7. Ellipsoid shrinking process shown in 2D, where grey area represents obstacle space of environment map. The blue solid line segment represents a single segment of the planned path. **Left** (**a**): We start with a black solid-line ellipsoid shown in (**a**). We find a point in the obstacle space that is closest to the center of the blue path segment to generate the black dashed-line ellipsoid. This completes the first iteration of ellipsoidal shrinking. **Center** (**b**): Now, we take the black dashed-line ellipsoid generated in (**a**) as a reference and repeat another iteration of the shrinking process to generate a black dashed-dotted-line ellipsoid. **Right** (**c**): Again, we take the black dashed-dotted-line ellipsoid generated in (**b**) as a reference and repeat another iteration of the ellipsoid shrinking process to generate the black dotted-line ellipsoid shown in (**c**). After this iteration, we may define xy-plane of ellipsoid.

Algorithm 2 Given line segment S^0, find the ellipsoid $\xi^0(E,d)$. The set of obstacle points is denoted by O.

function $Find\_Ellipsoid(S^0, O)$
$\xi^0(E,d) \leftarrow DefineInitialEllipsoid(S^0)$
$O_{inside} \leftarrow RemovePointsOutside(\xi^0, O)$
$j \leftarrow 0$
while $O_{inside} \neq Empty$ **do**
  $p_j^e \leftarrow ClosestPoint(\xi^0, O_{inside})$
  $\xi^0(E,d) \leftarrow ShrinkEllipsoidAlongYZ(\xi^0, p_j^e)$
  $O_{inside} \leftarrow RemovePointsOutside(\xi^0, O_{inside})$
  $j \leftarrow j+1$
end while
$n_{xy} \leftarrow FindXYPlane(\xi^0, p_j^e)$
$\xi^0(E,d) \leftarrow DilateEllipsoidZ(\xi^0, n_{xy})$
$O_{inside} \leftarrow RemovePointsOutside(\xi^0, O)$
$k \leftarrow 0$
while $O_{inside} \neq Empty$ **do**
  $p_k^e \leftarrow ClosestPoint(\xi^0, O_{inside})$
  $\xi^0(E,d) \leftarrow ShrinkEllipsoidAlongZ(\xi^0, p_k^e)$
  $O_{inside} \leftarrow RemoviePointsOutside(\xi^0, O_{inside})$
  $k \leftarrow k+1$
end while
return $\xi^0(E,d)$
end function

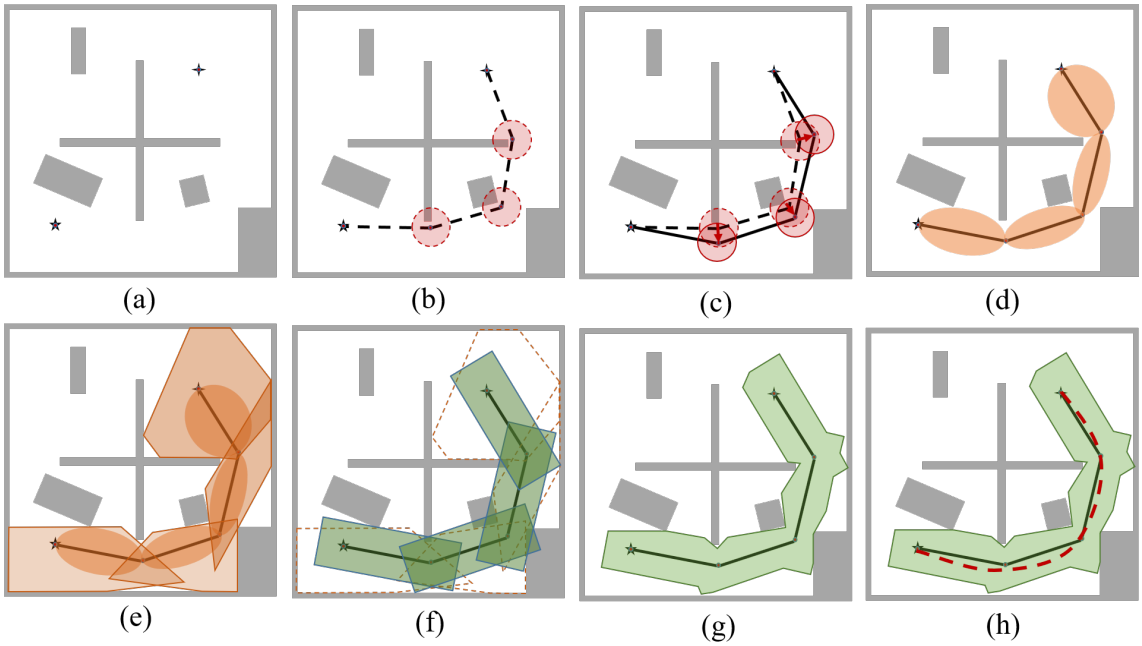

Figure 8. Robotic motion planning process in sequence in 2D. (**a**) Map of the environment representing obstacles (grey areas), starting location (four-point star), and goal location (five-point star). (**b**) Initial path planned with planning algorithm (dotted line segments). Margin spheres are represented in red at waypoints. (**c**) The modified path after the application of the path relocation procedure (solid line segments). (**d**) Path segments with obstacle-free ellipsoids superimposed on corresponding segments. (**e**) Representation of polyhedra encapsulating corresponding ellipsoids (in orange color) on every path segment. (**f**) Cuboidal bonding of polyhedra (in green) to find tunnel-like pathway. (**g**) Intersection of polyhedra and cuboids to find obstacle-free corridor (OFC) represented in green. (**h**) Obstacle-free corridor along with optimized trajectory represented in red with dashed lines.

4.2.2. Find Polyhedron

In this step, we start with the ellipsoid derived in the previous section corresponding to each segment and find an obstacle-free polyhedron surrounding every segment of the path. As in the case of the ellipsoid in the previous section, the space inside the boundary of the polyhedron must be free of all obstacle points. To derive this polyhedron, we slowly dilate the ellipsoid found in the previous step to define a halfspace every time we find an obstacle overlapping the ellipsoid and continue the process. All such halfspaces provide us with a polyhedron free from obstacles. Assume we have an ellipsoid $\xi^0(E, d)$ that we are slowing dilating until we find a point p_0^c at the boundary of the ellipsoid. The tangent plane to the ellipsoid at the point can be defined as a halfspace $H_0 = \{p | a_0^T p < b_0\}$ containing the ellipsoid. After computing the halfspace, we can remove all the obstacle points O_{remain} away from that halfspace. Later, we can again dilate the ellipsoid slowly until we obtain another obstacle point and repeat the process. Continuous repetition of the process produces a set of halfspaces that collectively form a polyhedron as shown in Figure 9 in a 2D setting. This polyhedron contains the segment and the initial obstacle-free ellipsoid inside it. Algorithm 3 shows the pseudo-code for this process. The hyperplane defining the jth halfspace H_j is tangent to the ellipsoid ξ^j at point p_j^c and is computed as:

$$a_j = \left.\frac{d\xi^j}{dp}\right|_{p=p_j^c} = 2E^{-1}E^T(p_j^c - d) \tag{9}$$

$$b_j = a_j^T p_j^c \tag{10}$$

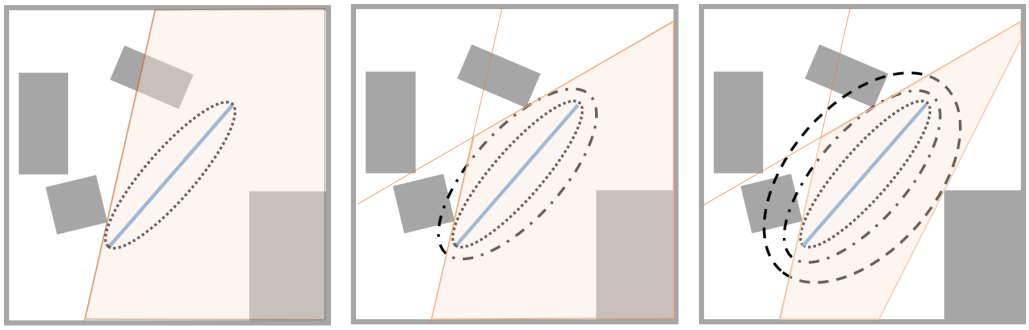

Figure 9. Ellipsoidal dilation and polyhedron finding process. Orange-colored solid-lines represent the hyperplanes that we find in each iteration of the ellipsoidal dilation process. Light-shaded orange areas represent the halfspaces defined by these hyperplanes. **Left** (**a**): We start the ellipsoidal dilation process with the ellipsoid represented with dotted-line and find the first hyperplane as shown in (**a**). **Center** (**b**): We dilate further to find the second hyperplane, which is tangent to the ellipsoid represented with dashed-dotted-line and shown in (**b**). **Right** (**c**): Again, we dilate once last time to find the third hyperplane, which is tangent to the third ellipsoid (represented with dashed-line) and shown in (**c**). By finding the intersection of these halfspaces (defined by corresponding hyperplanes), we may form a polyhedron for the corresponding segment.

Algorithm 3 Given $\xi^0(E,d)$, find the polyhedron $C(A,b)$. The set of obstacle points is denoted by O.

function $FIND\_POLYHEDRON(\xi^0, O)$
$O_{remains} \leftarrow O$
$j \leftarrow 0$
while $O_{remains} \neq Empty$ **do**
  $p_j^c \leftarrow ClosestPoint(\xi^0, O_{remains})$
  $\xi^j \leftarrow DilateEllipsoid(\xi^0, p_j^c)$
  $\langle \mathbf{a_j}, b_j \rangle \leftarrow FindHalfPlane(\xi^j, p_j^c)$
  $j \leftarrow j + 1$
end while
$C : \mathbf{A} \leftarrow [\mathbf{a_0}, \mathbf{a_1}, \dots], \ \mathbf{b} \leftarrow [\mathbf{b_0}, \mathbf{b_1}, \dots]$
return $C(\mathbf{A}, \mathbf{b})$
end function

4.2.3. Cuboidal Bonding

Since polyhedrons generated through the last part contain the ellipsoid inside them and the ellipsoids themselves contain the corresponding line segment, we are guaranteed to have our planned path inside the union of all polyhedrons. Moreover, polyhedrons corresponding to adjacent segments are always guaranteed to overlap to some extent with each other because the ellipsoids contained by these polyhedrons are connected with each other. Thus, technically speaking, we have an obstacle-free space containing the whole planned path inside it and we can plan our trajectory inside this obstacle-free space. However, this obstacle-free space is not that efficient yet because the trajectory generated inside this obstacle-free space may go too far away from the original planned path. To keep the trajectory close to the original planned path, we use the concept of cuboidal bonding. We enclose every segment of the planned path inside a cuboid and take the intersection of the cuboid with the polyhedron as shown in Figure 8f,g. By doing so, we further restrict the obstacle-free corridor such that it provides a free space like a tunnel surrounding the modified path generated through the path relocation step. The maximum width of the obstacle-free tunnel-like corridor is dependent upon the dimensions of the cuboid we use. Wider the cuboid we use, wider the tunnel-like obstacle-free space will be. We choose the

dimensions of the cuboid considering the size of free space openings (e.g., door openings in walls) in our environment. This tunnel-like pathway is defined with a set of hyperplanes and is called the obstacle-free corridor (OFC). We can use this obstacle-free corridor as inequality constraints while optimizing our trajectory with respect to our objective criterion as explained in the next section. Before we jump to the next section, it is important to emphasize the importance of the path relocation step in our trajectory planning pipeline. The relocation step in our pipeline helps to generate paths sufficiently far from obstacles with the least possible segments. The further the paths are from the obstacles, the easier it will be to generate wider tunnel-like obstacle-free corridors surrounding the path. The wider the obstacle-free corridors surrounding the path, the easier the constraints will be for the optimization solver.

4.3. Trajectory Generation and Optimization

In this section, we introduce the general setup of trajectory generation and optimization. Given the map of an environment and starting/goal position of the robot, we first planned the path, which is comprised of multiple piece-wise linear segments connected with each other by waypoints. We also improved the planned paths by modifying them for better optimization performance in the path relocation stage of the trajectory generation pipeline. Later, we defined obstacle-free corridors (OFCs) from the starting position to the ending position. Now we want to have a trajectory that our robot may track from the start to the goal position through these corridors, without colliding with obstacles. First, we represent our trajectory with piece-wise polynomial basis functions as shown in Equation (11) where $\sigma_m(t)$ represents the trajectory corresponding to the mth segment of the path represented by the nth order polynomial basis function.

$$\sigma(t) = \begin{cases} \sigma_1(t) = \sum_{i=0}^{n} \sigma_{1i} t^i & t_0 \leq t < t_1 \\ \sigma_2(t) = \sum_{i=0}^{n} \sigma_{2i} t^i & t_1 \leq t < t_2 \\ \vdots & \\ \sigma_m(t) = \sum_{i=0}^{n} \sigma_{mi} t^i & t_{m-1} \leq t < t_m \end{cases} \tag{11}$$

We know that our robot is a dynamical system and it cannot follow arbitrary trajectories. Therefore, we also need our trajectories to be smooth enough so they can be tracked by our dynamical system. The smoothness of trajectories translates to minimizing the rate of change of trajectories. Since a quadrotor is a fourth-order dynamical system, we should be able to control the fourth derivative of the trajectory with the quadrotor. This means that we need to have trajectories that are differentiable to the fourth order and their fourth-order derivative should be as low as possible. Therefore, we formulate our trajectory generation problem as an optimization problem where our objective function is the fourth derivative of the trajectory (snap) and we constrain our trajectory within the obstacle-free corridors found in the previous section.

$$\begin{aligned} \arg\min_{\sigma} J &= \sum_{i=0}^{m-1} \int_0^{\Delta t_i} \left\| \frac{d^4}{dt^4} \sigma_i(t) \right\|^2 dt \\ s.t.\ \mathbf{A}_{eq}^T \sigma_i(t) &= \mathbf{b}_{eq} \\ \mathbf{A}_{i-eq}^T \sigma_i(t) &< \mathbf{b}_{i-eq} \end{aligned} \tag{12}$$

Here the equality constraints represented by matrix A_{eq}^T and vector b_{eq} correspond to the lower order derivatives of the trajectory at waypoints (junction points between independent segments of the planned path) [34]. To ensure the smoothness of the trajectory at waypoints, derivatives up to the order of three (zeroth, first, second, and third derivatives)

of the trajectory at the end of any segment must be equal to the corresponding derivatives at the start of the next connecting segment as in Equation (13). Moreover, inequality constraints represented by matrix A^T_{i-eq} and vector b_{i-eq} correspond to the half-planes representing obstacle-free corridors (OFCs).

$$\frac{d^k}{dt^k}\sigma_i(t_i) = \frac{d^k}{dt^k}\sigma_{i+1}(t_i), \;\; k = 0 \ldots 3 \tag{13}$$

We can formulate the optimization problem in Equation (12) as a Quadratic Program (QP) by considering coefficients of piece-wise polynomial functions $c_\sigma = [c_{\sigma_1}, \cdots, c_{\sigma_m}]$ as decision variables as formulated in Equation (14). Here, c_{σ_m} represents the vector of polynomial coefficients corresponding to the mth segment. The construction of the Hessian matrices Q_{σ_m} is omitted for brevity but follows from the differentiation of the square of the mth polynomial with respect to each element of its coefficient vector c_{σ_m} [35]. We can solve this quadratic programming problem efficiently with traditional convex optimization solvers provided in commercial software packages as in the MATLAB optimization toolbox.

$$\begin{aligned} \arg\min_{c_\sigma} J = & \begin{bmatrix} c_{\sigma_1} \\ \vdots \\ c_{\sigma_m} \end{bmatrix}^T \begin{bmatrix} Q_{\sigma_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & Q_{\sigma_m} \end{bmatrix} \begin{bmatrix} c_{\sigma_1} \\ \vdots \\ c_{\sigma_m} \end{bmatrix} \\ s.t. \; & \mathbf{A}^T_{eq} c_\sigma = \mathbf{b}_{eq} \\ & \mathbf{A}^T_{i-eq} c_\sigma < \mathbf{b}_{i-eq} \end{aligned} \tag{14}$$

Another important element in this approach to solve the optimization problem for trajectory generation is the effect of time corresponding to each segment of the planned path. Changing the time $\Delta t_m = t_m - t_{m-1}$ corresponding to a segment not only changes the velocity of the robot at that segment but also the path of the trajectory. We simulated the waypoint trajectory generation optimization problem in [9] and found results depicted in Figure 10 by varying the time allocations for the single segment in the piece-wise linear path while keeping other segment times constant. Our simulation contains three segments and we vary the segment times of the middle segment. We can see considerable trajectory variation in Figure 10. It is important to mention that our approach of obstacle-free corridors may considerably avoid the obstacle overlapping problem in Figure 10 but time allocation is still important because of finite thrust forces in robotic vehicles. We used the trapezoidal velocity profile method to allocate time to each segment while taking into consideration the map of our environment and the robotic thrust thresholds. We use static starting and goal states with zero velocity, acceleration, and jerk to ensure safety. The vehicle is supposed to accelerate at the start position and decelerate while reaching the goal position.

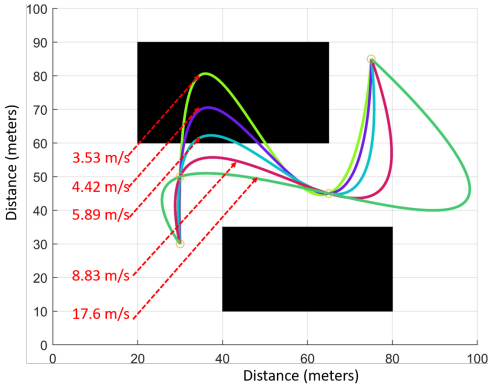

Figure 10. Trajectory optimization problem solved with the same objective and constraints but with different segment times for the middle segment. Corresponding average vehicle velocities during the segment passage are shown in the figure.

5. Trajectory Tracking and Control

We now present a geometric tracking control of a quadrotor to follow specified trajectories [36]. We assume that we are provided with a time-varying reference trajectory in terms of flat outputs as $\sigma_T(t) = [\mathbf{r}_T(t)^T, \psi_T(t)]^T$, where $\mathbf{r}_T(t) = [x(t), y(t), z(t)]^T$ is the desired position as a function of time in the world frame of reference and $\psi_T(t)$ is the desired yaw profile. We also assume that this time-varying reference trajectory is smooth enough to be tracked via a nonlinear controller. First, we define position error as $\mathbf{e}_p = \mathbf{r}_T(\mathbf{t}) - \mathbf{r}$ and velocity error as $\mathbf{e}_v = \dot{\mathbf{r}}_T(t) - \dot{\mathbf{r}}$. If we want our robot to follow a desired position profile $\mathbf{r}_T(t)$, we need these errors $\mathbf{e}_p$ and $\mathbf{e}_v$ to exponentially go to zero as.

$$(\ddot{r}_T(t) - \ddot{r}_c) + K_d e_v + K_p e_p = 0 \tag{15}$$

Here, $\ddot{\mathbf{r}}_c$ is the commanded acceleration required such that the position and velocity errors may exponentially go to zero. We may also calculate the desired force $\mathbf{F}_{des}$ required by the controller by multiplying Equation (15) with the mass of the vehicle and considering acceleration applied to the vehicle by force due to gravity.

$$\mathbf{F}_{des} = K_p \mathbf{e}_p + K_d \mathbf{e}_v + mgz_W + m\ddot{r}_T(t) \tag{16}$$

where K_p and K_d are positive definite gain matrices. Since quadrotor propellers are fixed to the body of the quadrotors and may only apply forces along the z-axis of the body frame of reference, we need to find the projection of the desired force vector onto the body frame's z-axis in order to compute the first element of the control input vector as $u_1 = \mathbf{F}_{des} \cdot \mathbf{z}_B$.

To determine the other three control input elements $[u_2, u_3, u_4]^T$ as shown in Equation (4), we must consider the attitude tracking error, $\mathbf{e}_R$ and angular velocity error, $\mathbf{e}_\Omega$. We may give similar treatment to these errors as we did to position and velocity errors. We assume that we know the current orientation, R_{WB}, of the quadrotor in the world frame and we need to know the desired orientation, R_{des}, of the quadrotor in the world frame. Once we know R_{des}, we may compare it with R_{WB} to find the attitude tracking error $\mathbf{e}_R$ and the angular velocity error $\mathbf{e}_\Omega$. R_{des} is comprised of three unit vectors $R_{des} = [\mathbf{x}_{B,des}, \mathbf{y}_{B,des}, \mathbf{z}_{B,des}]$. First, we observe that the $\mathbf{z}_{B,des}$ direction should be along the desired thrust vector to maximize the effect of the force applied by propellers as:

$$\mathbf{z}_{B,des} = \frac{F_{des}}{\|F_{des}\|} \tag{17}$$

Knowing the specified yaw angle along the trajectory, $\psi_T(t)$, we may compute $\mathbf{x}_{B,des}$ and $\mathbf{y}_{B,des}$ as:

$$\mathbf{x}_{mid} = [\cos\psi_T \tau, \sin\psi_T, 0]^T \tag{18}$$

$$\mathbf{y}_{B,des} = \frac{z_{B,des} \times x_{mid}}{\|z_{B,des} \times x_{mid}\|} \tag{19}$$

$$\mathbf{x}_{B,des} = \mathbf{y}_{B,des} \times \mathbf{z}_{B,des} \tag{20}$$

provided $\mathbf{z}_{B,des} \times \mathbf{x}_{mid} \neq 0$. This defines the desired rotation matrix R_{des}. Next we define the attitude tracking error, $\mathbf{e}_R$, and angular velocity error, $\mathbf{e}_\Omega$ as:

$$\mathbf{e}_R = \frac{1}{2}\left(R_{des}^T R_{WB} - R_{WB}^T R_{des}\right)^\vee \tag{21}$$

$$\mathbf{e}_\Omega = \Omega - R_{WB}^T R_{des} \Omega_{des} \tag{22}$$

where $\vee$ represents the vee map which takes elements of $so(3)$ to $\mathbb{R}^3$, and Ω is the angular velocity of the quadrotor. Now the desired moments $[u_1, u_2, u_3]$ required by the controller

such that the attitude tracking and angular velocity error exponentially go to zero are computed as:

$$[u_2, u_3, u_4]^T = -K_R \mathbf{e}_R - K_\Omega \mathbf{e}_\Omega \tag{23}$$

where K_R and K_Ω are diagonal gain matrices. This allows unique gains to be used for roll, pitch, and yaw angle tracking. Finally, we compute the desired rotor speeds to achieve the desired control vector $[u_1, u_2, u_3, u_4]^T$ by using Equation (4). When controlling position and attitude together, we use a cascade control loop where attitude is controlled in the inner loop while the position is controlled in the outer loop as shown in Figure 11.

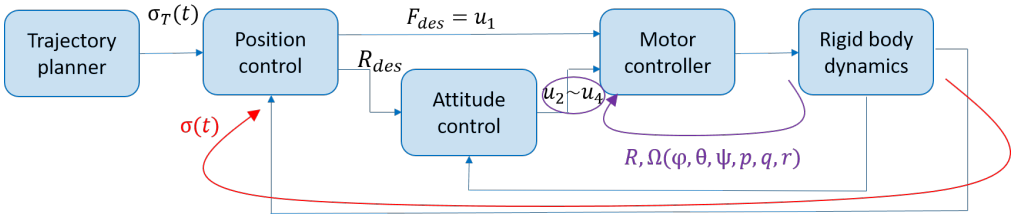

Figure 11. Cascade control loops for position and attitude control of quadrotor. Inner loop takes commanded orientation, $R_{des} = [\phi_{des}, \theta_{des}, \psi_{des}]$, as input and current attitude with angular velocities R, Ω from feedback to provide control signal, $[u_2, u_3, u_4]$, for attitude tracking. Outer loop takes desired trajectory, $\sigma_T(t)$, as input and current trajectory $\sigma(t)$ from feedback to provide desired orientation, R_{des}, and desired force $F_{des} = u_1$

6. Results and Discussion

We use five different scenarios to test the run time of our algorithm by generating trajectories through them. These five scenarios are shown in Figure 12 and named (a) vertical boxes, (b) vertical columns, (c) multiple stories, (d) multiple rooms, and (e) long corridor. These scenarios vary in terms of volume, obstacle density, and obstacle size. These maps may be considered as typical for different environments in real-life scenarios. To evaluate the computational expense of our algorithm, we split the complete process into five parts: path planning, path relocation (PR), obstacle-free corridor generation, trajectory optimization, and trajectory tracking. The time cost for the first four components while generating trajectories is shown in Table 1. Results for the trajectory tracking are presented in Table 2. For path planning, we compared both GSBA and SBA methods. While planning our path using grid-based maps, we used JPS-3D for GSBA and RRT$*$ for SBA. We simulated our concept on MATLAB environment using Intel(R) Core(TM) i7-9700KF CPU running at 3.60 GHz with an installed memory of 32 GB. As can be seen from the results, we are able to generate trajectory through multiple indoor scenarios in under a few hundred milliseconds.

We can see from Table 1 that the volume of our environment varies from 9 cubic meters to 400 cubic meters. The first thing we may infer from the results concerns the planning times using the sampling-based algorithm called RRT$*$. We can see that the planning time while using RRT$*$ has no relevance to the size of the map. Instead, it has some relevance to the obstacle density and the distribution of those obstacles in the environment. RRT$*$ path planning through 'Multiple Floors' takes maximum time because our starting and ending locations are blocked by two obstacles and there are very small openings to reach the goal position. Path planning through graph-search-based methods somewhat depends upon the size of the map and the distribution configuration of the obstacle inside that map. Therefore, it is better to use the sampling-based method for relatively bigger maps and search-based methods for small maps. Another thing to notice is the importance of the path relocation step before we can find obstacle-free corridors and optimize our generated trajectory. It is pertinent to mention that we can skip the path relocation step altogether and perform the obstacle-free corridor and trajectory optimization steps. However, as is evident from the results in Table 1, the path relocation step provides two benefits. First, it reduces the time required to find obstacle-free corridors as well as trajectory optimization. Second, it

improves the chances of obtaining a solution to the trajectory optimization problem. Safety corridors produced after the path relocation step are farther from the obstacles and may be relatively wider. While solving the trajectory optimization problem in the Multiple Rooms scenario, our solver failed to provide a solution when we skipped the path relocation step in our pipeline. On the other hand, it provided a good solution while solving the problem with the path relocation step included. After we generate the trajectory, the last step is to track the trajectory with the control algorithm. Trajectory tracking performance is tested with the full trajectory generation pipeline and is listed in Table 2.

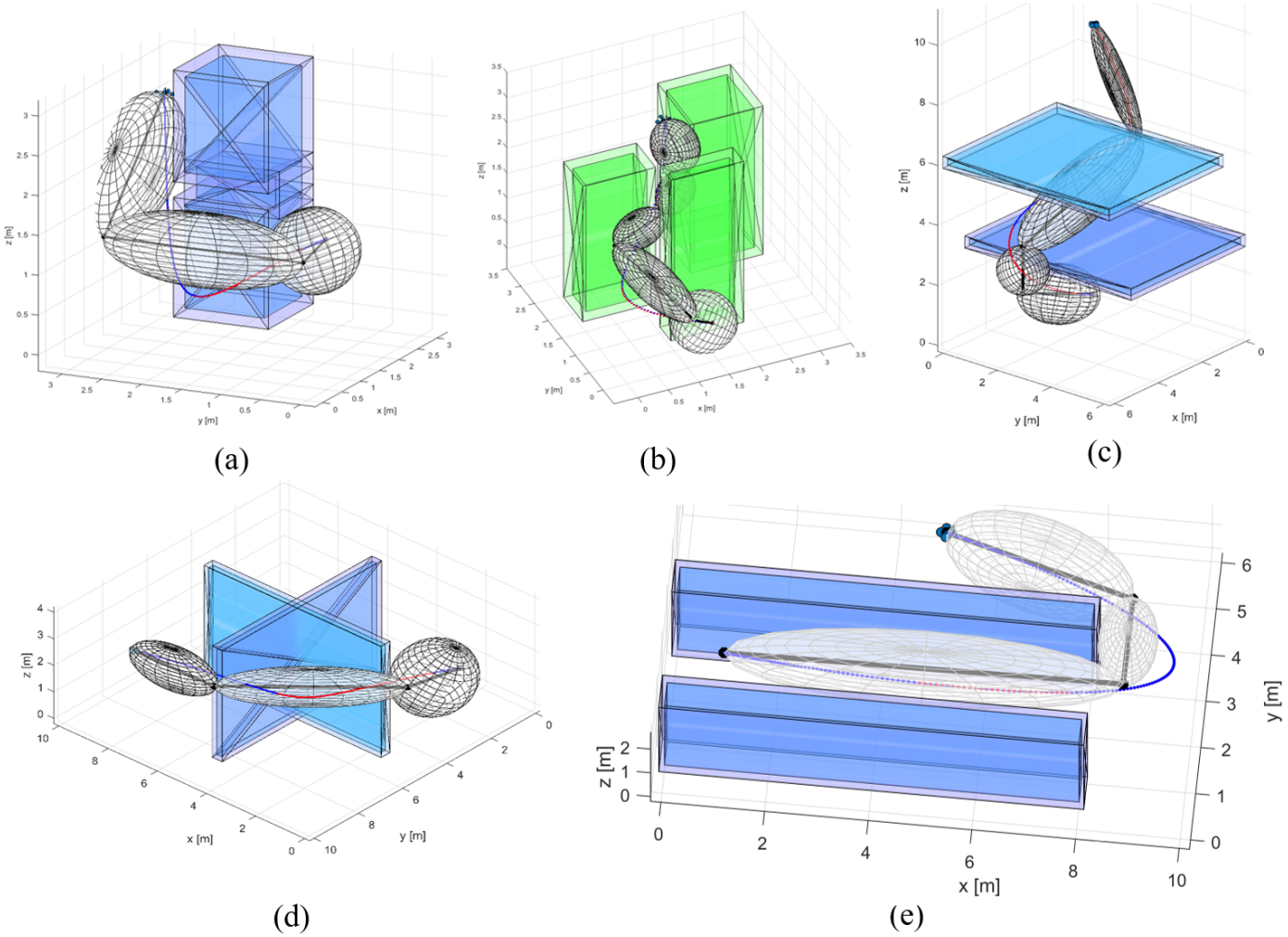

Figure 12. Trajectories simulated with different scenarios. Optimized reference trajectories are presented in blue while trajectories tracked with a controller are presented in red. We used five different scenarios for the simulation of our navigation algorithm. Each scenario represents ellipsoids generated for Obstacle-Free Corridor (OFC) construction for corresponding path segments. Scenarios are named (**a**) Vertical boxes, (**b**) Vertical columns, (**c**) Multiple floors, (**d**) Multiple rooms, and (**e**) Long Corridor.

Lastly, we tested the optimization performance of our trajectory planning pipeline. The path relocation step in the trajectory planning pipeline is the unique feature of our algorithm as compared to state-of-the-art methods to the best of our knowledge. To evaluate the effect of the Path Relocation (PR) step on optimization performance, we simulated our system with this step and without this step. Based on the results of this evaluation, it is evident that our algorithm solves the optimization problem in relatively fewer iterations. Secondly, we observe that the value of the objective function at the end of the optimization considerably decreases by including the path relocation step in the trajectory planning pipeline. Finally, we found that the inclusion of the path relocation step in the trajectory

generation pipeline increases the chances of providing the optimal solution. Results for the optimization evaluation are presented in Table 3.

Table 2. Control errors for different scenarios.

Scenario		Trajectory Tracking Errors (m)					
		Position Error			Velocity Error		
		(X)	(Y)	(Z)	(X)	(Y)	(Z)
(a) Vertical Boxes	Avg	0.0004212	0.0003679	−0.000943	−0.00011	−0.000128	-5.9×10^{-5}
	Std	0.0043779	0.0010597	0.0006761	0.006752	0.001215	0.0014844
(b) Vertical Columns	Avg	0.0020147	0.0016125	−0.002118	−0.00012	0.000418	−0.000220
	Std	0.0078534	0.0033446	0.0015237	0.015492	0.006828	0.0042388
(c) Multiple Floors	Avg	0.0010556	-2.6×10^{-6}	−0.000627	-3.21×10^{-5}	-5.76×10^{-6}	-3.62×10^{-6}
	Std	0.0026302	0.0003872	0.0005803	0.002554	0.000284	0.0007287
(d) Multiple Rooms	Avg	−0.000168	0.0003581	−0.000636	−0.000158	5.586×10^{-5}	-1.09×10^{-5}
	Std	0.0015317	0.0017368	0.0004299	0.001163	0.00131	0.0004918
(e) Long Corridor	Avg	-5.88×10^{-5}	0.00010533	−0.000768	−0.000411	-5.73×10^{-5}	-3.03×10^{-5}
	Std	0.0022831	0.0004938	0.0006614	0.0029307	0.0003791	0.0007432

Table 3. Optimization performance of our algorithm for different scenarios with and without the Path Relocation (PR) step in trajectory planning pipeline.

Scenario	Map Size (m)	No. of Iterations		Objective Function Value	
		w/o PR	with PR	w/o PR	with PR
(a) Vertical Boxes	$3 \times 3 \times 3$	9	8	447.88	0.9492
(b) Vertical Columns	$3 \times 3 \times 3$	11	9	574.15	109.33
(c) Multiple Floors	$6 \times 6 \times 10$	2000	8	Failure [1]	3.1678
(d) Multiple Rooms	$10 \times 10 \times 4$	2000	8	Failure [1]	0.9492
(e) Long Corridor	$10 \times 6 \times 2.5$	2000	7	Failure [1]	1.3553

[1] Solver stopped prematurely after reaching the maximum iteration limit of 2000 iterations. A probable cause for solver failure is the presence of at least one extra-small segment in the path close to the edge of a thin wall.

7. Conclusions and Future Work

Motion planning for dynamical systems through a cluttered indoor environment is a challenging problem because of (a) the constraints on dynamics that have to be incorporated into motion planning; (b) the constraint on the environment that needs to be considered for obstacle-free navigation; (c) the limited computation resources for planning. In this paper, we describe a trajectory generation and optimization algorithm that derives kino-dynamic-feasible, collision-free, and optimal trajectories for quadrotor navigation through cluttered indoor environments. We used different geometric techniques to define obstacle-free corridors inside obstacle-cluttered environments and used those corridors as geometric constraints to optimize trajectories with respect to the fourth derivative of position (snap). Our method is able to generate optimal trajectories within a few hundred milliseconds and is suitable for its implementation in real-time applications. We tested our algorithm in five different indoor settings with varying degrees of obstacle density and environment volume. We analyzed the effect of novel features of our method on optimization performance and found that our method provides much better optimization performance as compared to standard techniques.

In this paper, we solved the problem of inefficient path generation by a path planner by adding a path relocation step in our trajectory planning pipeline. We post-processed the generated path provided by the path planner to make it more feasible and a better guide for obstacle-free corridor generation. However, this step of path relocation increased the execution time of the pipeline by 50 ms to 100 ms. In practice, there is a trade-off between speed and optimization performance of the trajectory planning pipeline. In our future work, we intend to exploit the structures of the algorithm that can be executed in parallel and

formulate our trajectory generation problem with parallel computing architecture specifically for multicore computers. Secondly, our current algorithm infers segment times from segment lengths and does not pay any attention to obstacle density around that segment. We also intend to improve the path planning stage of the algorithm such that we may infer the segment time of each segment of the generated path directly from the path planner. By doing so, we may take into consideration the obstacle density around segments while deciding about segment times because obstacle density considerably impacts quadrotor velocity during navigation. The study of the whole system and the balance between the subsystems may further improve our procedure. We expect the speed of navigation to improve with future research, particularly in the path-planning stage of our pipeline. We also expect that our research will lead to more generalized methods for robotic motion planning that can be equally applied to outdoor, indoor, and underground robotic applications.

Author Contributions: Conceptualization, M.A.A. and H.B.; methodology, M.A.A. and H.B.; software, M.A.A. and J.A.; validation, M.A.A. and J.A.; formal analysis, M.A.A. and J.A.; investigation, M.A.A. and J.A.; resources, H.B.; data curation, M.A.A. and J.A.; writing—original draft preparation, M.A.A. and J.A.; writing—review and editing, M.A.A. and J.A.; visualization, M.A.A. and J.A.; supervision, H.B.; project administration, H.B. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Ministry of Science and ICT (NRF-2017R1A5A1015311).

Institutional Review Board Statement: Not applicable for studies not involving humans or animals.

Informed Consent Statement: Not applicable for studies not involving humans.

Data Availability Statement: Not applicable.

Acknowledgments: Authors would like to thank Natnael S. Zewge at Aerospace Systems and Control Laboratory of KAIST for his time and technical support in this research work.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

QP	Quadratic Programming
OFC	Obstacle-Free Corridor
GSBA	Graph Search Based Algorithms
SBA	Sampling-Based Algorithms
MILP	Mixed Integer Linear Programming
MIQP	Mixed Integer Quadratic Programming
JPS	Jump Point Search
SFC	Safe Flight Corridor
A$*$	A-Star
LPA$*$	Lifelong Planning A-Star
D$*$	Dynamic A-Star
RRT	Rapid-exploring Random Trees
PRM	Probabilistic Road Maps
PR	Path Relocation

References

1. Leonard, J.J.; Durrant-Whyte, H.F. Mobile robot localization by tracking geometric beacons. *IEEE Trans. Robot. Autom.* **1991**, *7*, 376–382. [CrossRef]
2. Hornung, A.; Wurm, K.M.; Bennewitz, M.; Stachniss, C.; Burgard, W. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Auton. Robot.* **2013**, *34*, 189–206.
3. Pomerleau, F.; Colas, F.; Siegwart, R. A review of point cloud registration algorithms for mobile robotics. *Found. Trends® Robot.* **2015**, *4*, 1–104.
4. Aurenhammer, F.; Klein, R. Voronoi Diagrams. *Handb. Comput. Geom.* **2000**, *5*, 201–290.

5. Oleynikova, H.; Taylor, Z.; Fehr, M.; Siegwart, R.; Nieto, J. Voxblox: Incremental 3D euclidean signed distance fields for on-board mav planning. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1366–1373.
6. Han, L.; Gao, F.; Zhou, B.; Shen, S. Fiesta: Fast incremental euclidean distance fields for online motion planning of aerial robots. In Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China, 3–8 November 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 4423–4430.
7. González, D.; Pérez, J.; Milanés, V.; Nashashibi, F. A review of motion planning techniques for automated vehicles. *IEEE Trans. Intell. Transp. Syst.* **2015**, *17*, 1135–1145.
8. Van Nieuwstadt, M.J.; Murray, R.M. Real-time trajectory generation for differentially flat systems. *Int. J. Robust Nonlinear Control. IFAC-Affil. J.* **1998**, *8*, 995–1020. [CrossRef]
9. Mellinger, D.; Kumar, V. Minimum snap trajectory generation and control for quadrotors. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 2520–2525.
10. Mellinger, D.; Kushleyev, A.; Kumar, V. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In Proceedings of the 2012 IEEE International Conference on Robotics and Automation, Saint Paul, MN, USA, 14–18 May 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 477–483.
11. Culligan, K.F. Online Trajectory Planning for UAVs using Mixed Integer Linear Programming. Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2006.
12. Deits, R.; Tedrake, R. Efficient mixed-integer planning for UAVs in cluttered environments. In Proceedings of the 2015 IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 42–49.
13. Richter, C.; Bry, A.; Roy, N. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*; Springer: Cham, Switzerland, 2016; pp. 649–666.
14. Chen, J.; Liu, T.; Shen, S. Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden, 16–21 May 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1476–1483.
15. Gao, F.; Shen, S. Online quadrotor trajectory generation and autonomous navigation on point clouds. In Proceedings of the 2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), Lausanne, Switzerland, 23–27 October 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 139–146.
16. Deits, R.; Tedrake, R. Computing large convex regions of obstacle-free space through semidefinite programming. In *Algorithmic Foundations of Robotics XI*; Springer: Cham, Switzerland, 2015; pp. 109–124.
17. Karaman, S.; Frazzoli, E. High-speed flight in an ergodic forest. In Proceedings of the 2012 IEEE International Conference on Robotics and Automation, Saint Paul, MN, USA, 14–18 May 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 2899–2906.
18. Liu, S.; Watterson, M.; Mohta, K.; Sun, K.; Bhattacharya, S.; Taylor, C.J.; Kumar, V. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *IEEE Robot. Autom. Lett.* **2017**, *2*, 1688–1695.
19. Harabor, D.; Grastien, A. Online graph pruning for pathfinding on grid maps. *Proc. AAAI Conf. Artif. Intell.* **2011**, *25*, 1114–1119. [CrossRef]
20. Zhou, B.; Gao, F.; Wang, L.; Liu, C.; Shen, S. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robot. Autom. Lett.* **2019**, *4*, 3529–3536. [CrossRef]
21. Tordesillas, J.; Lopez, B.T.; How, J.P. Faster: Fast and safe trajectory planner for flights in unknown environments. In Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China, 3–8 November 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1934–1940.
22. Tordesillas, J.; How, J.P. MADER: Trajectory planner in multiagent and dynamic environments. *IEEE Trans. Robot.* **2021**, *38*, 463–476.
23. Zhou, X.; Wen, X.; Wang, Z.; Gao, Y.; Li, H.; Wang, Q.; Yang, T.; Lu, H.; Cao, Y.; Xu, C.; et al. Swarm of micro flying robots in the wild. *Sci. Robot.* **2022**, *7*, eabm5954. [CrossRef] [PubMed]
24. Zhou, X.; Zhu, J.; Zhou, H.; Xu, C.; Gao, F. Ego-swarm: A fully autonomous and decentralized quadrotor swarm system in cluttered environments. In Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 30 May–5 June 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 4101–4107.
25. Wang, H.; Yu, Y.; Yuan, Q. Application of Dijkstra algorithm in robot path-planning. In Proceedings of the 2011 Second International Conference on Mechanic Automation and Control Engineering, Hohhot, China, 15–17 July 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 1067–1069.
26. Hart, P.E.; Nilsson, N.J.; Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *4*, 100–107. [CrossRef]
27. Pivtoraiko, M.N. Differentially Constrained Motion Planning with State Lattice Motion Primitives. Ph.D. Thesis, Carnegie Mellon University Pittsburgh, PA, USA, 2012.
28. Pivtoraiko, M.; Kelly, A. Kinodynamic motion planning with state lattice motion primitives. In Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, San Francisco, CA, USA, 25–30 September 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 2172–2179.
29. Stentz, A. Optimal and efficient path planning for partially known environments. In *Intelligent Unmanned Ground Vehicles*; Springer: Boston, MA, USA, 1997; pp. 203–220.
30. Koenig, S.; Likhachev, M.; Furcy, D. Lifelong planning A*. *Artif. Intell.* **2004**, *155*, 93–146.

31. Belanová, D.; Mach, M.; Sinčák, P.; Yoshida, K. Path planning on robot based on D* lite algorithm. In Proceedings of the 2018 World Symposium on Digital Intelligence for Systems and Machines (DISA), Košice, Slovakia, 23–25 August 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 125–130.
32. Quan, L.; Han, L.; Zhou, B.; Shen, S.; Gao, F. Survey of UAV motion planning. *IET Cyber-Syst. Robot.* **2020**, *2*, 14–21. [CrossRef]
33. Karaman, S.; Frazzoli, E. Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.* **2011**, *30*, 846–894.
34. Burke, D.; Chapman, A.; Shames, I. Generating minimum-snap quadrotor trajectories really fast. In Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 24 October 2020–24 January 2021; IEEE: Piscataway, NJ, USA, 2020; pp. 1487–1492.
35. De Almeida, M.M.; Akella, M. New numerically stable solutions for minimum-snap quadcopter aggressive maneuvers. In Proceedings of the 2017 American Control Conference (ACC), Seattle, WA, USA, 24–26 May 2017; IEEE: Piscataway, NJ, USA, 2017; pp. 1322–1327.
36. Lee, T.; Leok, M.; McClamroch, N.H. Geometric tracking control of a quadrotor UAV on SE (3). In Proceedings of the 49th IEEE conference on decision and control (CDC), Atlanta, GA, USA, 15–17 December 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 5420–5425.