Efficient Planning for High-Speed MAV Flight in Unknown Environments Using Sparse Topological Graphs

Matthew Collins
August, 2019
CMU-RI-TR-19-62

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania

Thesis Committee:
Nathan Michael, *Chair*
Maxim Likhachev
Rogerio Bonatti

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Abstract

Safe high-speed autonomous navigation for MAVs in unknown environments requires fast planning to enable the robot to adapt and react quickly to incoming information about obstacles within the world. Furthermore, when operating in environments not known *a priori*, the robot may make decisions that lead to dead ends, necessitating global replanning through a map of the environment. Many state-of-the-art approaches are limited to reduced velocities—precluding their usage in time-critical applications such as search and rescue—while those that do allow for high-speeds are myopic, restricting their success in large-scale, complex environments that require reasoning about the world outside of a local planning grid. This thesis proposes a computationally efficient planning architecture for safe high-speed operation in unknown environments that incorporates a notion of longer-term memory into the planner to enable the robot to accurately plan to locations no longer contained within the local map. A motion primitive based local receding horizon planner that uses a probabilistic collision avoidance methodology enables the robot to generate safe plans at fast replan rates. To provide global guidance, a sparse topological graph is created online from a time history of the robot's path and a notion of visibility within the environment that may be searched to find alternate pathways towards the desired goal if a dead end is encountered. The safety and performance of the proposed planning system is evaluated at speeds up to 10m/s, and the approach is tested in a set of large-scale, complex simulation environments containing dead ends. These scenarios lead to failure cases for competing methods; however, the proposed approach enables the robot to safely reroute and reach the desired goal.

Acknowledgments

I would like to thank my advisor, Prof. Nathan Michael for all his support and guidance during my graduate studies. Thank you to the members of RISLab for all the help along the way, specifically Arjav for all the planning discussions and Wennie for all the feedback on my thesis work. Finally, thank you to my family and friends, without your support this would not be possible.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Autonomous navigation in unknown environments presents a challenging planning problem due to the safety guarantees that must be met and the number of subsystems that are affected, e.g., control, mapping, and state estimation. Due to the wide ranging applicability in exploration and search and rescue, there is interest in developing robotic systems that can meet these demands and enable faster search and rescue operations. Micro aerial vehicles (MAVs) are increasingly being deployed for search and rescue and reconnaissance due to their affordability and maneuverability, which enables coverage over a large area in a short amount of time. Due to these technological advantages, MAVs are beginning to become a ubiquitous tool. A 2019 study from Bard College reported that at least 910 state and local police, sheriff, fire, and emergency-service agencies in the U.S. are actively using drones [42]. This investment in MAVs has yielded results. Michael Oldenburg, the Product Communication Lead at DJI, stated that there have been 144 cases of people rescued with the help of a drone, and that this numbers continues to increase each month [10]. Typically these MAVs are operated by trained human pilots, but there been works [43, 47, 50] attempting to develop autonomy for search and rescue missions.

The common usage of MAVs in search and rescue scenarios is to fly at high altitudes

and use the incoming camera or thermal data to locate survivors. This has been used to locate humans lost in the wilderness, and even to try to identify survivors in rubble following natural disasters [42]. With the increase in autonomy capabilities, recent works have looked into using MAVs for cluttered, indoor environments. One example is the DARPA Fast Lightweight Autonomy (FLA) program that seeks to develop high-speed navigation to enable urban reconnaissance and search and rescue in dense forests or in damaged buildings [1]. To operate in these types of environments, MAVs need to be small to accurately maneuver through the tight confines of hallways within buildings and through dense clutter. Additionally, the time-critical nature of the real-world applications motivates the need for fast MAV flight.

For autonomous search and rescue and defense operations, the core problem from a planning perspective is that the robot must be capable of online navigation at high-speeds through unknown environments. Operation in these conditions places demands on safety, planner reaction time, and robustness. Most current works limit the velocity, preventing the high-speed agile flight ideal for time-critical scenarios. Fast flight with limited sensor field of view (FOV) requires the system to process sensor data quickly, making real-time planning and mapping paramount for ensuring performance and safety. The challenge is that MAVs are constrained in terms of size, weight, power, and compute. As a result, online planning must be both **computationally-** and **memory-efficient**.

Current MAV planning methodologies have enabled high-rate planning in unknown environments by using motion primitive libraries [11, 41] or fast optimization-based techniques [48], but these planners can be myopic, limiting their applicability in large-scale, complex environments. Typically, the robot builds a global map of the environment to be used for planning; however, when the scale of the environment is not known *a priori*, works often employ a sliding map to cap memory usage [35]. The drawback of this approach is that the sliding map does not provide information about the world outside its boundaries.

This becomes an issues if its size if not large enough to cover the entire area of interest, leading to scenarios where the robot gets stuck and can no longer plan to the desired location. Another issue with the conventional sliding map approach is that it does not retain any information of the robot's travel history, which would be useful to enable the robot to safely and accurately return to its takeoff location.

These challenges and the operational requirements necessitate a planning methodology that is both computation- and memory-efficient, while still performing at high-speeds. Moreover, the robot cannot be overly myopic, resulting in the planner failing due to limiting its ability to make an informed decision based on the global structure of the environment. These necessities motivate the work in this thesis of developing a computationally-efficient planning framework that generates smooth, safe trajectories, and builds a memory-efficient graph representation of the environment online enabling the robot to reason and accurately plan towards locations outside of its local map.

1.1 Thesis Problem

This thesis addresses the planning problems associated with navigating safely at high-speeds in unknown environments while taking into account the computation and memory limitations for real-time operation. The proposed approach decouples the planning problem into a hierarchical local and global framework. The local component works at a higher fidelity and faster rate to generate trajectories for the robot to execute, while the global component provides guidance for the location towards which the robot should plan, i.e., the desired movement direction. This division naturally leads this thesis to focus on two primary problems:

1. Developing a fast, efficient local planning strategy that generates smooth, collision-free trajectories for the robot to execute to achieve high-speed flight performance.

2. Designing a memory-efficient mapping strategy that adds a longer-term memory component to allow the robot to reason about the nature of the environment it has previously traversed that may be outside of its local map. Specifically, the robot must to be able to identify possible alternative pathways from that which it took and to be able to quickly and accurately return to its starting location after operation.

1.2 Contributions and Outline

This thesis provides two contributions. The first is the development and evaluation of a local planning framework that is designed to be efficient, safe, and robust even when the robot is operating at high-speeds. Next, the online construction of a global sparse topological graph is presented that is built through a combination of the time history of the robot's path and the identification of unexplored pathways in the environment. These two planning modules are then integrated together to yield a planning architecture that is capable for operation in large-scale, complex environments where conventional sliding map planning approaches fail.

The remainder of this thesis is structured as follows. Chapter 2 provides a review of concepts necessary for understanding issues encountered when planning within unknown environments and a discussion of relevant research in the field. Chapter 3.2 provides an overview of the proposed planning system architecture and then details and evaluates the local planner component. Chapter 4 describes the creation and usage of the sparse topological graph for the global planner aspect of the system architecture. Additionally, simulation results for the full planning framework are presented. Finally, Chapter 5 discusses areas and directions for future work and the conclusions garnered throughout the duration of this work.

Chapter 2

Background

This chapter serves as a reference to provide background information on topics that will be discussed throughout the body of the thesis. First, an overview of planning methods will be examined with a focus on planning for MAVs. Afterwards, the trajectory representation of motion primitives will be detailed. Next follows a discussion on planning under uncertainty and probabilistic collision checking methods. Then attention will be brought to the topic of planning in unknown environments, before finishing with a brief look into topological graphs.

2.1 Planning Methods

Most planning approaches can be divided into one of three predominant planning methods: search-based, sampling-based, or optimization-based planning. All three of these methods have been widely used in planning for MAVs. Search-based methods, such as A*, operate on a discretized graph of the environment and offer completeness guarantees for finding the shortest path. Liu et al. [24] presents a search-based planning method that constructs a state lattice of short duration motion primitives that are generated by solving an optimal control

problem. Graph search algorithms are then used to find a smooth, dynamically-feasible trajectory from a start configuration (does not assume stationary initial state) to a desired goal configuration. This work has been extended to allow for planning in SE(3), by modeling the robot as an ellipsoid and using the orientation obtained through the acceleration of the system [26], thus allowing the robot the plan through narrow gaps that are only feasible if the vehicle is passing through the gap at certain orientations. MacAllister et al. [28] uses a search-based planning method known as Anytime Dynamic A* (AD*)—an anytime and incremental variant of A*—that is run backwards from the goal location to the robot pose, allowing for the greatest reuse of the previous path as the robot moves. A state lattice of motion primitives comprised of 3D position and yaw angle is constructed upon which AD* is run to determine the trajectory for the quadrotor to follow. These search-based planners are subject to discretization effects, do not incorporate vehicle dynamics, and can be slow for large 3D maps. Despite this, search-based planners are widely used for MAVs for at least part of the planning problem.

The second type is sampling-based planners, which sample the configuration space of the robot. FMT* [20] and BIT* [14] are examples of sampling-based planning techniques. In the context of MAV planning, [40] refines an initial geometric path obtained by RRT*. Allen and Pavone [2] presents a real-time kinodynamic planning framework built upon the kinodynamic extension of the FMT* algorithm [44]. During an offline phase a SVM is trained on the optimal boundary value problem, such that during the online planning phase, it may be used to quickly classify whether a sampled configuration is dynamically feasible allowing for real-time planning. These planners offer asymptotic optimality guarantees and scale well for high-dimensionality workspaces.

The third type is optimization-based planners. These include Mixed Integer Quadratic Programs (MIQP) used in [22] or [49]. Another popular method for MAVs is optimizing polynomials. Mellinger and Kumar [29] shows that a quadrotor may be modeled as a

differentially flat system—the states and inputs may be written as algebraic functions of *flat outputs* and their derivatives—enabling real-time generation of optimal trajectories. For accurate flight, qualities such as smoothness or the minimization of higher-order terms are desired for the resultant trajectory. Methods such as [25, 29, 32, 38, 40] seek to optimize trajectories subject to some metric, e.g., minimize control effort, distance to obstacles, etc. Typically for all these methods the differential flatness of quadrotors is exploited and polynomials trajectories in the position and yaw ($x(t)$, $y(t)$, $z(t)$, and $\psi(t)$) are computed.

Many works use a combination of these planners to extract the maximum benefits of each in terms of speed and performance. Examples include [15, 31, 40, 41, 48, 49, 51]. A typical approach is to use a search-based planner to quickly obtain a geometric path with a sampling and optimization-based planner to generate a trajectory for the robot to follow.

Recently there have been works incorporating learning into planning. The learning is used to improve the performance of these standard planning methodologies [37] or allow for planning in lower-dimensional latent spaces [19]. Another example is [39], which uses an autoencoder to detect the novelty within the environment relative to the training data to determine which type of planning methodology to use.

2.2 Motion Primitives

Another planning method that has been widely used in robotics and for MAVs is motion primitives. Using closed form equations, trajectories can be efficiently generated between start and end configurations. Due to the differentially flat nature of the quadrotor model, [18, 30] use a triple integrator model to generate motion primitives from the robot's initial configuration to a desired end configured over a specified time duration. They present a closed-form equation that minimizes a cost function (e.g., minimum jerk) to yield a fifth-order polynomial trajectory for each axis. Instead of the triple integrator model [45, 52]

uses a unicycle kinematics model that has been modified for use with aerial robots. By propagating the unicycle dynamics, forward-arc motion primitives parameterized as eighth-order polynomials are generated for the trajectories. By sampling over a parameter space of inputs for motion primitives, a motion primitive library can be created. Instead of continuous non-convex optimizations for online flight trajectory generation, the best motion primitive can be chosen from a discrete set within the library and sent to the robot for execution. Motion primitives have the advantages of providing optimal trajectories and are very computationally efficient.

2.3 Planning Under Uncertainty

When planning on any real system or in an environment that is not known fully known, the robot will have some uncertainty the should be factored into the planner. For operation in unknown environments with limited sensor range, this uncertainty stems from the lack of complete knowledge of the map along the path. Janson et al. [21] seeks to develop a framework for planning that guarantees that the robot never collides with obstacles, while trying to define some notion of optimality for planning in unknown environments. They seek to ensure that the robot never selects policies that could potentially lead to an inevitable collision state with respect to any unobserved part of the environment. This means the plan must take into account how actions affect future actions and their methodology inherently leads the planner to make decision such as taking wide turns at blind corners. By creating a risk density map that acts as a cost function for the planner, [36] allows for navigation in unknown, cluttered environments without explicitly detecting and tracking objects, even those that are dynamic. One common problem with operating in uncertain and incomplete information of the environment is that the robot may enter a state from which it cannot safely exit. Fridovich-Keil et al. [13] builds a graph of forward-reachable state within free

space and implicitly an under-approximation for the backward-reachable set from the initial state that ensures the selected path will always be safe.

Another avenue that may impact the planner is state uncertainty. Methods such as [5, 53] take the uncertainty of the state estimate into account during their planning procedure to favor paths that not only get the robot closer to the desired goal, but also provide good state estimates due to high photometric texture.

2.4 Probabilistic Collision Checking

Classical approaches for collision avoidance are deterministic and do not take state uncertainty into account. These methods leave little room for error during operation and do not provide any information about collision besides a yes or no. This either/or result can not say whether one trajectory is safer than another, even though one may pass much closer to an obstacle. By adopting a probabilistic approach, a collision probability is returned which provides a more detailed notion of the safety of a trajectory and can better handle the uncertainty that exists within the system. Du Toit and Burdick [9] presents a probabilistic collision checking using chance constraints that takes into account the uncertainty associated with the sensed object by using a small volume approximation. Zhu and Alonso-Mora [54] presents a probabilistic collision checking formulation and avoidance method that allows for navigation among dynamic obstacles including other robots. The approximate collision probability forms a tight bound allowing for real time operation using a chance constrained nonlinear model predictive control framework. The uncertainties in the positions of the robot and the obstacle are assumed to be Gaussian distributed, while the probabilistic nature of the collision checking helps account for and deal with localization and sensing uncertainties.

2.5 Planning in Unknown Environments

Navigation in unknown environments is a challenging planning problem due to the limited sensor information and the computational constraints that must be imparted on the system to ensure safety during operation. Due to the wide ranging applicability in domain such as search and rescue and exploration, this topic has become an active research area. The works in this area are varied and use a multitude of different planning methodologies. Despite the differences, there are some common threads that appear throughout the works such as the integration of the mapping and planning components and the use of a hierarchical planning approach. What follows are short descriptions of a subset of the works in this area.

Many works try to define a notion of a safe flight corridor within the map representation to create a connected region through which an optimal trajectory can be calculated. Gao and Shen [15] generates a safe flight region through a point cloud of the environment through which it can optimize a trajectory. Using a sampling-based rapidly exploring random graph (RRG) where each node is the center of a spherical safe region. Edges in the graph indicate connections between these safe regions. A* is run over the RRG to find the connected safe region composed of connected spheres through which the trajectory will be generated. The trajectory generation problem is formulated as a quadratically constrained quadratic program to optimize a set of minimal jerk piecewise polynomials. Liu et al. [23] presents a system that addresses high speed autonomous navigation in unknown environments. An A* search is run to multiple candidate goal points, a convex decomposition of free space is performed around the selected path, and finally a polynomial trajectory is optimized through these polyhedral regions. Chen et al. [4] represents the environment using an octree data structure from which they are able to generate free-space flight corridors, i.e., connected 3D grids within free-space that are initialized via a A* search. With this flight corridor in place, a minimum-snap piecewise polynomial may be optimized that meets

dynamic and corridor constraints. Watterson and Kumar [51] uses a polyhedral convex decomposition of free space Delaunay tetrahedralization through which trajectories may be optimized for receding horizon control. The short range receding horizon controller uses motion primitives which the robot follows. Liu et al. [25] generates a *Safe Flight Corridor* through a convex decomposition of free space, generating a series of overlapping polyhedra that provides a connected region from the starting location to the goal. A quadratic program (QP) then optimizes polynomials through the region for the quadrotor to follow. This work extended the Jump Point Search (JPS) search algorithm originally presented in [17] to 3D; however, the proposed pruning rules for the 3D extension are not entirely accurate, thus adding more neighbors than is required.

Oleynikova et al. [33] describes a complete system for mapping, planning, and control for a quadrotor in cluttered environments. A probabilistic occupancy grid and a Euclidean Signed Distance Field (ESDF) is created of the environment. Using their Local Continuous Optimization planner [32] that optimizes a trajectory through a predefined set of waypoints. Collisions are modeled as soft constraints and the problem structure follows that defined in CHOMP [38]. This method takes a conservative approach, modeling unknown space as occupied and only planning within free space.

Tordesillas et al. [48] uses multi-fidelity models to better capture some of the higher-order dynamics that are typically not included in the global planner. This is done to reduce the possibility of unstable behavior created through the interaction between the two planner levels. Uses geometric (JPS) search for global planner, velocity-controlled primitives for the middle fidelity, and jerk-controlled primitives for trajectories for the quadrotor to follow. Uses a sliding map voxel grid and an array of kd-trees for collision checking. Tordesillas et al. [49] extends the approach of [48] by allowing the local planner to generate trajectories in free and unknown space. This less conservative approach enables faster flight speeds. Convex decomposition, presented in [25], is utilized and an MIQP formula-

tion is used to generate the optimal trajectory through the free space regions. To maintain safety, a safe back-up is also generated that remains entirely in free space.

The following works take a similar approach to the proposed work and represent trajectories as motion primitives that are chosen from a library. Lopez and How [27] models the quadrotor as a state and input constrained triple integrator as in [30] and performs collision avoidance with only the instantaneous sensor information. To ensure safety, the generated motion primitives are constrained to remain within the sensor field of view (FOV). Collision checking is performed using kd-trees and is done efficiently using safety certificates to more intelligently sample points along the trajectories. Florence et al. [11] uses a library of motion primitives (25 used in paper) for fast replanning allowing for high-speed operation. For robustness even in the presence of high state uncertainty that may affect the accuracy of fusion-based mapping approaches, only instantaneous depth information is used creating a local map [12]. Collision probabilities are checked probabilistically using the chance constrained formulation of [9], which better deals with uncertainty within the sensor data. Primitives are discretized over the acceleration and heading space, and a triple-double integrator model is utilized for the system dynamics. At each planning iteration, the primitive with the lowest cost according to a Euclidean progress metric and a cost on the terminal speed is chosen for the quadrotor to follow. To allow for navigation among non-convex obstacles an A* global planner is used to provide a waypoint for the robot. Experiments show flight results at up to 10 m/s, but are limited to 2D. Ryll et al. [41] uses a receding horizon planner comprised of a set of 270 minimum-jerk motion primitives using the closed-form solution presented in [30]. The primitive set is generated by sampling over yaw-heading, final altitude, and distance from the vehicle. The framework consists of a local planner (i.e., motion primitives) which is responsible for the reactive collision avoidance and a global planner that is tasked with providing the local goal. Planning is performed on two body-centered $60\text{m} \times 60\text{m} \times 6\text{m}$ voxel grids with a resolution of 0.5m,

where the first is probabilistic occupancy grid of fused sensor data, and the second uses an incremental distance transform to determine the distance to the nearest obstacle for each voxel location. A breadth-first search (2Hz) is run over a weighted sum of these two grids determine the global path from which the local goal may be generated. A new primitive is chosen at sensor rate (30Hz) that has the minimum cost according to a metric that incorporates Euclidean distance to goal, the number of control points along the primitive that lie in unknown space or outside the map, and the distance to obstacles. Experiments present flight tests up to 9m/s.

2.6 Topological Graphs

Topological graphs seek to represent the connectivity of the environment, or in other words, how can the free space within the environment be traversed. This methodology incorporates a greater degree of contextual information present within the environment, such as what areas connect to each other, that is not encoded in a typical occupancy map. The traversable areas within the map can then be represented by a small set of nodes, which allows for very fast planning on a topological graph. Thrun [46] showcased topological planning for robotics through a construction of a 2D topological graph from an occupancy grid by using a Voronoi diagram. The key idea in their algorithm for partitioning the space was to find "critical points," such as doorways, that section of parts of the grid into distinct regions. Voronoi diagrams have been used in other works to partition the space, but another method to generate the topology of the space is through a convex decomposition of free space, as is done in IRIS [7]. The map is decomposed into polygonal regions that provide the connectivity of the traversable space. The main drawback of this method is the computation time, which is very high for 3D environments. For this reason, it is only used for known environments where there is time available to preprocess the map, and is not a useful method

for real-time operation. The computation time issue is alleviated in [25], enabling online usage. By using a geometric path to seed the decomposition the computation is reduced, but this comes at the cost of limiting the planning to the single homotopy class of the original geometric path. Cover et al. [6] takes a different approach and creates a distance field around obstacles to construct a sparsely connected graph (based upon visibility graphs) to quickly find paths to a set goal. Blochliger et al. [3] generates a topological graph from the feature map provided by a feature-based SLAM system. The occupancy information from the features are used to create a set of convex free space clusters where the connection points between these clusters form the vertices of the topological graph. Oleynikova et al. [34] extracts a 3D Generalized Vornoi Diagram from an ESDF of the environment to generate a skeleton structure representing the topological information present within the map. The assumption for the latter two works is that the robot has an initial exploration to create the map that is processed to generate the sparse graph; therefore, this methodology is geared more towards planning to revisit locations in the previously explored map rather than to be created and used online.

Chapter 3

Planning Architecture and Local Planner

3.1 Planning Architecture Overview

Prior to delving into the planner components, it is important to take a step back and look at the whole system and examine how the planner interacts with the hardware and the other software systems. The realities and limitations of these components need to be kept in focus during the decision making process. The remainder of this section provides a brief overview of the whole system architecture as a primer to provide context of how all the planner elements discussed in this work fit together to achieve the goal of efficient planning for safe, high-speed flight in unknown environments that introduces a notion of longer-term memory that is lacking in current methods.

A RGB-D camera is used as the primary sensor to provide depth measurements that will be used to build the maps that will be utilized by the planning subsystem. Two maps are created: a kd-tree local map and a sliding voxel map. The kd-tree mapping has very low latency—runs at the sensor rate of 30Hz—by only maintaining a spatially consistent

map of a small area surrounding the robot. This type of map works very well for reactive approaches and for collision avoidance due to efficient nearest neighbor queries. Moreover, it is not impacted by state uncertainty because it only considers sensor information around a local relative pose. There is a drawback to this mapping technique, as it is computationally expensive to run geometric search-based planners with all the collision checks. For this reason a fusion-based voxel grid is utilized to aid the planner in these scenarios, while a sliding approach caps the memory and computation needs for the map.

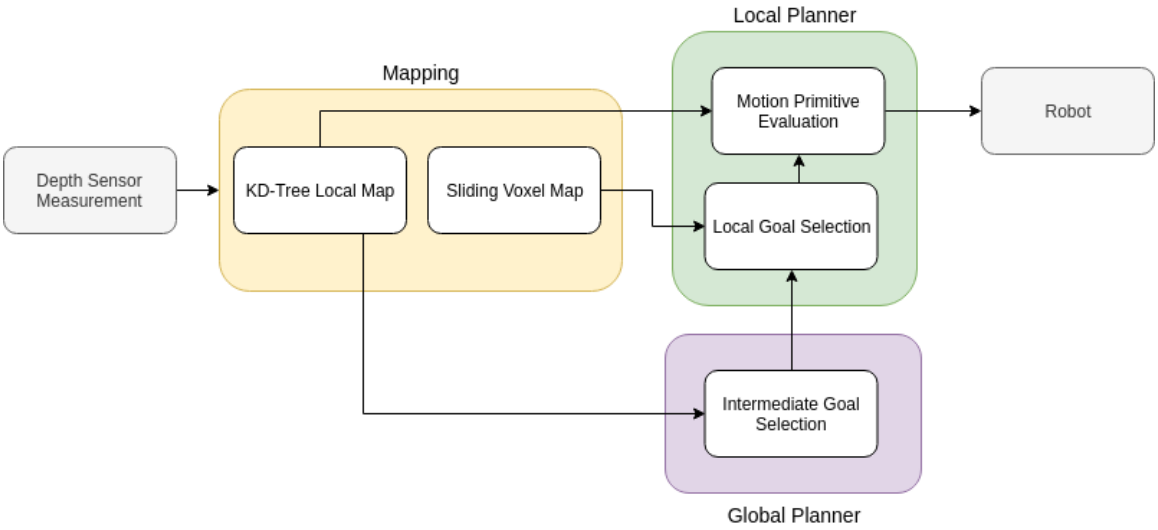


Figure 3.1: System Architecture

The planning subsystem is comprised of two distinct components: a local planner and a global planner. This type of division of labor for the planning system is common when navigating in unknown environments and may be seen works such as [4, 11, 23, 31, 33, 41, 48, 51]. The local planner runs at a much faster frequency and supplies a trajectory for the robot to execute. For this work a motion primitive library is used and collision checks are performed using the kd-tree local map. At each replanning step a primitive is chosen which gets the robot closest to a local goal that is selected by running a geometric planner to an intermediate goal point that is projected onto the sliding voxel map. The global planner decides this intermediate goal and builds a sparse topological graph of the explored section

of the environment to aid in global navigation of the robot, and to help mitigate long-term memory issues that arise due to the fixed size of the sliding map. A detailed description of the local planner is provided in Section 3.2 and the global planner in Section 4. A system architecture diagram is shown in Figure 3.1.

3.2 Local Planner

For high-speed, aggressive flight in unknown environments it is imperative to compute trajectories at a high replan rate to ensure safety with respect to the objects revealed through the incoming sensor measurements. These operating conditions also require precise position tracking of the robot, meaning the planner must generate smooth, feasible trajectories. With these constraints in mind, this work utilizes a local planner that produces a trajectory for the MAV to execute. The design goals for the local planner are efficiency (high replan and reaction rates), safety, robustness to uncertainty, and that it is able to handle complex environments that feature non-convex obstacles and dead ends.

The rest of this section describes the local planning component in the system architecture and provides a break down of each constituent part that makes up the planner. After the detailed overview, the algorithm and methodology behind the fully integrated planner is discussed in Section 3.2.4. Finally, results are presented in Section 3.2.5 that showcase the performance of the local planning component and offer a rationale for some of the design decisions that underlay the proposed methodology.

3.2.1 Motion Primitives

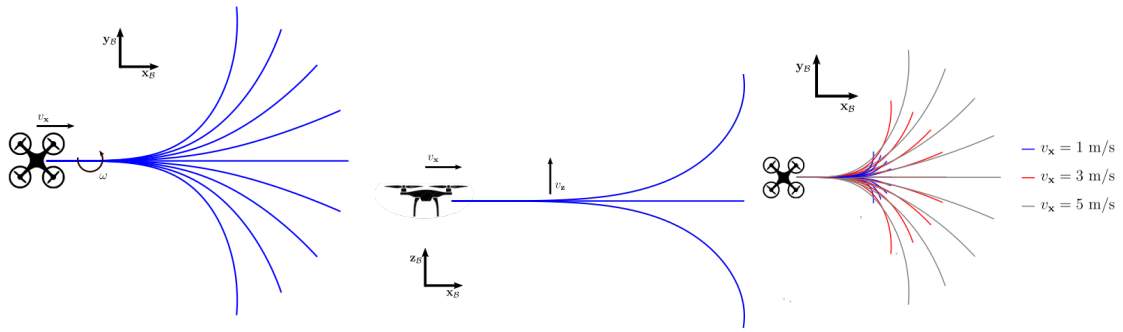
For accurate position tracking for multirotors, the trajectory references should be continuous through high-order derivates of position [29]. Continuous optimization-based planning methods meet this criteria, but can be computationally expensive. Another option, and the

one chosen in this work, is to use a collection of predefined discrete motion primitives creating a motion primitive library from which trajectories may be selected for the robot to execute. This method has the benefits of very fast replanning and smooth control references; however, it comes at the cost of expressiveness of the planner, i.e., the discretization limits the maneuverability of the multirotor. Since motion primitive planning approaches have shown great performance [11, 27, 30, 41, 45, 53] and this work seeks high-rate efficient planning, they were chosen for the local receding horizon planner.

While other works such as [41, 53] have used closed-form 5$^{\text{th}}$ order polynomials according to [18, 30] or a double-triple integrator model [11], the proposed methodology utilizes forward-arc motion primitives framework proposed by Yang et al. [52], which are generated by forward propagating a unicycle kinematic model with constraints on the higher-order endpoints. The primitives are parameterized by $\mathbf{a} = [v_{\mathbf{x}}\ v_{\mathbf{z}}\ \omega]$, where $v_{\mathbf{x}}$ and $v_{\mathbf{z}}$ are the desired forward ($\mathbf{x}_{\mathcal{B}}$ axis) and altitude ($\mathbf{z}_{\mathcal{B}}$ axis) end velocities in the body frame $\mathcal{B}$ of the robot, and ω is the yaw angular rate. From this action parameterization, the unicycle dynamics model, and a trajectory duration τ, a motion primitive $\gamma(t)$ is generated as a time-parameterized 8$^{\text{th}}$ order polynomial (3.1) where the end position is unconstrained and all derivatives higher than velocity are constrained to zero at the end point. Finally, the motion primitive is transformed into the world frame for evaluation and use with the planner. Additionally, a stopping primitive is added to the library that consists of an action parameterization of all zeros that may be executed to ensure safety of the robot if all other primitives are not feasible. Please refer to [45, 52] for a more detailed description on forward-arc motion primitives.

$$\dot{\gamma}(t) = [v_{\mathbf{x}}\cos(\omega t) \quad v_{\mathbf{x}}\sin(\omega t) \quad v_{\mathbf{z}} \quad \omega], \quad t \in [0, \tau] \tag{3.1}$$

To create the library uniform samples are taken over the action space in ω and $v_{\mathbf{z}}$,



(a) ω variation (b) $v_{\mathbf{z}}$ variation (c) $v_{\mathbf{x}}$ variation

Figure 3.2: Parameterization of forward-arc motion primitive library [52]

with the bounds and number of samples user specified. Instead of sampling over the end velocities $v_{\mathbf{x}}$ to create a fixed library, an adaptive method is used where the velocities are dependent upon the current reference velocity of the robot. For example, if the current reference velocity is 3m/s and three different velocities are used, end velocities of 1m/s, 3m/s, and 5m/s might be chosen, as shown in Figure 3.2c. The idea behind this decision is that due to the high replan rate and the system dynamics, the speed of the robot will naturally ramp up or down (choose primitives close to its current speed) and acceleration constraints will limit the jump between linear velocity levels in the primitive library. By taking an adaptive approach, the size of the library is reduced—faster computation times—without a substantial loss in performance.

3.2.2 Probabilistic Collision Checking

For safe navigation within unknown environments, obstacle avoidance is imperative; however, uncertainties that exist in state estimation and noise in sensor information and control policies complicate the problem. Typical collision checking methods, such as discretely checking a voxel grid or using a fixed size collision radius, do not inherently take the uncertainties of the robot and obstacle positions into account. To deal with this problem, this work looks at chance-constrained collision avoidance literature [9, 54] for probabilistic

collision checking methods.

Specifically this works implements the collision probability equation developed by Zhu et al. [54] since it maintains a tighter bound of the collision probability. By assuming that the state uncertainty of the robot follows a Gaussian distribution, the collision probability can be calculated solely through the mean and covariance of the state. For a position p_i along the trajectory, the collision probability bound at that point can be calculated according to

$$P(C, p_i) \leq \frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{r_c - a_{ij}^T(p_i - p_j)}{\sqrt{2a_{ij}^T \Sigma_p a_{ij}}}\right) \tag{3.2}$$
$$a_{ij} = \frac{(p_i - p_j)}{||p_i - p_j||}$$

where r_c is the collision radius of the robot (assumed to be spherical), p_j is a obstacle point, and Σ_p is the robot position covariance. For safety the upper bound is taken as the collision probability.

The environment is represented as a spatially consistent kd-tree map [45] that stores a history of depth sensor measurements obtained at poses that lie in the vicinity of the vehicles current pose. The benefits of this mapping approach are that it runs at sensor rate, it performs better in the presence of state uncertainty, and the kd-tree structure of the map allows for efficient queries. By running at sensor rate, the collision avoidance should yield a more reactive, safer planner since all new sensor information will be present during collision checks. At every query point p_i along the robot trajectory, a one-nearest neighbor search is performed and the returned depth point p_j is used for the collision calculation. Since this map representation only stores occupied points, it is possible that the robot could choose a trajectory that takes it into space that has not been observed by the sensor. To alleviate this issue, the local map stores a history of sensor field of view (FOV) frustums

that are used as an additional check. If any point along a trajectory is outside this FOV set, the trajectory is deemed infeasible as a conservative measure.

For calculating the robot position covariance, a method similar to [11] is utilized where the position uncertainty scales linearly with the speed of the vehicle. Due to the differentially flat nature of MAVs, the robot is modeled as a triple integrator (see Figure 3.4) with the robot's position uncertainty given by

$$\Sigma_p = \frac{\sigma}{10} t^2 \begin{bmatrix} v_x & 0 & 0 \\ 0 & v_y & 0 \\ 0 & 0 & v_z \end{bmatrix}. \tag{3.3}$$

It should be noted that each axis is assumed independent to yield a diagonal covariance matrix. σ is a noise level parameter that can be defined based upon the expected level of velocity noise within the system, and for this work σ is set to 0.1. At low speeds, the position uncertainty will be small which allows the robot to move more tightly around obstacles. At higher speeds, the covariance will be larger, which will essentially increase the effective collision radius for the robot. This will push the robot further away from obstacles or force the robot to slow down to remain safe; both behaviors are desirable to account for the lower control margin as the flight approaches the dynamic limits of the robot. A comparison for how the position covariance scales with velocity is shown in Figure 3.3.

(a) Low Speeds (b) High Speeds

Figure 3.3: Effect of velocity on position covariance with the 1-σ of the position covariance shown in pink and an obstacle in black. At low speeds, the covariance will be smaller, allowing the robot to move closer around obstacles. At high speeds, the robot has less control margin to maneuver around obstacles. The higher position uncertainty pushes the robot further away from obstacles or forces it to slow down to reamin safe.

$$n = \text{ceil}\left(\frac{v_{\mathbf{x}}\tau}{r_c}\right)$$
$$n = \begin{cases} 3, & n < 3 \\ n, & 3 \leq n < 20 \\ 20, & n \geq 20 \end{cases} \tag{3.4}$$

Each motion primitive is sampled at n positions (3.4) uniformly in time, where n is dependent upon the forward velocity of the primitives, its duration, and the collision radius. n is capped at a maximum of 20 for computation and a minimum of 3 to ensure that there enough samples to measure the collision probability. To get the collision probability for the entire primitive an independence assumption for each point along the path is made, so that the total probability $P(\gamma, C)$ is just the product of the non-collision probabilities $P(\neg C, p_i)$ at each sample point:

$$P(C, \gamma) = 1 - \prod_{i=1}^{n} \left[1 - P(C, p_i)\right]. \tag{3.5}$$

There is one issue with this method and it arises due to how the sensor perceives obstacles within the environment. Since sensors can only see the surface of an obstacle, any test point behind the obstacle may have a low collision probability, which is not necessarily realistic and may lead to the planner making a poor choice due to the inflated probability. This work makes the assumption that all space behind an obstacle surface is occupied. For a scenario where the robot sees the surface of a planar object, shown in Figure 3.4, any test point behind the wall will have the same collision probability as the closest point in front of the wall surface.

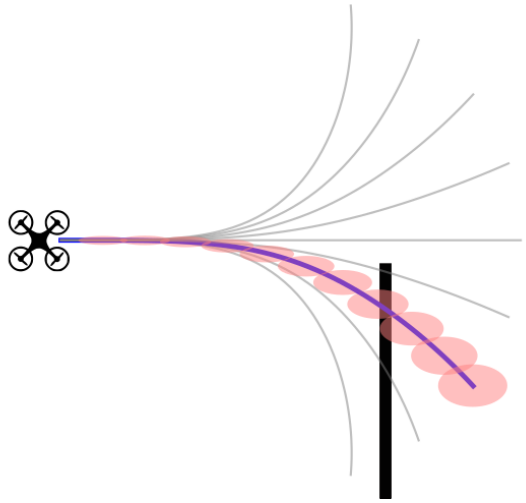


Figure 3.4: Probabilistic collision checking of motion primitives with a triple integrator model. The pink ellipses show the 1-σ of the Gaussian position covariance and the surface of a planar obstacle is shown in black.

3.2.3 Handling Non-Convex Obstacles

To be able to navigate in complex, realistic environments that contain non-convex obstacles it is insufficient to rely solely on an terminal goal position and select the primitive that minimizes the Euclidean distance. Due to the lack of knowledge of how the environment looks—the free and occupied space—the planner will only seek to maximize some myopic

reward making it prone to becoming entrapped in a dead end. Since the environment is only known through the incoming sensor information, it is not reasonable to assume that the planner should be able to completely avoid dead ends, as the robot may not know it exists until it is too late. Instead, the task of the planner should be to plan around non-convex obstacles and out of dead ends so that the robot does not get stuck.

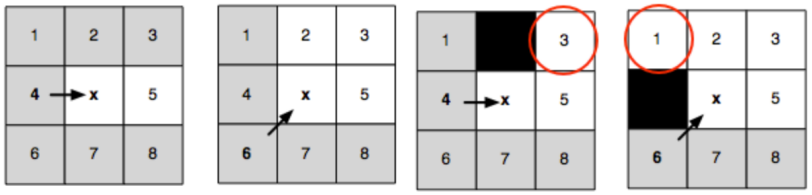
To solve this problem a separate geometric planner is run over a voxel map of the environment to provide additional guidance within the local planner to better inform primitive selection. The use of a geometric planner to handle non-convex obstacles is a common technique utilized in [11, 23, 41, 48, 49, 51]. The local planner does not follow this geometric path exactly; instead, it uses a loose integration that is designed to provide guidance of the direction the robot needs to travel that is factored into the primitive selection.

3.2.3.1 Jump Point Search

In the planning domain two or three-dimensional uniform-cost grids are a typical representation of the environment. Finding a geometric path then involves running a search algorithm, such as A*, which guarantees optimality and therefore the shortest path between two cells. A*, since it is not tailored for uniform-cost grids, suffers from some inefficiencies. This includes not considering path symmetry, which cause the search algorithm to explore more cells than are necessary. To improve performance, Jump Point Search (JPS) [17], an online symmetry breaking search algorithm, was developed to offer speed-ups by ”jumping over” many of the grid cells and only considering a reduced set. This method maintains all the completeness and optimality guarantees of A* for uniform-cost grids, but runs up to an order of magnitude faster.

The process of ”jumping over” cells in JPS is due to the symmetry breaking around the neighborhood of a node through two pruning rules: identifying natural and forced neighbors. The primary insight behind both rules is that neighboring nodes only need to be

explored if they may be reached optimally through the current node. The natural neighbors of a node are those where the optimal path must visit the current node assuming all adjacent nodes are free. However, in real path-finding scenarios obstacles may obstruct connections between nodes causing new optimal connections through the expanding node. These nodes that are added due to obstacles are forced neighbors. Examples of natural and forced neighbors may be found in Figure 3.5.



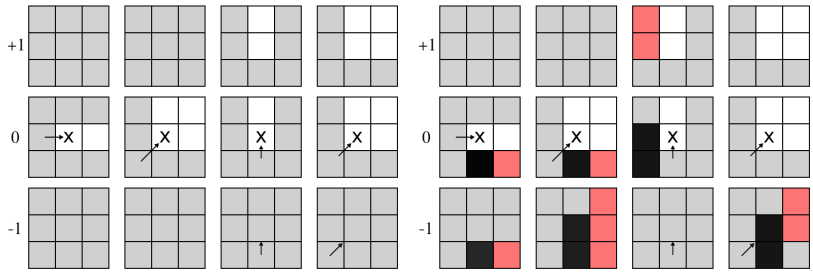
(a) Natural neighbors (b) Forced neighbors

Figure 3.5: Natural and forced neighbors for 2D JPS. The node to be expanded is labelled as **x** and the arrow indicates the direction of travel. Natural neighbors are shown in white, and forced neighbors are circled in red. Figure taken from [16].

During the search over the graph these pruning rules are performed recursively, jumping over nodes that do not meet the optimality criteria for being a forced neighbor of the expanded node. This recursion is performed until an obstacle or a *jump point* is reached. Jump points indicate locations where the search changes traversal direction, and any optimal path must go through these points. These jump points are the only nodes which need to be expanded, leading to performance gains (especially in maps with large open areas where path symmetry can be exploited to speed up the search).

Although the original work was for 2D grids, the method may be used for any uniform cost grid, including 3D voxel representations. The basic ideas that improve search performance still hold, find the natural and forced neighbors of the node being expanded and then search for jump points within the environment. Instead of an 8-connected grid, the 3D case uses 26-connected grid that corresponds to a $3 \times 3 \times 3$ voxel grid centered on the node to be expanded. For determining neighbors, 3D moves have highest priority, followed by

2D moves, and then 1D moves. For 2D JPS there are two kinds of moves—straight and diagonal—while the 3D case has 4 distinct types of moves: 1D, 2D in plane (stays in same z level), 2D out of plane (moves between z levels), and 3D. Examples of natural and forced neighbors for the 3D case may be seen in Figure 3.6. For more information on the theory and rules behind JPS please refer to [17].



(a) Natural neighbors (b) Forced neighbors

Figure 3.6: Natural and forced neighbors for 3D JPS. The node to be expanded is labelled as **x** and the arrow indicates the direction of travel. Natural neighbors are shown in white, and forced neighbors are shown in red. The 26-connected grid is sliced along the z-axis, with the top layer denoted by +1, the bottom by 1, and the current layer by 0.

3.2.4 Algorithm and Motion Primitive Selection

With all the individual components in place, they may now be combined to create the local planning framework. A detailed diagram of all the facets of the planner, its requisite inputs, and its output may be seen in Figure 3.7 and the algorithm is presented in Algorithm 1.

Before moving through the steps of the algorithm some notation that will be used through this section needs to be defined. At the onset of each replanning iteration a reference pose along the current trajectory is selected at a time a short duration in the future which is defined as $\mathbf{x}_{ref}$. The library of motion primitives is denoted by Γ and each primitive by γ_i, where i indicates the index of the primitive in the library. A position along primitive i at time $t \in [0, \tau]$ is given by $\mathbf{x}_i(t)$.

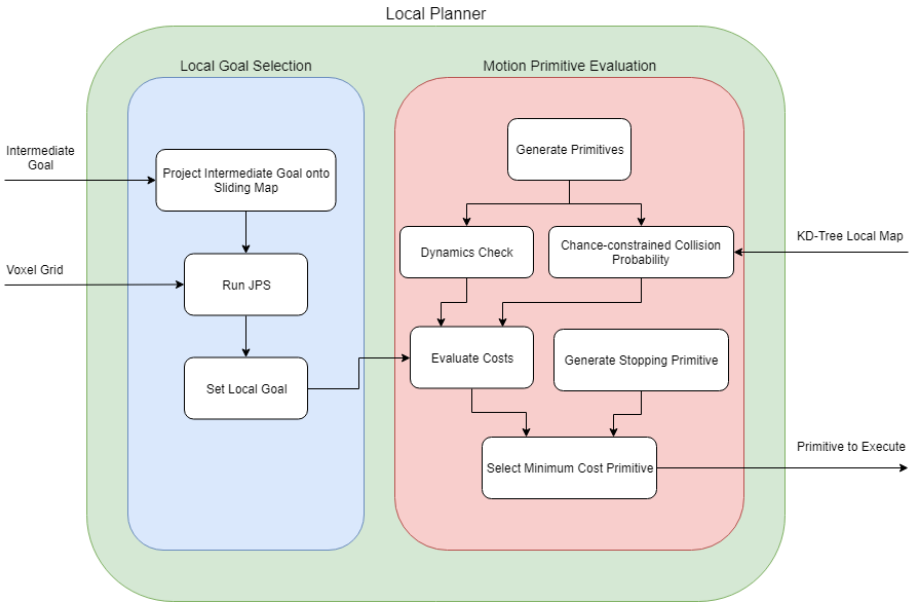


Figure 3.7: Local Planner Diagram

As input the local planner takes in an intermediate goal G_{int} that is provided by the global planner, and the maps that are built from the incoming depth sensor information. The local planner is divided into two main subsystems that are run in parallel: a local goal selection thread and motion primitive evaluation/selection. Local goal selection first takes G_{int} and projects that point onto the surface of the sliding voxel map to obtain a projected goal G_{proj} that is used as the goal for the JPS geometric planner. Each waypoint q_i along the JPS path is tested to see if it meets the criteria for local goal selection. The waypoint needs to meet one of following critera: 1) the angle between the robot heading and the vector $\overrightarrow{\mathbf{x}_{ref}q_i}$ being greater than 30° and $||\overrightarrow{\mathbf{x}_{ref}q_i}|| > 2$, 2) the path crossing over from one side of the robot to the other, or 3) $\angle q_{i-1}q_iq_{i+1} = 90^\circ$. This waypoint is then set as G_{local} and if no point along the path if selected, the local goal is set to G_{proj}. The local goal selection runs in a separate thread at 5Hz.

The other half of the local planner is the motion primitive evaluation module that has the primary task of selecting a primitive from the library for the robot to follow. At the

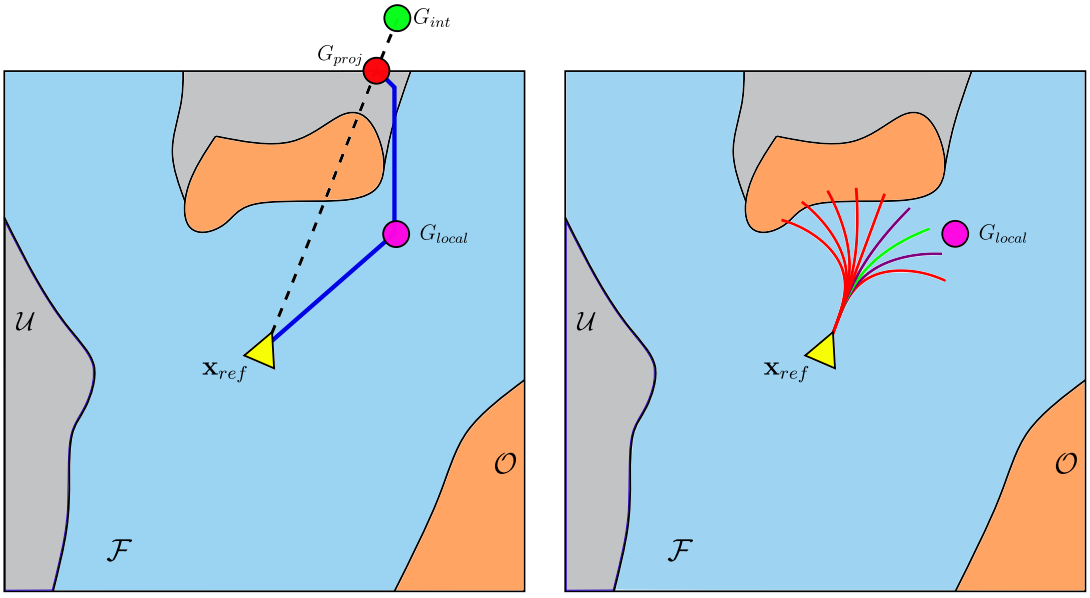
Algorithm 1 Local Planner

Input: G_{int}, $\mathbf{x}_{ref}$
function LOCALPLANNER( )
    REGENERATEPRIMITIVELIBRARY(v_{ref})
    $G_{proj} \leftarrow$ PROJECTGOAL($\mathbf{x}_{ref}$, G_{int})
    $G_{local} \leftarrow$GETLOCALGOAL($\mathbf{x}_{ref}$, G_{proj})
    for $\forall \gamma_i \in \Gamma$ **do**
        $P(C, \gamma_i) \leftarrow$ GETCOLLISIONPROBABILITY(γ_i)
        $c_i \leftarrow$ GETPRIMITIVECOST($P(C, \gamma_i)$, G_{local}, $\mathbf{x}_i(\tau)$)
    $j \leftarrow$ index of minimum cost primitive
    return γ_j

beginning of every replan iteration the motion primitive library Γ regenerates the motion primitives based upon the reference state of the robot and uses the reference velocity v_{ref} in the adaptive method detailed in Section 3.2.1. Afterwards, the collision probabilities $P(C, \gamma_i)$ are computed for each primitive using (3.2) and the kd-tree local map. Additionally, a dynamics check is performed by sampling points along the path to verify that the primitives remain within the robots's acceleration and jerk limits. If any primitive is dynamically infeasible, the collision probability for that primitive is set to 1. Furthermore, if the collision probability for any primitive is above 0.3 that primitive is deemed in collision and removed from consideration. Once the safety and feasible of all primitives has been assessed, the costs must be calculated. The total cost c_i (3.6) of primitive γ_i depends upon a weighted sum of a Euclidean progress metric c_{goal}, i.e., how close does the end position of the primitive come to reaching G_{local}, a collision probability cost $c_{collision}$, and a smoothness cost $c_{heading}$ to penalize large changes in ω values from the previously selected primitive. For this work $w_g = 1$, $w_c = 100$, and $w_h = 0.3$. Since a collision probability is calculated, the cost function inherently has a method for balancing progress towards the goal vs. safety in terms the robot's minimum distance to obstacles. Finally, the primitive with the lowest cost is selected for execution. If all primitives are infeasible or in collision, a stopping primitive that was calculated during the previous planning iteration is performed.

An example showcasing the local planner algorithm is shown in Figure 3.8.

$$
\begin{aligned}
c_i &= w_c c_{collision} + w_g c_{goal} + w_h c_{heading} \\
c_{collision} &= P(C, \gamma_i) \\
c_{goal} &= ||G_{local} - \mathbf{x}_i(\tau)|| \\
c_{heading} &= |\omega_i - \omega_{prev}|
\end{aligned}
\tag{3.6}
$$



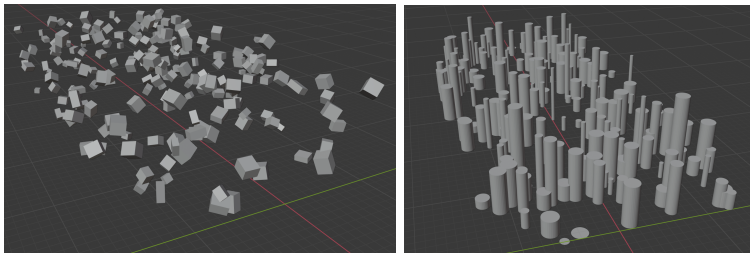
(a) Local goal selection

(b) Motion primitive evaluation

Figure 3.8: Example of local planner algorithm running on an example environment. The map depicts free space $\mathcal{F}$, occupied space $\mathcal{O}$, and unknown space $\mathcal{U}$. (left) The intermediate goal G_{int} is projected onto the sliding map to yield G_{proj}. A geometric path (■) between $\mathbf{x}_{ref}$ and G_{proj} is calculted by running JPS. A waypoint along this path is selected for the local goal G_{local}. (right) The motion primitive library is generated at the reference position, and the collision probability and cost for all primitives (■) are calculated. Primitives that are not dynamically feasible or that have a collision probability above a set threshold ■ are discarded from consideration. Finally, the primitive with the minimum cost (■) is selected for execution.

3.2.5 Evaluation of Local Planner

This sections presents an evaluation of the local planner performance and provides a rationale behind some of the design decisions for components of the planner. Simulation results are presented for three comparisons: probabilistic versus deterministic collision checking methods, the adaptive motion primitive generation against a larger fixed set, and the local planner against an offline optimal geometric planner. For the simulation tests, the depth sensor range is set to 10m and the motion primitive duration is 1s. For all comparisons, a total of 20 trials are performed in a set of randomly generated 3D block and forest environments that measure $60\text{m} \times 30\text{m} \times 10\text{m}$ with between 100 to 200 obstacles. Representative examples of these environments are shown in Figure 3.9.



(a) Random Block Environment (b) Random Forest Environment

Figure 3.9: Examples of 3D local planner testing environments

The first set of experiments is designed to evaluate the collision avoidance to verify that the robot remains safe during flight even at higher speeds. Three methods are tested, the first of which is a fixed collision radius while the other two are probabilistic. The former is a deterministic method where any point within the collision radius is deemed infeasible and is used as a baseline to compare against the two probabilistic approaches. The first probabilistic approach is from Du Toit et al. [9] and used in [11, 53] that calculates the collision probability through a small volume approximation. The second probabilistic approach, and the one utilized within this work, from Zhu et al. [54] is another chance

constrained strategy that claims to provide a tighter bound on the collision probability. Both probabilistic methods have their uncertainties scaled by the velocity of the robot, and for the tests a collision radius of 0.4m and a σ of 0.1 are used. To compare the collision checking methods, the success rate, minimum obstacle distance, and average solution time are calculated across the 20 trials. To determine the robustness over the flight regime, tests are performed at four different maximum allowable speeds $\{4, 6, 8, 10\}$ m/s. For all trials the motion primitive library contains 85 primitive and each trajectory has a duration of 1s. To isolate the collision checking and primitive selection components of the local planner and since the obstacles are all convex, the geometric JPS path is not used and the selection process relies entirely on the straight line goal projection. A screenshot from one of the trials is shown in Figure 3.10 and a trajectory through a random forest environment is shown in Figure 3.11. A comparison of the collision checking methods is provided in Table 3.1.

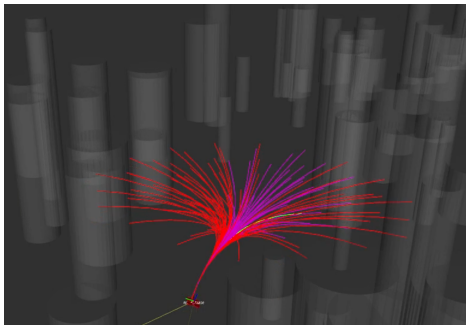
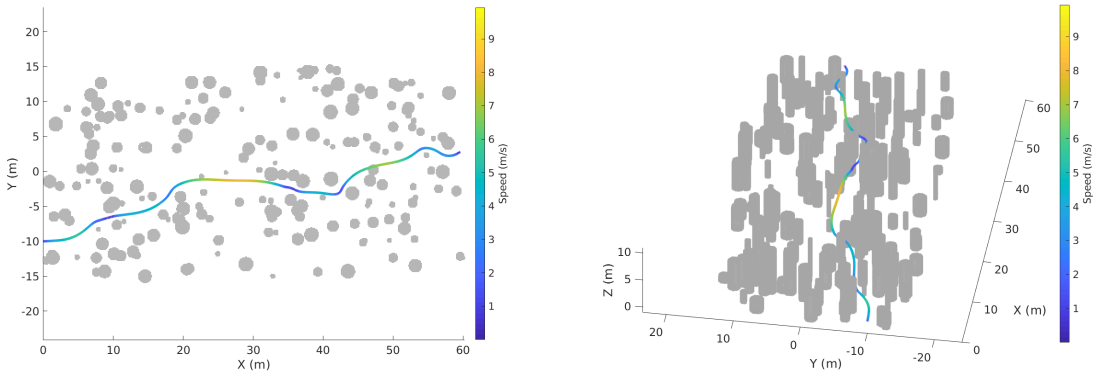


Figure 3.10: Screenshot showing the motion primitive library during operation in a random forest environment. Primitives shown in red are infeasible or in collision, and the selected primitive is shown in green.

Since collision avoidance methodologies are begin evaluated, the primary metric to determine and rank their effectiveness is the planner success rate. For these tests, a trial was deemed successful if the MAV reaches the goal and no obstacles are within the collision radius throughout the duration of the flight. For all flight speeds the method used in this work, from Zhu et al., had the highest success rates. In all the unsuccessful trials failure



(a) Top down view

(b) 3D view

Figure 3.11: Example trial for a random forest environment with a maximum allowable speed of 10 m/s. The trajectory is colored according to the speed of the robot, while obstacles are shown as a gray point cloud.

Max Allowable Speed	Collision Checking Method	Success Rate	Min. Obstacle Dist. (m)	Avg. Time (s)
4 m/s	Fixed Radius	0.9	0.553	17.2
	Du Toit	0.95	0.535	**16.9**
	Zhu	**1.0**	**0.790**	17.4
6 m/s	Fixed Radius	0.85	0.516	**12.2**
	Du Toit	0.8	0.477	12.4
	Zhu	**1.0**	**0.791**	13.0
8 m/s	Fixed Radius	**0.9**	0.520	**12.1**
	Du Toit	0.85	0.498	12.2
	Zhu	**0.9**	**0.819**	12.5
10 m/s	Fixed Radius	0.8	0.529	11.6
	Du Toit	0.8	0.493	**11.5**
	Zhu	**0.95**	**0.803**	12.3

Table 3.1: Comparison of collision checking methods (Du Toit et al. [9] and Zhu et al. [54]) with changing maximum allowable speed.

was caused by the robot getting stuck due to the collision avoidance parameters preventing the robot from navigating through narrow gaps in the obstacle-dense environments. This method also maintains the largest distance to obstacles during flight. These two facts really highlight its safety and robustness even for high-speed operation. There is a tradeoff, however, as the additional safety comes at the cost of a longer average time to reach the goal, but the stark advantages in safety and robustness far outweigh the longer flight times.

For the deterministic fixed radius case, there is little margin for error when maneuvering

around obstacles since the only data provided is a yes/no for whether the trajectory is in collision. Therefore, uncertainty in the state estimate or some tracking error could lead to a collision. Interestingly, during the comparison the probabilistic method from Du Toit et al. that was used in [11] for high-speed flight performed on par with the fixed radius method. To better understand why there was this performance gap, the two probabilistic methods were examined to see how the collision probabilities vary with velocity. Before moving into the results, it is helpful to look at the equations. The equation for the method from Zhu et al. is provided in (3.2) and the one from Du Toit et al. is given by

$$P(C) \approx V_r \times \frac{1}{\sqrt{\det(2\pi\Sigma_p)}} \times \exp\left[-\frac{1}{2}(p_i - p_j)^T \Sigma_p^{-1}(p_i - p_j)\right], \tag{3.7}$$

where V_r is the robot spherical volume, p_i is a position along the trajectory, p_j is an obstacle point, and Σ_p is the robot position covariance. A collision probability surface plot is constructed to investigate how the surface changes around an obstacle point with robot velocity. For visualization only two dimensions are considered. The plots are shown in Figure 3.12 with the Du Toit method in the left column and the Zhu method on the right. The probability at the collision radius is shown in green. For low velocities—low position uncertainty—the collision surface is cylindrical for both methods and then as the velocity increases the surface spreads out over a larger area. For the Zhu method, the collision probability at all points along the collision radius is 0.5 for all velocities. This sets a tight bound on the collision probability. This is not the case for the Du Toit method. At low velocities the collision probability is high only very close to the obstacle point, meaning the the collision probability is low (sometimes zero) within the collision radius. Since this method uses an inverse on the position covariance, the velocity of one axis affects the collision probability along the others, shown in the second row in Figure 3.12, which leads to large changes at different locations along the collision radius. It is evident that this method

does not provide a good bound on the actual collision probability, especially with larger objects and position uncertainties. This helps to explain the results shown in Table 3.1 for the planning failures and smaller distance to obstacles.

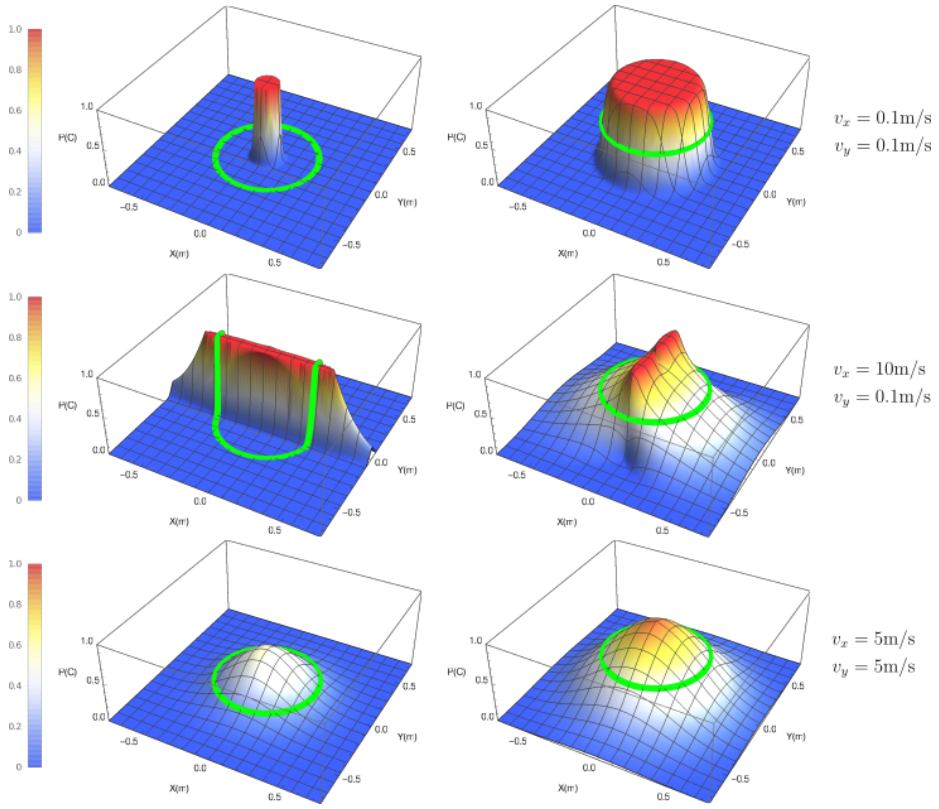


Figure 3.12: 2D comparison of the collision probability surfaces at different velocities between the Du Toit (left column) and Zhu (right column) methods. The obstacle point is centered at (0,0) and green path indicates the collision probability at the robot collision radius.

The last major design decision that needs to be evaluated is the choice to use a smaller number of adaptive primitives. This was chosen to reduce the computation time, but tests need to be performed to determine the impact on performance. 20 trials were run in the random block and forest environments using a motion primitive library of 85 primitives using the adaptive technique described in Section 3.2.1 and compared against a fixed library containing 370 primitives comprised of 10 different forward velocity discretizations. The results are shown in Table 3.2. Both have the same success rate with the larger primitive

library having a slightly lower average time to goal and minimum obstacle distance. The main difference between the two is the computation time, with the smaller adaptive set taking on average 8.97ms and the larger set of 370 primitives taking over twice as long with 20.23ms per planning iteration.

# of Primitives	Success Rate	Min. Obstacle Dist. (m)	Avg. Time (s)	Avg. Computation Time (ms)
85 Adaptive	0.95	0.803	12.31	8.97
370	0.95	0.786	12.16	20.23

Table 3.2: Comparison between using 85 adaptive velocity primitives vs. 370 fixed primitives

Another metric to compare the two is in terms of path smoothness. To measure this a FFT was run over the roll and pitch angles throughout the flight to determine the angular frequencies within the system (Figure 3.13). Ideally, the angular frequencies will be low, indicating a smoother flight that is beneficial for trajectory tracking and state estimation. Both the adaptive and the fixed primitive approaches have similar FFT responses with a majority of the frequencies below 20Hz. This result coupled with the other comparison results shows that the adaptive primitive generation method leads to similar flight performance for much lower computation.

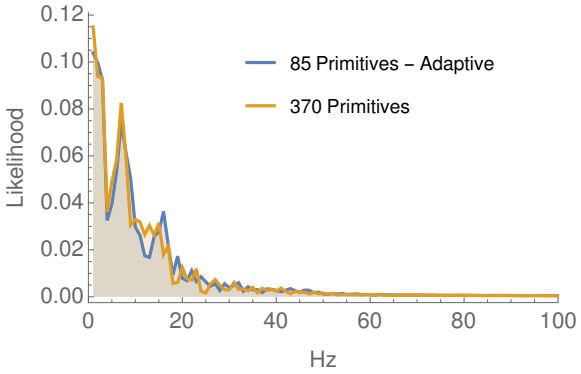


Figure 3.13: FFT comparison of the roll and pitch frequencies for the adaptive primitive approach vs. a larger fixed motion primitive library. Smoother flights will contain a greater likelihood of low frequencies.

For planning in unknown environments it is difficult to evaluate the planner path qual-

ity or optimality since the environment is revealed incrementally. However, if the environments used for testing contains only convex obstacles, a shortest path metric may be used to evaluate the path quality. An offline asymptotically optimal geometric planner RRT* with 2500 samples is used for path length comparison. Across all trials, the proposed local planning framework had a path that was on average only 1.7% longer than the optimal RRT* path.

Chapter 4

Global Sparse Topological Planner

When navigating in an environment that is not known *a priori*, mapping becomes a significant issue as the scale of the map is indeterminate at initialization. This could lead to the robot not having the requisite memory to store a record of the entire environment the robot has seen. Sliding map approaches limit the size of the map, and therefore the memory required, but compromise by forgetting parts of the environment that could potentially be of importance later. Another issue with global mapping strategies is that often times they do not take into account planning efficiency. For example, search-based planning such as A* on a typical occupancy map representation could take minutes to solve, while sampling-based methods often struggle with finding paths through doorways or other thin passageways. For efficient planning, however, it is better to represent the environment in a manner that informs the robot about how areas are connected rather than just differentiating free and occupied space.

To overcome these issues, this work leverages topological mapping and global planning. The key idea behind topological graphs is that they represent environment connectivity and traversability through a small set of nodes, thus allowing for very fast planing through the graph. Thrun [46] are one of first works to showcase topological planning in robotics

by constructing a 2D topological graph using a Voronoi diagram of an occupancy grid. Their key idea for partitioning the space is to find "critical points" that section of the grid. Another way to generate the topology of the space is through a convex decomposition of free space, as is done in IRIS [7], that decomposes the space into connected polygonal regions. The main drawback of the approach is high computation times for 3D environment, which is not suitable for real-time flight. Liu et al. [25] enable online usage by leveraging a geometric path to seed the decomposition, but this approach limits the planning to a single homotopy that does not store information about alternative pathways. Cover et al. [6] creates a distance field around obstacles creating a sparsely connected graph to quickly find paths to a set goal, but once again, it retains no information about the environment it has seen. Blochliger et al. [3] creates a topological graph by generating convex free space clusters from a feature-based SLAM system, while Oleynikova et al. [34] extracts a 3D Generalized Vornoi Diagram from an ESDF of the environment to generate a skeleton structure representing the topological information present within the map. Both of these methods assume an initial exploratory phase to generate a map that may be used to revisit locations and are not designed for online usage.

The proposed global planning strategy creates a sparse topological graph of the environment online that can identify possible alternative pathways to goal and allows for efficient planning to return to the starting location. Unlike works that only use a sliding voxel map—only has short-term memory of environment—and will fail when no available path lies within the boundaries of the sliding map, our approach does not discard information [41, 48, 49]. This allows the global planner to revisit parts of the environment that are no longer contained within the sliding map and find paths that were not chosen on the first pass but may lead to the desired goal. Due to the sparse nature of the graph, the memory footprint is very small and the planning on the graph may be performed very quickly.

The main objective of the global planner is to select an intermediate goal for use within

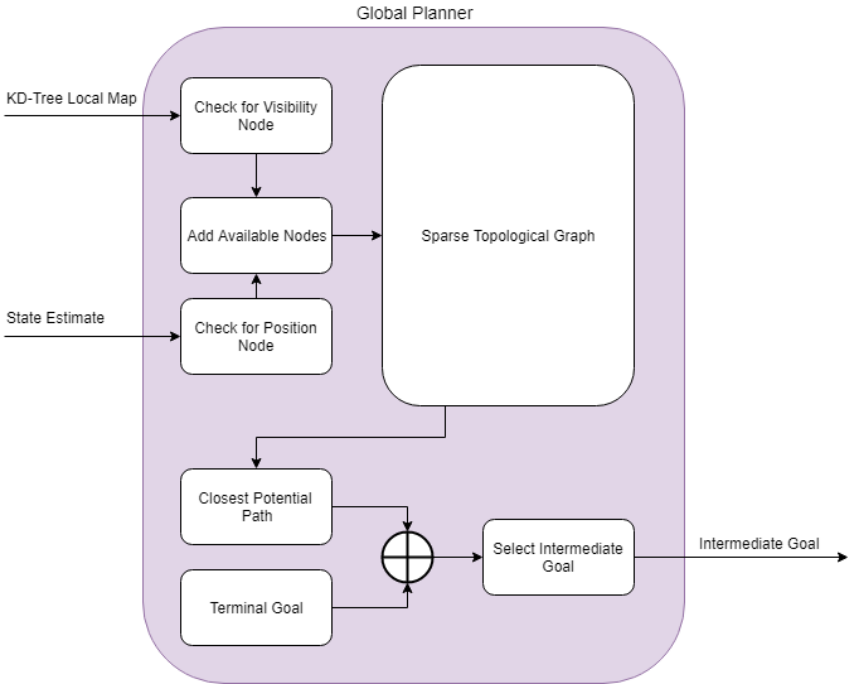


Figure 4.1: Global Planner Diagram

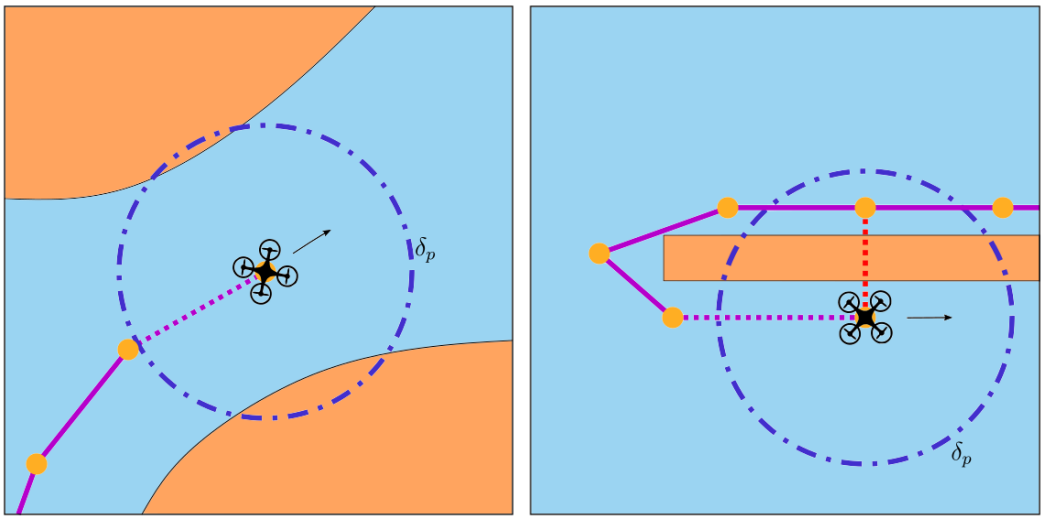
the local planner. During operation a sparse graph is created by combining a history of the robot's path with a set of *visibility* nodes that aim to identify alternative pathways that the robot did not take, e.g., doorways the robot passed. If the robot reaches a dead end along its current path, the sparse graph can be searched to find a path to an alternate path that may lie outside of the sliding map the robot has of the environment, allowing for backtracking over long distances in situations where competing approaches fail. A diagram of the global planner is shown in Figure 4.1. The rest of this section describes how nodes are added to the sparse graph, the method for searching over the graph, and the integration of the global planner with the local planner from Section 3.2. Finally, results are shown for the full planning strategy.

4.1 Adding Nodes to Graph

Since the sparse graph must be used online for the global planner, it is designed to allow for very fast updates. The graph is created through combining a history of the robot's positions during the course of operation and by identifying potential pathways that the robot has passed but not taken that could be used if a dead end is encountered. This generates a rough topology of the environment based on where the robot has already traversed and locations where traversal is possible.

The first type of node that the graph contains are *position nodes* that keep a time history of the robot's path. If the robot has traveled a distance greater than a set threshold δ_p from the last node in the graph, a new node is added to the sparse graph $\mathcal{G}_s$ (Figure 4.2a). This type of node creates the skeleton of the sparse graph and is the primary type of node added. To quickly determine the distance between the current position and nodes within the graph, a kd-tree is maintained of all the node positions within the sparse graph. Position nodes are connected to the closest node in the sparse graph to the robot's current position. It is possible that during operation the closest node is blocked from the current location by an obstacle, as shown in Figure 4.2b. To ensure no connections are made that pass through an obstacle, a collision check is performed at a set of sampled locations along a straight-line between the node and current position before any node is added. In the case of the example presented in Figure 4.2b, the collision check fails and no connection is made to the closest node. Since the distance to the previous node is above the distance threshold, a connection is made that reflects the travel path of the robot. Every time a new position node is added to the graph, the previous node n_p is set to the current position. These position nodes are very helpful in guiding the robot if it needs to backtrack through a previously explored section of the environment.

The other and more important type of node is the visibility node that is added at critical



(a) (b)

Figure 4.2: (a) A node is added to the sparse graph if the robot has traveled a distance greater than a threshold δ_p from the previous node. Nodes in the graph are shown in yellow and connections in purple. The newly added connection is represented by a dashed line. (b) The straight-line connection between the robot and the closest node in the graph is blocked by an obstacle. A collision check is performed and no graph connection is made (shown in red). Since the robot has moved a distance greater than δ_p from the previously added node, a new position node is added and connected to the graph.

points within the environment where potential pathways are detected, such as doorways that the robot has not yet entered. To identify the points, a set of test nodes $\mathbf{t}_n$ are created that encircle the robot. The test nodes are defined in the body frame of the robot, so that their positions relative to the robot are fixed. To get them into the world frame for use, the test nodes are translated to the current reference position and rotated to the current yaw as provided by $\mathbf{x}_{ref}$. Each test node has a set of neighbors that are used to check for its visibility. In this work eight test nodes were used and each had four neighbors, two on either side. Points along the straight-line connections between the test node and its neighbors are checked for collision with the kd-tree local map. If more than one connection is blocked and the test node is visible from the current robot position it is added to the sparse graph as a visibility node. A visibility node has a connection to the goal node n_g and a position

node in the graph. If a visibility node is added, a position node is added at the current robot location to connect the visibility node into the graph. The process for adding a visibility node is shown in Figure 4.3.

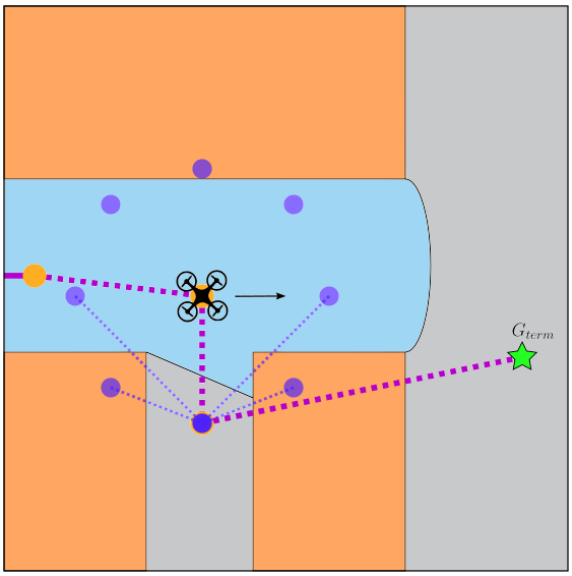


Figure 4.3: The visibility between test nodes (●) that encircle the robot are checked to see if a *visibility node* should be added. If connections to a test node are in collision and the node is visible to the current robot position, it is added to the sparse graph with a connection to the goal. These visibility nodes inform the global planner of possible paths to goal that the robot has not taken.

To add a node into the sparse graph three quantities are needed: the id of the node(s) to which it connects, the Euclidean distance between the nodes (used as the cost in an A* search), and whether the edge is traversable, i.e., has the robot physically moved between the two nodes. An example of a non-traversable connection is the one made between a visibility node and the goal node. This quantity is used within the A* search for the return home functionality. All of this information is kept in a series of adjacency vectors that provide the connectivity of the nodes within the graph and allows for efficient modification and search. One modification case is when the robot passes over a visibility node and continues down that path—creating a new position node—it is no longer an alternative path so the goal connection must be removed to reflect that new information about the topology

of the environment. The process for adding nodes to the sparse graph is summarized in Algorithm 2.

Algorithm 2 Adding nodes to the sparse graph

Input: $\mathbf{x}_{ref}$, $\mathcal{G}_s$, n_{prev}, d_{prev}
function CHECKFORNEWNODES( )
    $\mathbf{t}_n \leftarrow$ UPDATETESTNODES($\mathbf{x}_{ref}$)
    visNode $\leftarrow$ CHECKNODEVISIBILITY($\mathbf{t}_n$)
    posNode $\leftarrow$ CHECKFORPOSITIONNODE(visNode)
    if visNode or posNode **then**
        update sparse graph kd-tree
function CHECKNODEVISIBILITY($\mathbf{t}_n$)
    addNode = false
    for $t_n^i \in \mathbf{t}_n$ **do**
        $q_{vis} = 0$
        for each neighbor of t_n^i **do**
            if connection not visible **then**
                $q_{vis} \leftarrow q_{vis} + 1$
        if $q_{vis} > 1$ and isVisible(t_n^i) **then**
            $\mathcal{G}_s \leftarrow$ add visibility node and position node to sparse graph
            addNode = true
    return addNode
function CHECKFORPOSITIONNODE(visNode)
    if visNode **then**
        return false
    $n_{close}, d_{close} \leftarrow$ find closest node and its distance to current position
    if $d_{close} < \delta_l$ and $d_{prev} < \delta_l$ **then**
        $\mathcal{G}_s \leftarrow$ create loop connection between n_{close} and n_{prev}
        CHECKFORGOALCONNECTION(n_{close})
        return true
    if $d_{close} > \delta_p$ and isVisible(n_{close}) **then**
        $\mathcal{G}_s \leftarrow$ add new position node and set as n_{prev} for next iteration
        CHECKFORGOALCONNECTION(n_{close})
        return true
    if $d_{prev} > \delta_p$ **then**
        $\mathcal{G}_s \leftarrow$ add new position node and set as n_{prev} for next iteration
        return true
    return false

4.2 Searching the Graph: Alternative Paths and Return to Home

With the sparse graph built, it is time to discuss how to search through the graph to return useful information that can be used for the global planner. Searches are performed on the sparse graph either to find an alternative, unexplored path or return home to the original starting location. By planning on the sparse toplogical graph, both of these searches can be done very efficiently, and more importantly, the waypoints on the returned path can be located outside of the sliding map. Therefore, a notion of longer term memory is added and exploited to aid the robot in navigating large, complex environments.

The first type of search is used for the primary mission of planning through the unknown environment to a desired goal location. During the course of operation, the robot may encounter a dead end where no path is available within the sliding map. Earlier in the flight, however, the robot may have passed by a doorway that could lead to goal if the robot were to explore this route. Since these types of points have already been identified as *visibility nodes* within the sparse graph structure, they can be found simply by searching for nodes that have connections to the goal node n_g. This is done through a standard A* search over a bidirectional graph to find the shortest path from the current position to goal. The returned path can then be used by the global planner to instruct the robot how to move back through the sparse graph to reach this point. An example of how the alternative path functionality works can be seen in Figure 4.4. This process can continue until either the robot reaches the desired goal location or no alternative pathways are available, indicating the robot should enter an exploratory phase to try to find another option to try. If exploration is required, the sparse graph could be used to inform decisions to limit areas that the robot has already visited. This line of work is not explored further in this thesis, but the idea is discussed in more detail as future work in Section 5.2.

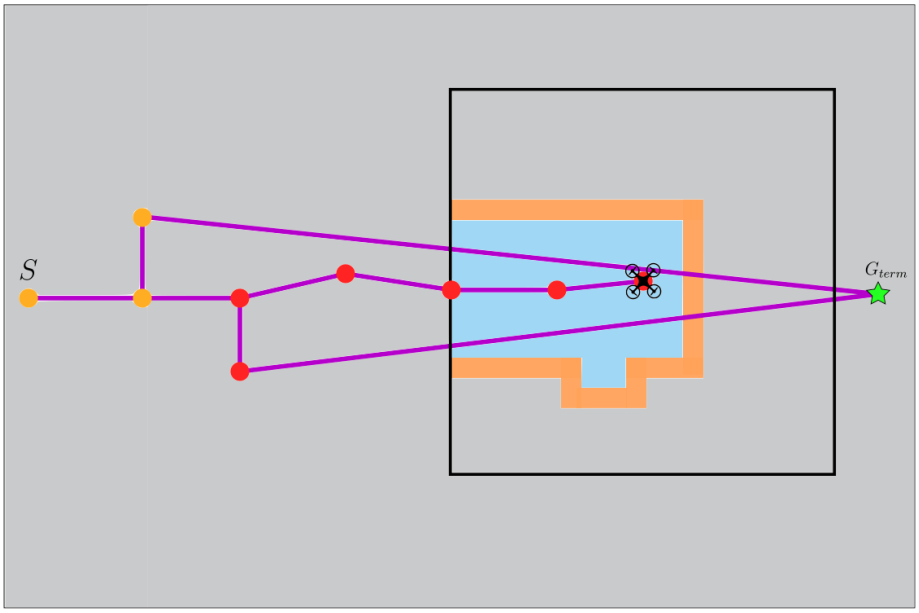


Figure 4.4: If the robot encounters a dead end, the sparse graph may be searched to identify an alternative path to goal (shown in red). This allows the robot to remember and plan towards parts of the environment are no longer within its sliding map (area within black boundaries).

After the robot has completed the mission and reached the desired goal location, the planning problem is still not finished. Robots are not one-time use, so they must be recovered, and ideally after completing their mission they should return to the starting location. Most works in the area of high-speed navigation through unknown environments ignore this aspect, and presumably just set a new desired goal location and repeat the process to the starting location. However, this type of method does not use any of the information of how the robot got the goal in the first place. Conversely, the proposed methodology exploits the travel history of the robot to accurately return to the starting position.

The search algorithm for the return to home functionality is very similar to that for finding the alternative paths. An A* search is run to find the shortest path between the current position and the start location S (start node n_s). The main difference is that the path must consist entirely of nodes that the robot has moved through to ensure that the path is through free space, i.e., the route can actually be followed. If visibility nodes

are unused within the graph, their connection to goal will still exist and typically lead to shorter overall path lengths. By not considering the traversability of an edge, the graph search would return a path through these untested connections and may not actually lead the robot back to its original location. This is the reason the boolean flag is added for each edge in the sparse graph during its construction. To rectify this issue, only edges that meet the traversal criteria are used within the A* search. The returned path is then guaranteed to be the route that the robot took to reach the goal. An example of a return to home graph search may be seen in Figure 4.11.

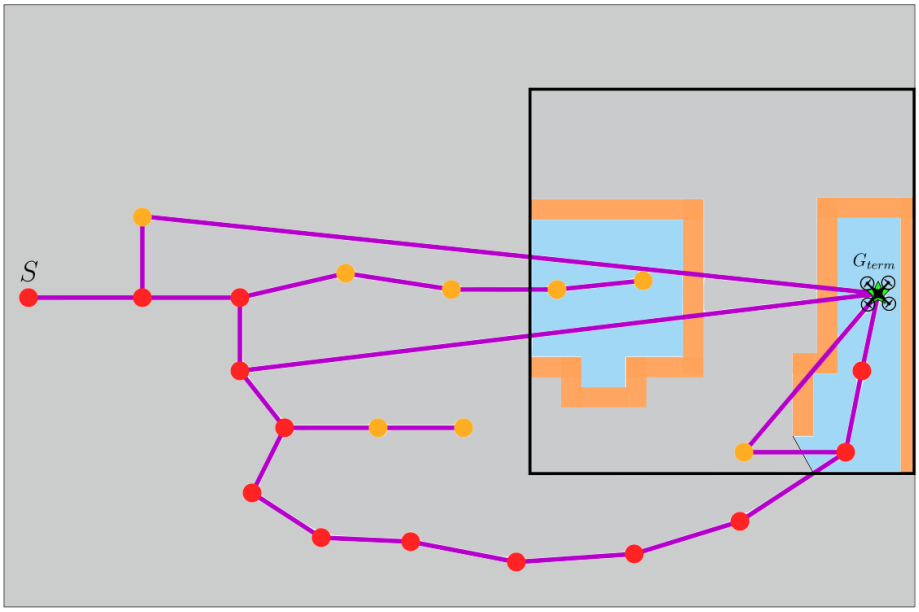


Figure 4.5: After the robot has completed its task, the sparse graph can be searched to find a path from the current location to the original starting position S. During the search, only traversable connections are considered and not connections from unused visibility nodes to goal. This ensures the return path consists entirely of nodes that the robot has previously moved through.

By introducing this longer term memory of the environment through the global sparse topological graph, the proposed method has the ability to revisit locations that the robot has previously seen using efficient graph search techniques. The robot can then find an alternative, unexplored path or return home to its original starting location, and is no longer

limited to plan only within the boundaries and computational limits of a sliding map. The algorithms for these two functions can be found in Algorithm 3.

Algorithm 3 Searching the sparse graph

```
1: Input: x_ref, G_s, n_g, n_s
2: function FINDALTERNATEPATH( )
3:     if size(neighbors of n_g) = 0 then
4:         return ∅
5:     else
6:         ξ ← find A* shortest path from n_prev to n_g
7:         return ξ
8: function FINDPATHHOME( )
9:     ξ ← find A* shortest traversable path from n_prev to n_s, i.e., ignores connections to
       n_g from visibility nodes
10:    return ξ
```

4.3 Integration with Local Planner

With the introduction of the sparse topological graph the robot now has the ability to construct and plan upon a global representation of the environment, opening possibilities for the global planner to solve problems that were previously not achievable. The part that is still left is the integration of this global planner with the local planner described in Section 3.2 to construct the full planning framework. For review, a system diagram for the entire planning architecture is shown in Figure 3.1.

The objective of the global planner is to provide an intermediate goal at each replanning iteration to guide the local planner and its trajectory selection. There are two options from which the global planner may choose: the terminal goal G_{term} or a waypoint along a path ξ returned from searching the sparse graph. On initialization, the intermediate goal G_{int} is set to the terminal goal location and this only changes if an alternative path search is required. There are two reasons why an alternative path search would be triggered. The

first is that no JPS geometric path can be found for an extended period (a threshold of two seconds was used) of time within the local planner. This indicates that the sliding map contains no available path, the local planner is stuck, and other options are required. The other trigger condition is if the global planner detects backtracking through the sparse graph $\mathcal{G}_s$. Backtracking is defined as revisiting a node within the sparse graph that is not a visibility node. What this comes down to is checking whether the id of the node the robot has revisited is lower than the previous node id, since node ids will always increase as the robot advances towards goal. If either condition is met, the global planner searches for an alternative path to use as intermediate goals.

After the search, the intermediate goal is set to the closest node in ξ to the robot's current location. Once the robot moves within a certain distance ϵ to the waypoint, the intermediate goal shifts to the next waypoint in the path. This continues until all waypoints in the path have been used and the intermediate goal returns to being the terminal goal position. While the robot is following the alternative pathway, the sparse graph does not check for the addition of any new nodes as the route has already been added to the topological graph. A similar intermediate goal selection process is used for the robot's return to home, with the G_{term} being updated to the starting position S. Likewise, no new nodes are added while the robot is returning to home. Details for the integration of the local and global planners can be found in Algorithm 4.

Algorithm 4 Integration of local and global planners

Input: $\mathbf{x}_{ref}$, $\mathcal{G}_s$, G_{term}
function PLAN( )
    $G_{int} \leftarrow G_{term}$
    $\xi \leftarrow \emptyset$
    alt_path $\leftarrow$ false
    $j \leftarrow -1$
    while $\neg$isGoal$(\mathbf{x}_{ref}, G_{term})$ **do**
        $G_{int} \leftarrow$ GETINTERMEDIATEGOAL$(\xi, \mathbf{x}_{ref}, j)$
        LOCALPLANNER( )
        if $\neg$ follow_path **then**
            CHECKFORNEWNODES( )
        if no JPS path found or backtracking detected in $\mathcal{G}_s$ **then**
            $\xi \leftarrow$ FINDALTERNATEPATH( )
            $j \leftarrow$ length$(\xi) - 2$
            alt_path $\leftarrow$ true
function GETINTERMEDIATEGOAL($\xi, \mathbf{x}_{ref}, j$)
    if alt_path = false **then**
        return G_{term}
    else if $j = 0$ **then**
        alt_path = false
        return G_{term}
    else
        if $||\mathbf{x}_{ref} - \xi_j|| < \epsilon$ and $j > 0$ **then**
            $j \leftarrow j - 1$
        return ξ_j

4.4 Results

To evaluate the performance of the integrated planning architecture, simulation experiments are run in large-scale, complex environments that feature dead ends that force the robot to reroute in order to reach the goal. These environments are designed to ensure that the global planner must search through the sparse graph it is creating online to find solutions to the goal. For all tests the robot has a maximum allowable velocity of 10m/s, a sensing range of 10m, and a 20m $\times$ 20m $\times$ 6m sliding voxel map with a resolution of 0.2m for use with the geometric planner. The motion primitive library contains 85 adaptive motion primitives

and a trajectory duration of 1s.

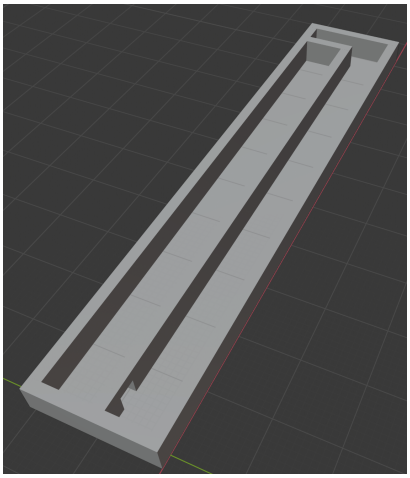
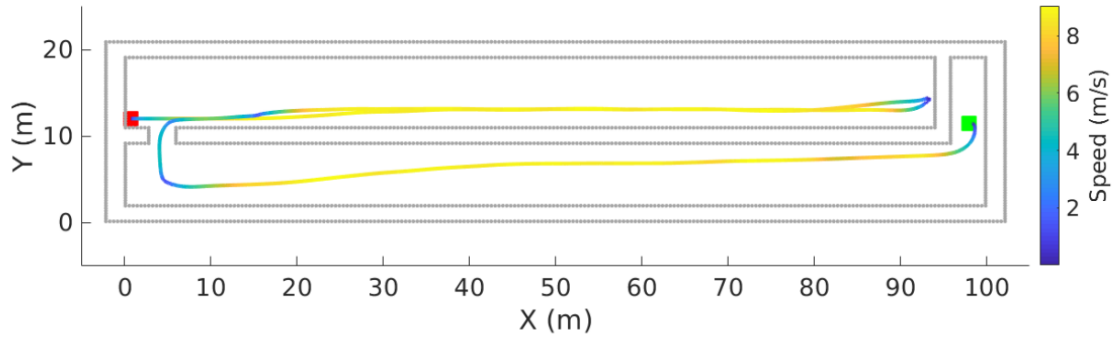


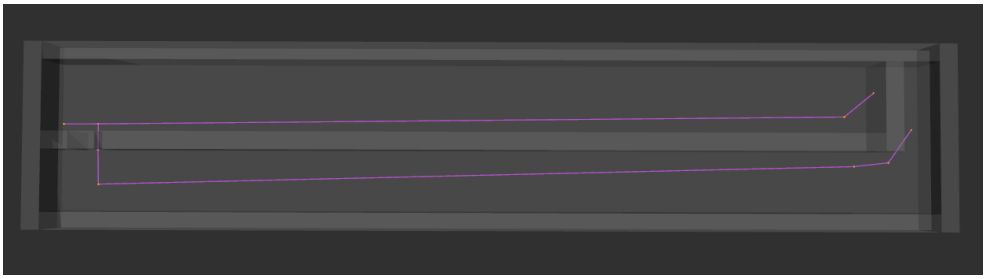
Figure 4.6: Corridor environment containing a small doorway connecting the two sides

The first environment (shown in Figure 4.6) is a corridor that has two sides and contains a small doorway that allows for passage between. The environment measures 100m $\times$ 20m $\times$ 6m with the doorway 5m from one end. The desired goal is at the other end in a separate corridor, meaning that the robot must circle back to the doorway to reach the goal (can be seen in Figure 4.7a). Due to the size of the environment and the placement of the doorway relative to the desired goal, relying just on a sliding map around the robot—even the 60m $\times$ 60m one used in [41]—will not contain the information about the doorway. This means that these competing approaches will fail even in this simplistic-looking environment. Using the proposed planning framework, the robot detects the dead end within the first corridor, searches its sparse global map, and then plans back towards its starting location to travel to the second corridor and then reaches the desired goal. The trajectory the robot takes to goal and the created global sparse graph are shown in Figure 4.7. During operation the robot reaches a maximum speed of 9m/s during the long open areas of the corridor and decreases its speed to make turns and accurately maneuver through the narrow doorway.

The second test is run in a multi corridor (100m $\times$ 35m $\times$ 8m) environment that has



(a)



(b)

Figure 4.7: (a) Trajectory through the corridor environment from the start location ( ) to the goal ( ). A maximum speed of 9 m/s was reached during the flight. (b) The resultant global sparse graph created during planning. Nodes are shown in orange and edges in purple.

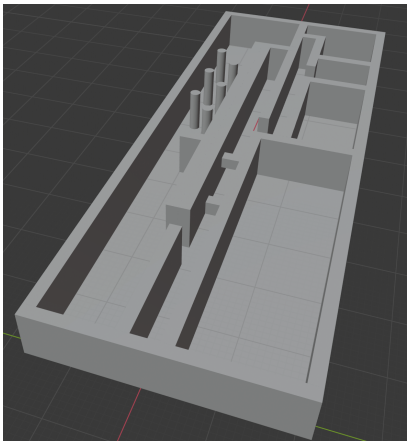
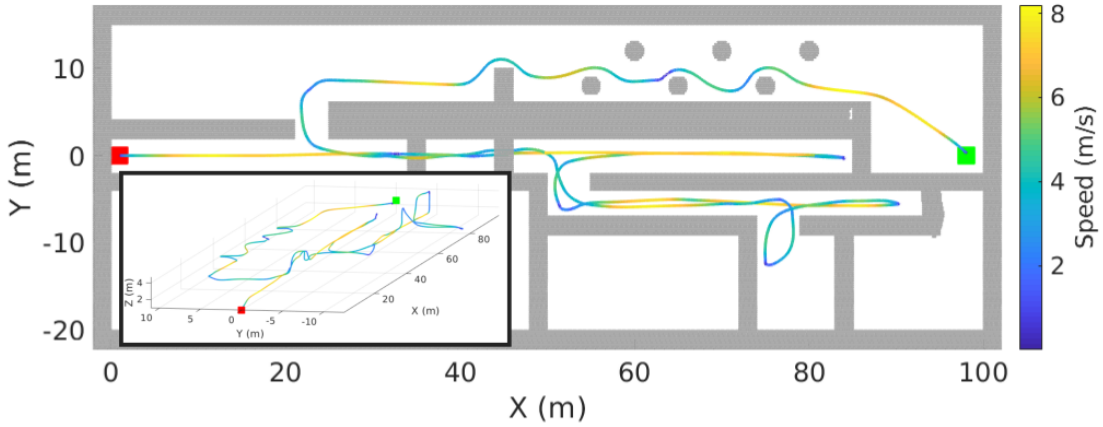


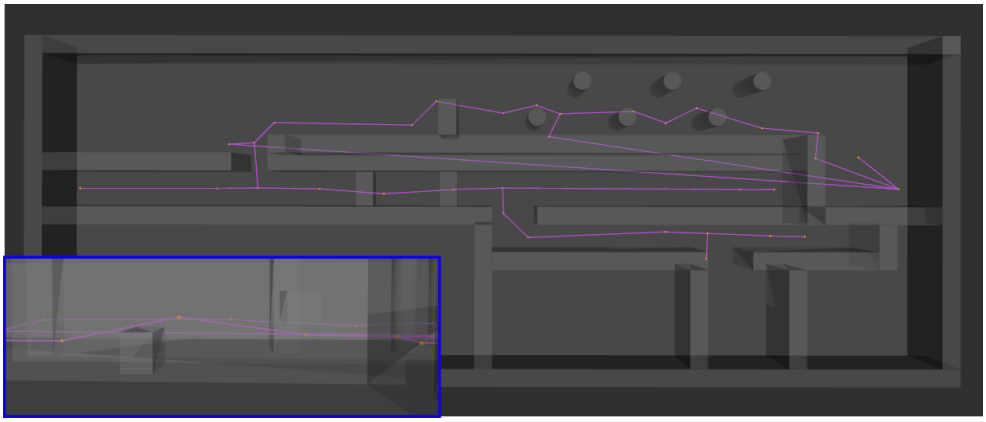
Figure 4.8: Multi corridor environment

greater complexity. It consists of more dead ends, obstacles that force the robot to change altitudes, and a set of pillars to really test both the local and the global planning components. An image of the multi corridor environment is provided in Figure 4.8. The global

sparse graph is searched on three separate occasions to identify potential pathways before the robot finally reaches the desired goal. The robot trajectory and the global sparse graph are shown in Figure 4.9. The velocity distribution for the flight can be found in Figure 4.10. The average speed of the robot is 4.13m/s for the flight and a maximum speed of 8m/s is achieved.



(a)



(b)

Figure 4.9: (a) Trajectory through the multi corridor environment colored according to flight speed. The inset shows a 3D view of the trajectory as it maneuvers above and below obstacles within the environment. (b) The global sparse graph after the robot reached the desired goal with an inset to highlight the 3D geometry of the graph.

Since the history of the robot's path underlays the global sparse graph, it may be uti-

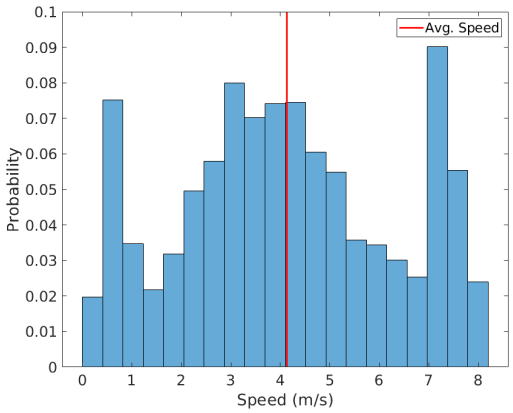


Figure 4.10: Flight speed distribution for the multi corridor environment. The average speed during the trial was 4.13 m/s and a maximum speed of 8 m/s was achieved.

lized to accurately return to the starting location. To test the benefits, a comparison was performed where the robot was tasked with planning back to the start location from the goal in the multi corridor environment with and without using the sparse graph. When the sparse graph is used, the robot can safely and accurately return to the starting location along its previously explored route. When the sparse is not used, the robot takes a longer initial path since it is unaware of the obstacles, gets stuck in a previously unseen section of the environment, and fails to reach the home position. This demonstrates the effectiveness of the proposed approach for quickly enabling the robot to move back to the original deployment site. A trajectory comparison is provided in Figure 4.11.

Since a major aspect of the proposed approach is its computational and memory efficiency, timing results and memory usage were gathered during the trial for the multi corridor environment with the data presented in Table 4.1. Each iteration through the local planner takes on average 14ms (faster than sensor rate), while additions to the sparse graph can be computed in 0.5ms. If a global voxel map were stored for the duration of the flight to be used for global planning, the memory required would be 27.32MB. Contrast this with the proposed global sparse graph that requires only 0.00128MB, or over 21,000 times less

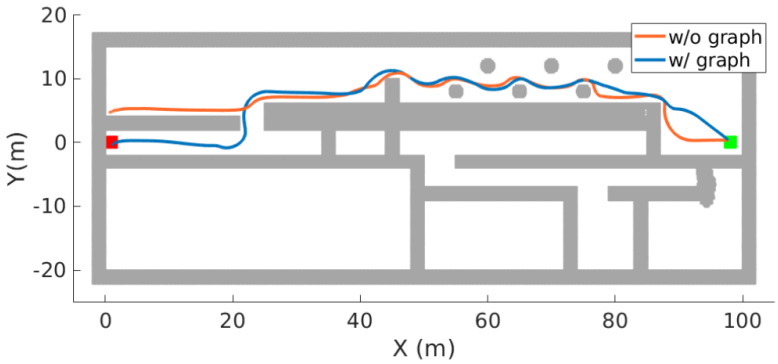


Figure 4.11: Comparison of flight trajectories during a return to home maneuver with and without using the sparse graph.

memory. The average search time through the sparse graph is less than 1ms; however, the search time is dependent upon the graph size and its connectivity. More importantly, planning over the sparse graph will be orders of magnitude faster than planning over the voxel grid itself, as shown in [34]. These numbers indicate that the proposed approach is very efficient and takes into account the practical considerations of computation and memory that are requisite for real-time operation, while still generating safe, reactive trajectories even at high-speeds.

Component	**Avg. Time (s)**	**Memory Usage (MB)**
Collision Checking (Single Primitive)	0.00016 ± 0.00046	-
Primitive Selection	0.0141 ± 0.0085	-
JPS	0.0058 ± 0.0083	-
Sparse Graph - Add Node	0.00054 ± 0.0009	-
Sparse Graph - Search	$<$ 1ms	-
Sparse Graph	-	0.00128
Global Voxel Map	-	27.32

Table 4.1: Timing results and memory usage measured during multi corridor environment trial. Note: The search times for the sparse graph are heavily dependent upon the size and connectivity of the underlying graph, so no exact times are provided. The search times are orders of magnitudes faster than searching directly over a global voxel grid and are typically under 1ms. Refer to [34] for a comparison of sparse topological planning times to A* and RRT* approaches.

Chapter 5

Conclusions and Future Work

5.1 Summary

This thesis presented a computationally-efficient planning architecture for safe high-speed operation in unknown environments that incorporates a notion a longer term memory into the planner to allow for exploring alternative pathways found within the environment and to accurately return to the starting location. Chapter 3.2 presented the system architecture, consisting of a hierarchical planning module (local and global) and how these components interact with each other and the mapping and control systems. Afterwards, a breakdown of the local planner and its constituent components was presented. To generate trajectories, a motion primitive library is utilized that implements an adaptive technique for discretizing the space of final velocities while maintaining computational speed. For safety and robustness the local planner makes use of a probabilistic collision avoidance strategy. The final component is the addition of a geometric planner to handle non-convex obstacles within the environment. Results are presented to provide a rationale behind the design decisions for the local planner, and highlight its computational efficiency and safety.

Chapter 4 presented the global planning framework that creates and plans over a sparse

topological graph online. The graph consists of a spatial history of the robot's path through the environment plus a set of visibility nodes that encode unexplored possible pathways to goal. Afterwards, the integration of the local and global planners is provided. Simulation results demonstrate the effectiveness of the proposed planning architecture across a set of large-scale complex environments by identifying dead ends and rerouting through the environment due to the global planner. These results highlight the applicability of the planning methodology in scenarios where more myopic sliding map approaches would fail.

5.2 Future Work

Future work first off includes shifting the planning architecture detailed in this thesis to hardware to test its performance on a real system. This would further stress test the computational and memory demands of the planner for flight over large-scale environments on the onboard computer while other systems such as state estimation and control are running.

One limitation with using a geometric planner to provide guidance is that it has no notion of the dynamics of the robot. This can lead to the local goals that rapidly change direction as new information is unveiled within the environment. Worse, these rapid changes could induce instability and affect flight performance. Tordesillas et al. [48] seeks to mitigate this issue by implementing a multi-fidelity approach that adds more kinodynamic information into the cost function to favor geometric paths in the direction of motion. For future work, a fast kinodynamic planner will be tested in place of the JPS component. One possible method is to use the geometric path returned from [8] that incorporates dynamics. By performing an offline reachability analysis, this method allows for quick computations using a stored data structure.

Most of the planning methodologies that seek to operate in unknown environments

perform a straight-line projection of the desired terminal goal onto some sort of sliding map and use this to compute the local plan. This is a logical step because without knowledge of the environment outside of the sensor range, the best decision is to just try to move towards the goal. Problems arise when the robot must move away from the goal direction in an effort to reach the goal. This may be seen in maze environments where the robot must move down unexplored corridors that force the robot to turn its back on the goal. In these scenarios, a straight-line projection is not sufficient and will force the robot to move only towards the goal. One possible avenue to deal with this problem is to add a notion of frontiers into the planner and making sure that the projected goal lies within a frontier. This will a slight exploratory component to the planner so the robot can plan trajectories that try to move towards the closest unexplored space towards the goal.

To develop a complete planning strategy—one that can deal with all types of scenarios without getting stuck—needs to incorporate some type of exploratory strategy if the robot reaches a point where no available pathways exist. For the proposed work, that situation occurs when the robot encounters a dead end and the sparse graph contains no connections to the goal node. The sparse graph may also benefit planning during exploration by providing a fast, efficient method to pathways to parts of the map that have been visited or to return to the starting location as opposed to planning over the occupancy map that it is producing. This incorporation and interplay with an exploratory planner is an interesting line of research that can extend the work within this thesis.

5.2.1 Extension to Dynamic Obstacles

Future work will focus on incorporating dynamic obstacles into the planning framework whilst maintaining safety. In search and rescue scenarios the robot may be operating in close proximity with humans or may encounter failing debris which would necessitate han-

dling moving obstacles while ensuring that the robot maintains a safe distance to avoid collision. One benefit of the probabilistic collision checking approach used within this work and described in Section 3.2.2, is that it naturally allows for the addition of dynamic obstacles and the uncertainty in their measure velocity and position. This is done by rewriting (3.2) as

$$P(C, p_i) \leq \frac{1}{2} + \frac{1}{2}\text{erf}\left(\frac{r_c - a_{ij}^T(p_i - p_j)}{\sqrt{2a_{ij}^T \Sigma_e a_{ij}}}\right) \tag{5.1}$$
$$a_{ij} = \frac{(p_i - p_j)}{||p_i - p_j||}$$

where Σ_e is the effective position uncertainty and is the addition of the velocity-scaled position uncertainties for the robot and the dynamic obstacle, i.e., $\Sigma_e = \Sigma_p + \Sigma_o$. What follows are preliminary results, with future work being to further this track of work. For all examples an assumption is made that the dynamic object can be detected and its velocity and position estimated. Figure 5.1 shows the trajectory for the robot as it avoids two constant velocity obstacles on its way to the goal. The planner forward propagates the obstacle's position by one second given a noisy estimate of its velocity and uses that within the collision checking. A screenshot from the trial is shown in Figure 5.2.

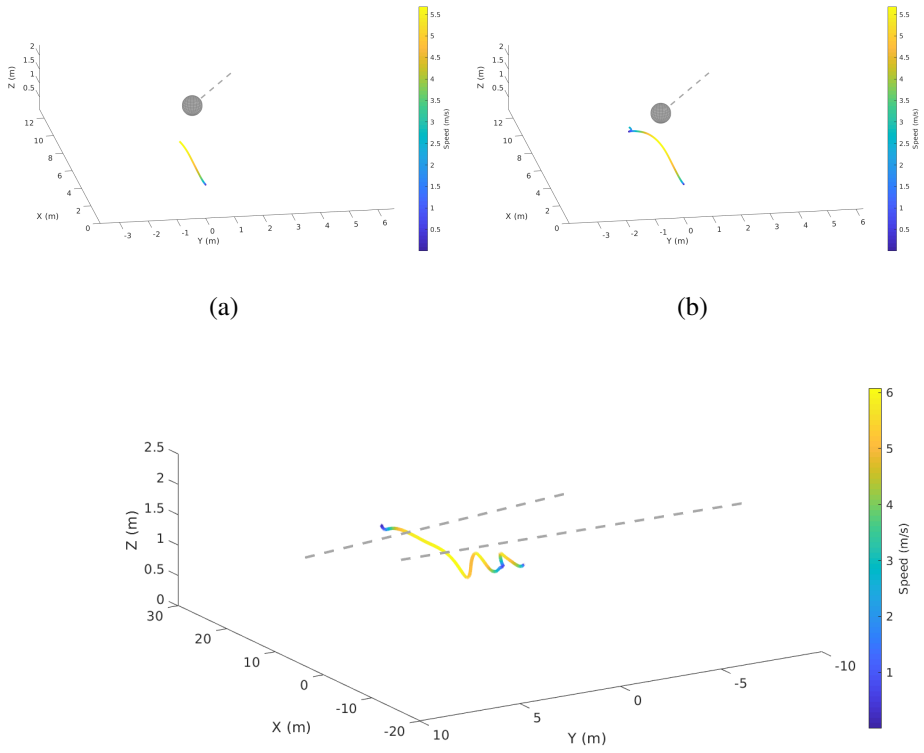


Figure 5.1: (a) Trajectory of the robot approaching a dynamic obstacle. The obstacle is shown as the gray sphere and its path as a dotted gray line. (b) The robot calculates the collision probabilities with respect to the dynamic obstacle and safely avoids it. (c) The final trajectory for the robot around two dynamic obstacles.

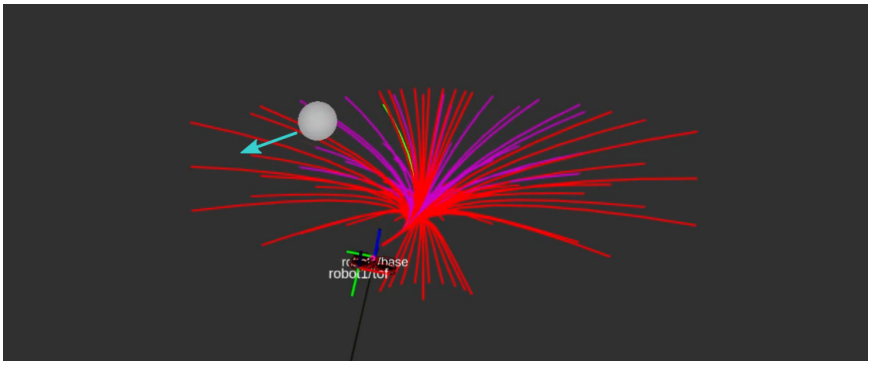


Figure 5.2: Robot planning around a dynamic obstacle. The arrow indicates the velocity vector of the obstacle.

Bibliography

[1] Faster, lighter, smarter: Darpa gives small autonomous systems a tech boost, Jul 2018. URL `https://www.darpa.mil/news-events/2018-07-18`.

[2] Ross Allen and Marco Pavone. A real-time framework for kinodynamic planning with application to quadrotor obstacle avoidance. In *AIAA Guidance, Navigation, and Control Conference*, page 1374, 2016.

[3] Fabian Blochliger, Marius Fehr, Marcin Dymczyk, Thomas Schneider, and Rol Siegwart. Topomap: Topological mapping and navigation based on visual slam maps. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[4] Jing Chen, Tianbo Liu, and Shaojie Shen. Online generation of collision-free trajectories for quadrotor flight in unknown cluttered environments. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1476–1483. IEEE, 2016.

[5] Gabriele Costante, Christian Forster, Jeffrey Delmerico, Paolo Valigi, and Davide Scaramuzza. Perception-aware path planning. *arXiv preprint arXiv:1605.04151*, 2016.

[6] Hugh Cover, Sanjiban Choudhury, Sebastian Scherer, and Sanjiv Singh. Sparse tan-

gential network (spartan): Motion planning for micro aerial vehicles. In *2013 IEEE International Conference on Robotics and Automation*, pages 2820–2825. IEEE, 2013.

[7] Robin Deits and Russ Tedrake. Computing large convex regions of obstacle-free space through semidefinite programming. In *Algorithmic foundations of robotics XI*, pages 109–124. Springer, 2015.

[8] Arjav Desai, Matthew Collins, and Nathan Michael. Efficient kinodynamic multi-robot replanning in known workspaces. In *2019 IEEE international conference on robotics and automation (ICRA)*, pages 1021–1027. IEEE, 2019.

[9] Noel E Du Toit and Joel W Burdick. Probabilistic collision checking with chance constraints. *IEEE Transactions on Robotics*, 27(4):809–815, 2011.

[10] Zacc Dukowitz. Drones in search and rescue: 5 stories of drones helping save lives, Jan 2019. URL `https://uavcoach.com/search-and-rescue-drones/`.

[11] Pete Florence, John Carter, and Russ Tedrake. Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps. In *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2016.

[12] Peter R Florence, John Carter, Jake Ware, and Russ Tedrake. Nanomap: Fast, uncertainty-aware proximity queries with lazy search over local 3d data. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7631–7638. IEEE, 2018.

[13] David Fridovich-Keil, Jaime F. Fisac, and Claire J. Tomlin. Safely probabilistically complete real-time planning and exploration in unknown environments. In *2019*

IEEE international conference on robotics and automation (ICRA), pages 7470–7476. IEEE, 2019.

[14] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3067–3074. IEEE, 2015.

[15] Fei Gao and Shaojie Shen. Online quadrotor trajectory generation and autonomous navigation on point clouds. In *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 139–146. IEEE, 2016.

[16] Daniel Harabor. Jump point search, Sep 2011. URL `https://harablog.wordpress.com/2011/09/07/jump-point-search/#3`.

[17] Daniel Damir Harabor and Alban Grastien. Online graph pruning for pathfinding on grid maps. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[18] Markus Hehn and Raffaello D'Andrea. Quadrocopter trajectory generation and control. *IFAC Proceedings Volumes*, 44(1):1485–1491, 2011.

[19] Brian Ichter and Marco Pavone. Robot motion planning in learned latent spaces. *IEEE Robotics and Automation Letters*, 2019.

[20] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research*, 34(7):883–921, 2015.

[21] Lucas Janson, Tommy Hu, and Marco Pavone. Safe motion planning in unknown environments: Optimality benchmarks and tractable policies. *Robotics: science and systems*, 2018.

[22] Benoit Landry, Robin Deits, Peter R Florence, and Russ Tedrake. Aggressive quadrotor flight through cluttered environments using mixed integer programming. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1469–1475. IEEE, 2016.

[23] Sikang Liu, Michael Watterson, Sarah Tang, and Vijay Kumar. High speed navigation for quadrotors with limited onboard sensing. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1484–1491. IEEE, 2016.

[24] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay 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)*, pages 2872–2879. IEEE, 2017.

[25] Sikang Liu, Michael Watterson, Kartik Mohta, Ke Sun, Subhrajit Bhattacharya, Camillo J Taylor, and Vijay Kumar. Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments. *IEEE Robotics and Automation Letters*, 2(3):1688–1695, 2017.

[26] Sikang Liu, Kartik Mohta, Nikolay Atanasov, and Vijay Kumar. Search-based motion planning for aggressive flight in se (3). *IEEE Robotics and Automation Letters*, 3(3): 2439–2446, 2018.

[27] Brett T Lopez and Jonathan P How. Aggressive 3-d collision avoidance for high-speed navigation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5759–5765. IEEE, 2017.

[28] Brian MacAllister, Jonathan Butzke, Alex Kushleyev, Harsh Pandey, and Maxim Likhachev. Path planning for non-circular micro aerial vehicles in constrained en-

vironments. In *2013 IEEE International Conference on Robotics and Automation*, pages 3933–3940. IEEE, 2013.

[29] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525. IEEE, 2011.

[30] Mark W Mueller, Markus Hehn, and Raffaello D'Andrea. A computationally efficient motion primitive for quadrocopter trajectory generation. *IEEE Transactions on Robotics*, 31(6):1294–1310, 2015.

[31] Matthias Nieuwenhuisen and Sven Behnke. Search-based 3d planning and trajectory optimization for safe micro aerial vehicle flight under sensor visibility constraints. In *2019 IEEE international conference on robotics and automation (ICRA)*, pages 9123–9129. IEEE, 2019.

[32] Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran. Continuous-time trajectory optimization for online uav replanning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5332–5339. IEEE, 2016.

[33] Helen Oleynikova, Zachary Taylor, Alexander Millane, Roland Siegwart, and Juan Nieto. A complete system for vision-based micro-aerial vehicle mapping, planning, and flight in cluttered environments. *arXiv preprint arXiv:1812.03892*, 2018.

[34] Helen Oleynikova, Zachary Taylor, Roland Siegwart, and Juan Nieto. Sparse 3d topological graphs for micro-aerial vehicle planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[35] Cormac OMeadhra. Generative point cloud modeling with gaussian mixture models for multi-robot exploration. Master's thesis, Pittsburgh, PA, August 2018.

[36] Alyssa Pierson, Cristian-Ioan Vasile, Anshula Gandhi, Wilko Schwarting, Sertac Karaman, and Daniela Rus. Dynamic risk density for autonomous navigation in cluttered environments without object detection. In *2019 IEEE international conference on robotics and automation (ICRA)*, pages 5807–5814. IEEE, 2019.

[37] Ahmed H Qureshi, Anthony Simeonov, Mayur J Bency, and Michael C Yip. Motion planning networks. *arXiv preprint arXiv:1806.05767*, 2018.

[38] Nathan Ratliff, Matt Zucker, J Andrew Bagnell, and Siddhartha Srinivasa. Chomp: Gradient optimization techniques for efficient motion planning. 2009.

[39] Charles Richter and Nicholas Roy. Safe visual navigation via deep learning and novelty detection. 2017.

[40] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research*, pages 649–666. Springer, 2016.

[41] Markus Ryll, John Ware, John Carter, and Nick Roy. Efficient trajectory planning for high speed flight in unknown environments. In *2019 IEEE international conference on robotics and automation (ICRA)*, pages 732–738. IEEE, 2019.

[42] Adrian Sainz. Rescuers use heat-seeking drones after tornado, Mar 2019. URL `https://www.pbs.org/newshour/nation/rescuers-use-heat-seeking-drones-after-tornado`.

[43] Jürgen Scherer, Saeed Yahyanejad, Samira Hayat, Evsen Yanmaz, Torsten Andre, Asif Khan, Vladimir Vukadinovic, Christian Bettstetter, Hermann Hellwagner, and

Bernhard Rinner. An autonomous multi-uav system for search and rescue. In *Proceedings of the First Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, pages 33–38. ACM, 2015.

[44] Edward Schmerling, Lucas Janson, and Marco Pavone. Optimal sampling-based motion planning under differential constraints: the driftless case. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2368–2375. IEEE, 2015.

[45] Alex Spitzer, Xuning Yang, John Yao, Aditya Dhawale, Kshitij Goel, Mosam Dabhi, Matt Collins, Curtis Boirum, and Nathan Michael. Fast and agile vision-based flight with teleoperation and collision avoidance on a multirotor. In *Proc. of the Intl. Sym. on Exp. Robot*. Springer, 2018, to be published.

[46] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[47] Teodor Tomic, Korbinian Schmid, Philipp Lutz, Andreas Domel, Michael Kassecker, Elmar Mair, Iris Lynne Grixa, Felix Ruess, Michael Suppa, and Darius Burschka. Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue. *IEEE robotics & automation magazine*, 19(3):46–56, 2012.

[48] Jesus Tordesillas, Brett T Lopez, John Carter, John Ware, and Jonathan P How. Real-time planning with multi-fidelity models for agile flights in unknown environments. *arXiv preprint arXiv:1810.01035*, 2018.

[49] Jesus Tordesillas, Brett T Lopez, and Jonathan P How. Fastrap: Fast and safe trajectory planner for flights in unknown environments. *arXiv preprint arXiv:1903.03558*, 2019.

[50] Sonia Waharte and Niki Trigoni. Supporting search and rescue operations with uavs. In *2010 International Conference on Emerging Security Technologies*, pages 142–147. IEEE, 2010.

[51] Michael Watterson and Vijay Kumar. Safe receding horizon control for aggressive mav flight with limited range sensing. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3235–3240. IEEE, 2015.

[52] Xuning Yang, Ayush Agrawal, Koushil Sreenath, and Nathan Michael. Online adaptive teleoperation via motion primitives for mobile robots. *Autonomous Robots*, pages 1–17, 2018.

[53] Zichao Zhang and Davide Scaramuzza. Perception-aware receding horizon navigation for mavs. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2534–2541. IEEE, 2018.

[54] Hai Zhu and Javier Alonso-Mora. Chance-constrained collision avoidance for mavs in dynamic environments. *IEEE Robotics and Automation Letters*, 4(2):776–783, 2019.