Depth-based Reactive Quadrotor Motion Planning and Control in Diverse Environments

Jonathan Lee

CMU-RI-TR-25-68

July 30, 2025

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

Thesis Committee:
Dr. Wennie Tabib, Carnegie Mellon University *(Chair)*
Dr. Wenshan Wang, Carnegie Mellon University
Seungchan Kim, Carnegie Mellon University

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

Abstract

This thesis explores motion planning and control strategies for enabling rapid quadrotor navigation in unknown environments using only limited field-of-view depth sensors. We propose real-time onboard algorithms that enable agile flight through diverse and cluttered spaces. First, we present a reactive planner based on forward-arc motion primitives that uses a short history of RGB-D observations to safely navigate near obstacles. A safe stopping strategy ensures that the quadrotor always maintains a trajectory that allows it to hover safely within known free space. Second, we build on recent advances in reinforcement learning with differentiable physics to develop a navigation policy that predicts thrusts directly from depth and state observations. We show that by using privileged information during training, our approach is able to navigate around large obstacles. Through extensive simulation and real-world experiments, we show that our methods outperform baselines in both speed and reliability in cluttered environments. These results contribute toward the deployment of agile autonomous flight in real-world applications such as search and rescue and exploration.

Acknowledgments

I would like to thank my advisor, Wennie Tabib, for her mentorship and support since the first day of my Master's. I deeply appreciate the trust she placed in me and helping me take on exciting projects as a new researcher in the lab. I am especially grateful for her passion and wealth of experience in field robotics, which have driven many of the results in this thesis and created many memorable robotic deployments. Above all, her unwavering commitment to her students and the lab has been a constant source of inspiration and motivation. I would also like to thank my committee members, Wenshan Wang and Seungchan Kim, for providing me with valuable feedback and being gracious with their time.

I am especially grateful to Kshitij Goel who, as the most senior member of the lab, took on the role of a mentor and met with me on a weekly basis. His first-principles approach to research helped keep me on the right track and improve the quality of my work. I would also like to thank Abhishek Rathod, not only for being an expert at all things controls and hardware, but for also being a great friend. I always enjoyed your thoughtful conversation whether it was tennis or research related.

This thesis also features custom robotic platforms which were engineered by John Stecklein with firmware support from Wennie and Abhishek. Thanks John for delivering on all my last minute 3D printed part and obstacle course requests. Thanks to Edsel Burkholder for providing field testing support and Ken Bailey for hosting our cave experiments.

And to all the members of RISLab - Ankit Khandelwal, Mike Anoruo, Nicole Chan, Johnny Tian, Steven Yang, Lucky Kant Nayak, and Akshay Chekuri - for your camaraderie and friendship, which made the lab such a fun and collaborative place to work.

Finally, I would like to thank my family who have always being there for me and encouraged me at every step.

Looking back, I feel incredibly fortunate to have been surrounded by such talented and inspiring group of people. Thank you all for being part of this journey.

Funding

This material is based upon work supported by, or in part by, the Army Research Laboratory and the Army Research Office under contract/grant number W519TC-23-C-0031 and W911NF-25-2-0153.

Contents

When this document is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

Autonomous quadrotors have the potential to aid in a wide range of real-world applications from disaster response and search-and-rescue missions to infrastructure inspection and exploration of unknown environments. These scenarios demand fast, agile flight through cluttered, and previously unseen spaces, often under sensing and compute constraints. In practice, quadrotors are typically equipped with lightweight onboard sensors such as depth cameras with limited field of view and range, which makes reliable perception of the surrounding environment both partial and uncertain. This poses a significant challenge for motion planning and control, especially when rapid navigation is required. To safely and efficiently operate in such settings, it is crucial to develop planning and control strategies that are not only computationally lightweight and reactive but also capable of generalizing to novel and complex environments. This thesis addresses this challenge by investigating how to design motion planning and control frameworks that leverage limited sensory information while enabling robust, high-speed autonomous flight in unstructured environments.

Chapter 2

Background

2.1 Quadrotor Motion Planning

A key requirement for quadrotor motion planning is to generate a set of instructions for the quadrotor to follow such that the resulting motion satisfies the vehicle's dynamic constraints while remains collision-free. This section discusses several common representations used in motion planning.

One widely used approach is to represent trajectories as a sequence of piecewise time-parameterized polynomials [1, 2, 3]. This representation is advantageous because polynomials can express complex motion using only a small number of control points. They are continuous, differentiable, and defined at every point in time, making them well-suited for generating smooth and dynamically feasible motion.

Various polynomial basis functions have been explored in the context of quadrotor motion planning, including Bernstein basis functions [1], B-splines [4], and Legendre polynomials [5]. Novel classes of polynomial representations have also been proposed, such as MINVO [6], which minimizes the volume of the polynomial convex hull, and MINCO [7], which generates time-optimal trajectories with minimal control effort.

Other methods instead produce a sequence of discrete waypoints, such as model predictive control (MPC) approaches that solve discrete-time optimal control problems using multiple shooting [8]. Some other approaches do not explicitly plan a trajectory into the future. For example, Zhang et al. [9] predicts a single reference state at a time and relies on high-rate replanning to ensure closed-loop performance.

2.2 Quadrotor Control

Once a high-level reference trajectory or control command is planned, a low-level controller computes the actuator signals sent to the electronic speed controllers (ESCs) to regulate the motors and generate thrust. This is achieved by defining a control law designed to minimize tracking errors by computing the required forces and torques.

The low-level controller runs onboard the flight controller at a high frequency, often in the range of hundreds to thousands of Hertz. Accurate state estimation is critical for effective feedback control. To this end, the flight controller fuses sensor data (such as gyroscope, accelerometer, barometer, and magnetometer measurements) using filtering techniques like complementary filters [10] or extended Kalman filters (EKFs) [11]. Position estimates can also be obtained by incorporating data from visual odometry, global navigation satellite system (GNSS), or motion capture systems, depending on the available sensing modalities.

A common strategy for low-level control is to use a cascaded control architecture that decouples the translational and rotational degrees of freedom, which simplifies the control design [12, 13]. The outer loop governs the translational dynamics, including position, linear velocity, and acceleration, while the inner loop regulates the rotational dynamics, such as attitude, angular velocity, and angular acceleration.

As an alternative to cascaded control, model predictive control (MPC) offers a unified framework that accounts for the full nonlinear dynamics of the quadrotor. MPC formulates and solves an optimization problem over a prediction horizon, allowing it to handle constraints and anticipate future states explicitly [8].

Chapter 3

Related Work

This section reviews methods for agile navigation with depth sensors in unknown environments.

3.1 Classical Navigation Strategies

Classical navigation strategies rely on hand-engineered components with explicit modules such as mapping, localization, motion planning, and control. These methods use geometric reasoning and optimization instead of learning from data. A key advantage of classical approaches is their interpretability. Trajectories can be explicitly verified for safety, including collision avoidance and dynamic feasibility.

3.1.1 Map-based Methods

Map-based approaches fuse sensor data into intermediate representations such as occupancy grids [14] and signed distance fields [2]. These representations are typically used in conjunction with trajectory optimization strategies to ensure that resulting paths are smooth and collision-free.

A common strategy is to decompose the planning problem into two stages. The first stage generates a kinematically feasible geometric path without considering dynamic constraints. Sample-based planners such as RRT [15], PRM [16], and their variants typically search for collision-free piecewise linear paths in the robot's

configuration space or on a volumetric map [3]. These serve to reduce the search space and initialize trajectory optimization.

In the second stage, a time-parameterized polynomial trajectory is initialized from the geometric path and iteratively refined to be both collision-free and dynamically feasible. Richter et al. [3] proposed resolving collisions by inserting additional waypoint constraints between trajectory segments. Oleynikova et al. [17] incorporated a collision penalty term into the objective function and used gradient information from an Euclidean Signed Distance Field (ESDF) to guide optimization.

ESDF-based methods [2, 17, 18, 19] maintain high-resolution volumetric representations, making them suitable for dense mapping applications. However, maintaining such a datastructure is computationally expensive and may not scale well to high-speed navigation. To address this, ESDF-free alternatives such as Zhou et al. [20] have been proposed. Their method leverages a collision-free guiding path to provide optimization gradients, significantly reducing computational overhead.

Another strategy for ensuring safety in trajectory optimization is to constrain motion within a collision-free safe flight corridor (SFC), a region of known free space where the entire trajectory is guaranteed to remain obstacle-free. One common formulation uses a sequence of convex polytopes [21], allowing optimization to be performed efficiently with continuous collision guarantees. Real-time implementations typically generate an initial path using graph search and inflate convex regions along it. Some methods construct axis-aligned corridors using octree operations [22], while others use polyhedron dilation to better approximate free space and reduce computation time [23, 24, 25].

SFCs can also be constructed from overlapping spheres [26], simplifying generation but potentially covering less free space. Most approaches plan conservatively within known free space, but FASTER [27] extends optimization into both known and unknown regions to support high-speed flight. Overall, SFC methods provide strong safety guarantees and support fast optimization, though their performance depends on the accuracy of the underlying map and the complexity of the environment.

The ability to maintain a local history about previously observed obstacles helps map-based methods plan complex trajectories and overcome incomplete observations of the environment from a limited field of view sensor. However, compounding errors and the added computation cost of maintaining intermediate data structures can

introduce more planning latency compared to reactive approaches [28].

3.1.2 Reactive Methods

Reactive methods act directly on instantaneous sensor data or maintain a short spatio-temporal history of observations. Common approaches for ensuring trajectories are collision free include generating a SFC within a depth image [29], maintaining a local 3D occupancy grid using a circular buffer [4], or generating a k-d tree for nearest neighbor queries [28]. Reactive methods often execute sampled trajectories or motion primitives in order to reduce planning latency.

RAPPIDS [29] generates trajectories within the latest depth image by iteratively partitioning free space into rectangular pyramids for efficient collision detection; however, the planner is prone to becoming trapped in cluttered environments due to the pyramidal corridor constraints and lack of yaw control. BiTE [30] accelerates collision checks with a local occupancy grid by pre-computing a bitwise map and trading off increased memory usage for a fixed path library. This library, however, is not continuous for higher-order derivatives of velocity. NanoMap [28] searches for the minimum uncertainty view of a queried point in space by maintaining a short temporal history of depth images and their relative poses. Ji et al. [26] improves upon NanoMap and constructs a forward spanning tree to find a SFC for trajectory generation.

In Chapter 4, we propose a reactive navigation approach most similar to Florence et al. [31], which utilizes NanoMap for collision queries. However, [31] uses a trajectory representation which demands high control effort and lacks guaranteed safe stopping behaviors. In contrast, our approach always includes a safe stopping trajectory, ensuring the robot will stop in known free space if no feasible action exists.

3.2 Learning-based Navigation Strategies

Although learning-based navigation methods can also be categorized as reactive, since they act on instantaneous sensor data, they differ fundamentally from traditional reactive approaches. Rather than optimizing a cost function online as in classical motion planning, learning-based methods shift the optimization offline.

Learning-based navigation techniques leverage a wide range of data-driven training strategies and vary in both methodology and neural architecture design. Imitation learning methods [32] rely on expert demonstrations to supervise policy learning. While effective, they often require large amounts of labeled data, which can be costly and time-consuming to collect. In contrast, other approaches exploit privileged information available during simulation, such as signed distance fields or closest point collision vectors, to train policies more efficiently [9, 33]. Some methods also incorporate differentiable system dynamics, enabling gradient-based optimization of policies without requiring expert supervision [9].

These approaches also differ in the nature of their outputs. Some predict a sequence of future reference states or trajectories for downstream controllers to track [32, 33, 34], while others generate single-step control actions at each timestep, relying on high-rate replanning to maintain stability and control [9].

3.2.1 Imitation Learning

Loquercio et al. [32] demonstrated high-speed, learning-based navigation by training a student policy in simulation using a privileged, sample-based expert planner. The student learned to predict control points of a polynomial trajectory based on top-ranked expert samples. To prevent the student from shifting out-of-distribution with expert samples, they employed dataset aggregation (DAgger) [35], which iteratively collects expert labels on states visited by the learner. While effective, this approach requires a large amount of expert-labeled data and increases the training overhead when the expert planner is computationally intensive.

Inverse reinforcement learning (IRL) methods [36] similarly rely on expert demonstrations, but instead of directly imitating the expert, they infer a reward function that explains the observed behavior. A policy is then trained to optimize this learned reward, often improving generalization to unseen scenarios. Unlike behavior cloning, which typically requires dataset aggregation to remain effective, IRL can better guide the learner in out-of-distribution regions through the learned reward. Kim et al. [34] developed an IRL method for high-speed, depth-based quadrotor navigation that successfully deployed in real-world scenarios without additional training or tuning. However, the policy lacked temporal awareness during inference, making it prone to

falling into local minima, particularly when navigating around large obstacles such as walls, which often led to collisions.

3.2.2 Learning with Privileged Information

Imitation learning often demands large amounts of expert data, which can be impractical if the expert policy is slow or costly to run. An alternative to relying on expert data is to train directly in simulation with access to privileged environment information, such as a ground-truth map or a signed distance field, that is unavailable at deployment.

YOPO [33] trains a neural policy to predict offsets and scores for motion primitives using a ESDF cost map as a privileged training signal. During training, gradients from the ESDF cost is backpropagated to the policy parameters, but this information is not used at test time. Compared to a trajectory optimization planner [37] that constructs ESDF maps online, YOPO achieves lower average cost and planning latency. This highlights the effectiveness of using full privileged information during training to learn robust policies that can generalize at deployment despite not relying limited sensor observations and no explicit map construction.

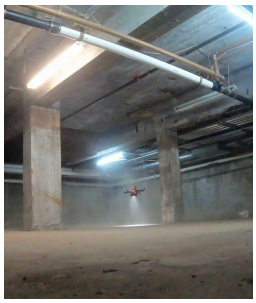
3.2.3 Differentiable Reinforcement Learning

Zhang et al. [9] recently proposed a recurrent neural network control policy trained via direct backpropagation of reward gradients through a differentiable simulator. This approach differs from model-free reinforcement learning and imitation learning, which rely on sampled interactions with the environment to estimate policy gradients. Instead [9] computes gradients analytically through a differentiable dynamics model. It eliminates the need for expert demonstrations and enables sample-efficient learning using only a small set of cost functions. Their results show that policies trained in simplified simulation environments can successfully transfer to real-world settings, achieving flight speeds of up to 20 m/s in previously unseen environments.

Chapter 4

Reactive Navigation with Forward Arc Motion Primitives

(a) Cave



(b) Industrial tunnel

(c) Forest

Figure 4.1: (a) Tight spaces, (b) high amounts of particulate matter such as dust and (c) thin obstacles such as branches are hazards found in search and rescue environments. To aid search teams, autonomous aerial systems must rapidly navigate these dangers without posing additional risks for rescuers or victims. This chapter proposes a rapid quadrotor navigation system, which uses forward-arc motion primitives and a forward-facing depth camera to achieve speeds up to 6 m/s in cluttered environments. Safety is achieved by executing a safe stopping action when no feasible action is found. Experiments are conducted in diverse environments, including caves and forests. A video of these experiments may be found at https://youtu.be/tk8vUot0gD4

4.1 Introduction

Extreme search and rescue environments exhibit challenging 3D geometry (e.g., confined spaces, rubble, breakdown), which preclude rapid traversal by legged, wheeled, and tracked robotic systems. Quadrotors are highly agile and maneuverable, but are limited by battery capacity and flight time. Therefore, these systems must rapidly navigate through the environment. More importantly, onboard autonomous navigation must be robust as collisions endanger rescuers and victims. To achieve these goals in highly cluttered environments with narrow gaps, it is advantageous to leverage smaller size robots equipped with a lightweight, short-range depth camera as opposed to a heavy, long-range LiDAR.

In this work, a rapid quadrotor navigation methodology is proposed, which leverages a forward-facing, limited field-of-view RGB-D camera to operate in diverse environments (e.g., caves, sewers, forests, industrial tunnels) without scene-specific parameter tuning or training. A reactive planner is developed using forward-arc motion primitives, which are differentiable up to jerk and continuous up to snap to enable aggressive flight. Motion primitive selection is based on a perception front-end that searches a history of RGB-D observations to safely maneuver in close proximity to obstacles.

4.1.1 Contributions

The contributions of this work are:

1. **Reactive planning with depth history:** A navigation framework that evaluates a library of forward-arc motion primitives using a short history of depth observations.
2. **Collision-aware trajectory scheduling:** A trajectory scheduling strategy that ensures safe stopping by maintaining a feasible collision-free trajectory that stops completely within known free space.
3. **Extensive simulation validation:** Evaluation in photorealistic simulation across 3150 trials in diverse, cluttered environments. The proposed approach achieved a 24% higher success rate compared to state-of-the-art reactive navigation approaches.

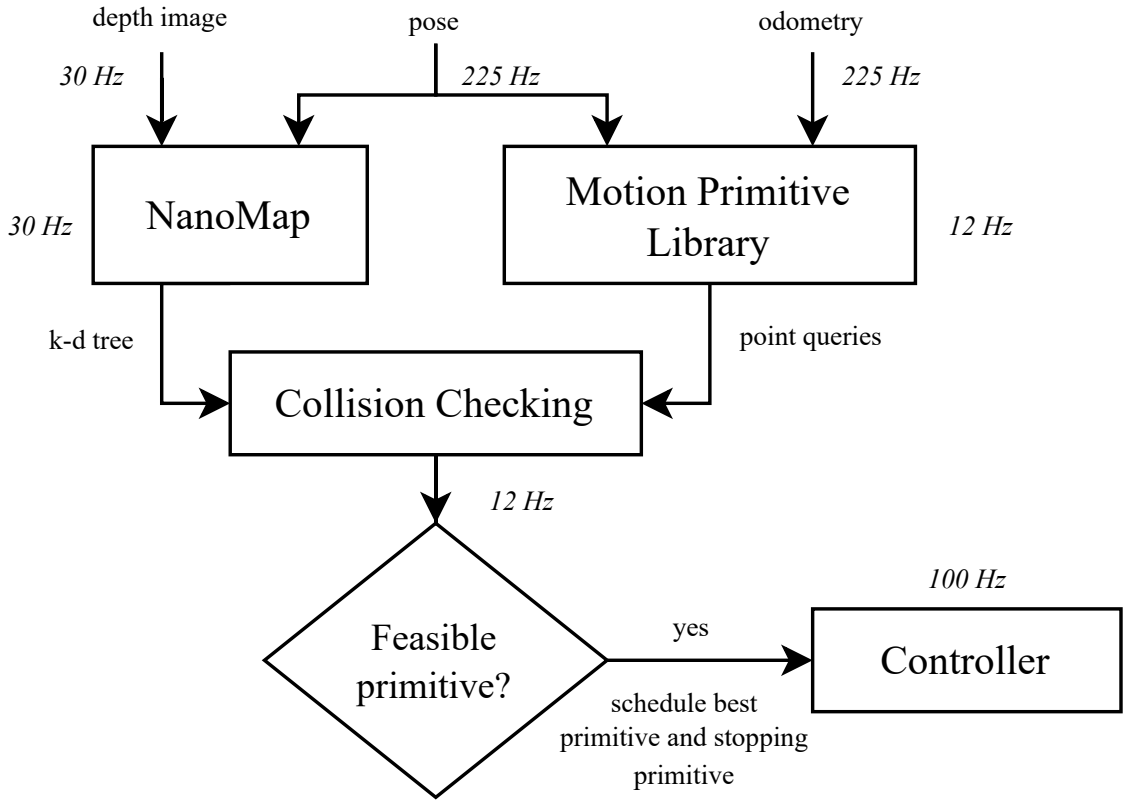


Figure 4.2: System diagram of the navigation algorithm. Given depth images and odometry, NanoMap [31] is used for collision avoidance and a library of forward-arc motion primitives is generated for motion planning. To maintain safety, collision-free trajectories are scheduled such that a feasible stopping action is always available within the known free space.

4. **Real-world deployment:** Hardware experiments demonstrating 571 m of collision-free flight in successful trials and achieving a top speed of 6 m/s.

4.2 Technical Approach

This section details the perception and planning strategy for rapid navigation. The high-level system architecture for the approach is illustrated in Fig. 4.2. A forward-arc motion primitive library is used to generate smooth actions, which are evaluated for collisions. The perception system maintains a 3D point cloud representation of the local environment and searches over a history of observations. Of the feasible motion

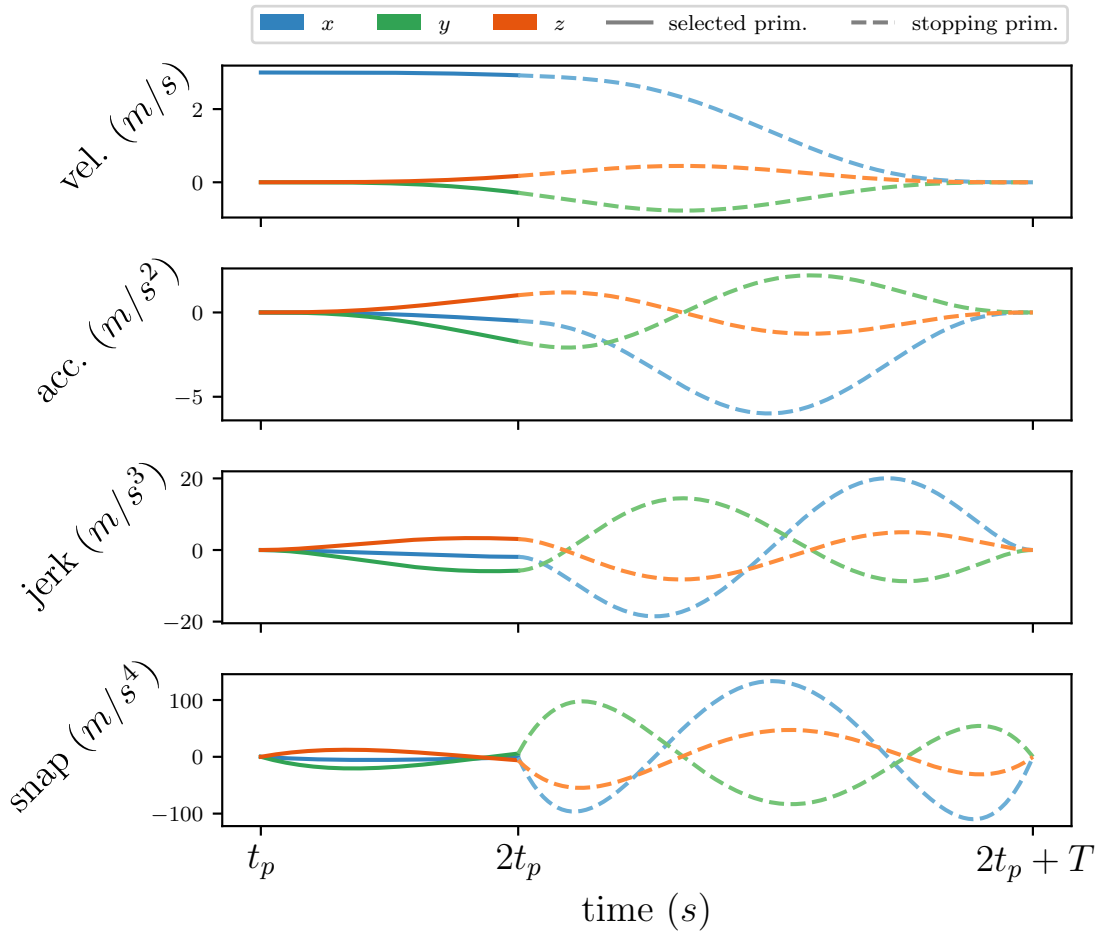


Figure 4.3: Derivatives of the scheduled trajectory are continuous up to snap and smooth up to jerk. The planning strategy ensures the robot stops in a safe region.

primitives, the one with the lowest cost is scheduled for execution. Additionally, a safe stopping primitive is scheduled, which is executed if no feasible motion primitive is found at the next planning round.

4.2.1 Forward-Arc Motion Primitive Library

Forward-arc motion primitives propagate the dynamics of a unicycle model for time T and are parameterized by a desired yaw rate and forward and vertical velocities [38]. This action formulation is differentiable up to jerk and continuous up to snap (see Fig. 4.3), which is advantageous for aggressive multirotor flight where

large angular velocities and accelerations are directly related to the jerk and snap of the reference position [39, 40].

Let $\mathcal{B}$ represent the body frame of the multirotor. The position and heading of the vehicle is represented as $\boldsymbol{\xi}_t = \begin{bmatrix} x & y & z & \theta \end{bmatrix}^\top$. The linear velocities are expressed in the body frame as $\begin{bmatrix} v^{\mathcal{B}}_{x,t} & v^{\mathcal{B}}_{z,t} \end{bmatrix}$. The angular velocity, $\omega^{\mathcal{B}}_{z,t}$ is expressed about the $\mathbf{z}^{\mathcal{B}}$ axis. The equations of the motion primitives are given by the solutions to the unicycle model [40, 41]:

$$\boldsymbol{\xi}_{t+T} = \boldsymbol{\xi}_t + \begin{bmatrix} \frac{v^{\mathcal{B}}_{x,t}}{\omega^{\mathcal{B}}_{z,t}}(\sin(\omega^{\mathcal{B}}_{z,t}T + \theta_t) - \sin(\theta_t)) \\ \frac{v^{\mathcal{B}}_{x,t}}{\omega^{\mathcal{B}}_{z,t}}(\cos(\omega^{\mathcal{B}}_{z,t}T) - \cos(\omega^{\mathcal{B}}_{z,t}T + \theta_t)) \\ v^{\mathcal{B}}_{z,t}T \\ \omega^{\mathcal{B}}_{z,t}T \end{bmatrix}. \tag{4.1}$$

The motion primitive, γ, is parameterized by $\mathbf{a}_t = \{v^{\mathcal{B}}_{x,t}, v^{\mathcal{B}}_{z,t}, \omega^{\mathcal{B}}_t\}$ and duration T. $v^{\mathcal{B}}_{x,t}$ is fixed by the user. $v^{\mathcal{B}}_{z,t} \in \mathcal{V}_z$ and $\omega^{\mathcal{B}}_t \in \Omega$ are varied according to a user-specified discretization. The motion primitive library, Γ, is generated as the set of motion primitives created by varying $v^{\mathcal{B}}_{z,t}$ and $\omega^{\mathcal{B}}_t$ (see Fig. 4.4 for an illustration of the library).

Motion primitives are scheduled at a fixed planning rate as shown in Figs. 4.3 and 4.4. The selected primitive is executed from $[t_p, 2t_p)$. The stopping primitive is scheduled from $[2t_p, 2t_p + T)$ so that if the next planning round fails to find a feasible motion primitive, the robot stops in free space.

4.2.2 Local 3D Perception and Collision Checking

Nanomap [28] is used for high-rate proximity queries for collision avoidance. We provide a brief overview of the query search algorithm, which uses k-nearest neighbor queries over a short temporal history of depth measurements.

NanoMap maintains a chain of edge-vertex pairs where each edge contains the relative transform $T^{\mathcal{S}_i}_{\mathcal{S}_{i-1}}$ between consecutive sensor frames and each vertex contains the ordered point cloud and corresponding k-d tree. The motion primitive representing the position of the vehicle, $\gamma^{(0)}$, is sampled in time using a fixed Δt up to the trajectory

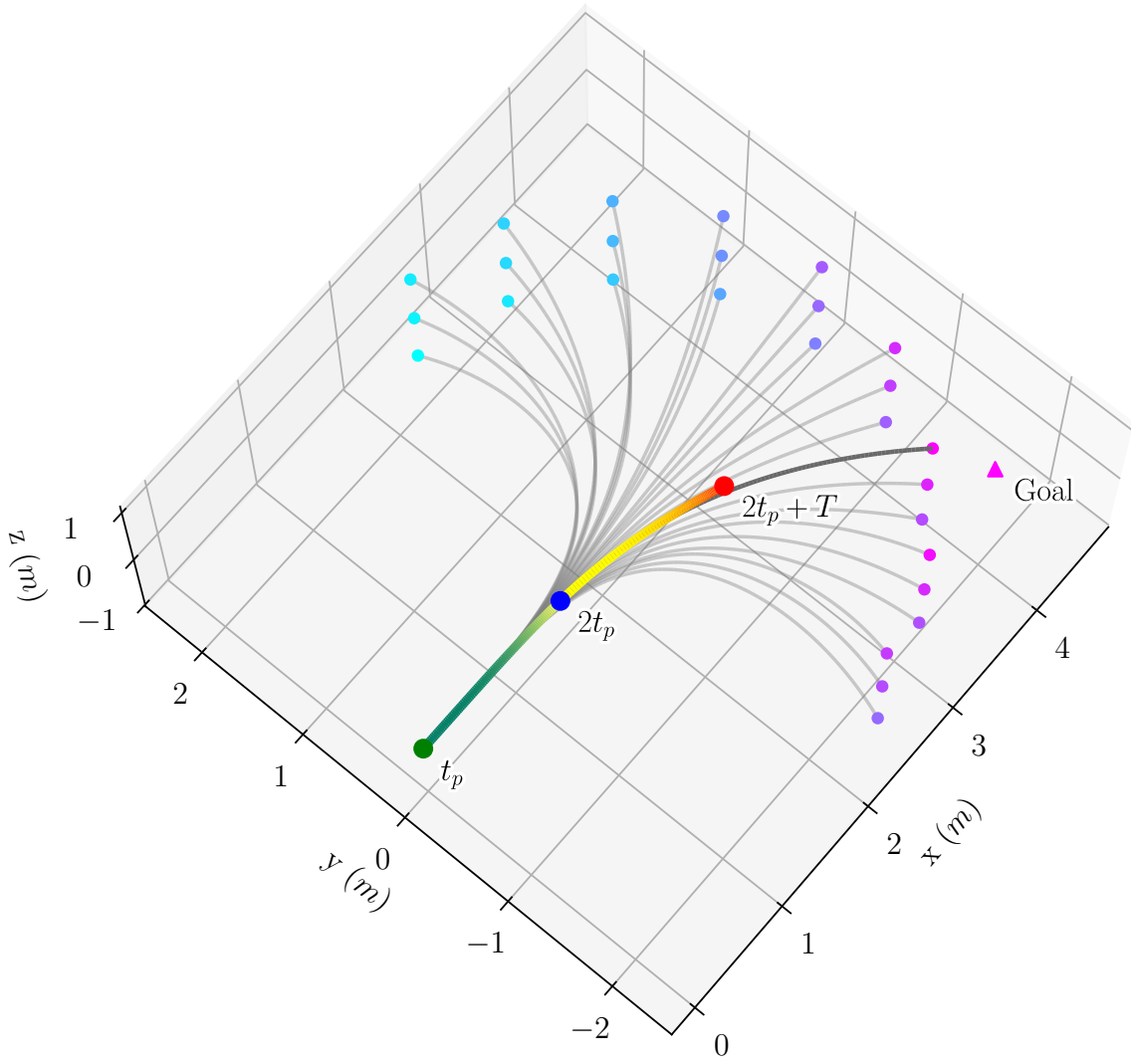


Figure 4.4: Illustrative example of trajectory scheduling with the motion primitive library. The motion primitives in the library are shown in gray. The cost of each primitive is evaluated by computing the Euclidean distance between the endpoint and goal (shown as a pink triangle). The endpoints are shown as dots colored from purple to blue, where more pink indicates closer to the goal. Motion primitives that are in collision are pruned. The primitive with lowest cost (shown in dark gray) is selected for execution. The selected primitive segment is scheduled from times $[t_p, 2t_p)$ and the stopping primitive is scheduled from $[2t_p, 2t_p + T)$.

duration, T, to generate query points, $\mathbf{p}^{\mathcal{B}}_{\text{query}} \in \mathbb{R}^3$.

$$\mathbf{p}^{\mathcal{B}}_{\text{query}} = \gamma^{(0)}(t). \tag{4.2}$$

Query points sampled from the motion plan are iteratively transformed into previous sensor frames until a view containing the query point is found. The transformation to the i-th coordinate frame is obtained by

$$\mathbf{p}_{\text{query}}^{\mathcal{S}_i} = \prod_{j=1}^{i} \left[T_{\mathcal{S}_{j-1}}^{\mathcal{S}_j} \right] T_{\mathcal{B}}^{\mathcal{S}_0} \mathbf{p}_{\text{query}}^{\mathcal{B}} \tag{4.3}$$

Once a query is determined to be in view, the ordered point cloud is used to determine whether the point is within free space, after which the k-d tree is used to evaluate the k-nearest neighbors. Note that in Eq. (4.3), the uncertainty propagation of the query point is disabled. This modification is made because we leverage a more accurate optimization-based visual inertial navigation system [42] for state estimation compared to what is used in [31].

For each query position, $\mathbf{p}_{\text{query}}^{\mathcal{B}}$, the distance, d, is found between $\mathbf{p}_{\text{query}}^{\mathcal{B}}$ and the nearest neighbor in the depth image. For a user-specified collision radius r_{coll}, the primitive is marked infeasible if $d < r_{\text{coll}}$. Feasible primitives are scored using the Euclidean distance to the goal position, g, from the end of the primitive, $\texttt{Cost}(\gamma, g) = \|g - \gamma^{(0)}(T)\|_2$. The primitive that minimizes the cost function is scheduled. Figure 4.2 shows an overview of the final algorithm along with operating frequencies.

4.3 Experiments

The approach is evaluated in simulation (see Sections 4.3.1 and 4.3.2) and with hardware experiments (see Section 4.3.3). In simulation evaluations, the proposed approach is compared against two reactive vision-based methods, RAPPIDS [29][1] and Florence et al. [31][2], which demonstrate local planning in unknown environments. For the rest of this section, we will refer to these approaches as *RAPPIDS* and *Florence*, respectively, and to the proposed approach as *Forward-Arc*. We disable the "dolphin" oscillations of [31], which were used to improve state estimation, because this is perfectly known in simulation. Primitives that modulate z-height were found to cause crashes due to altitude drift, so these were also removed. To set the desired

[1] https://github.com/nlbucki/RAPPIDS
[2] https://github.com/peteflorence/nanomap_ros

(a) Cave (b) Industrial area (c) Sewer system

Figure 4.5: ((a))–((c)) illustrate a subset of the simulation environments used to validate the proposed approach.

speed for *RAPPIDS* we adjust the allocated trajectory duration such that the planner achieves this speed in an open field. We also use a velocity tracking yaw angle to enable navigation in unstructured environments and generalize beyond the corridor environments tested in [29].

Two simulation evaluations are conducted. First, diverse environments representative of disaster scenarios are used to evaluate the success rate of reaching a goal within a specified time (detailed in Section 4.3.1). Second, simulations that vary the obstacle density (e.g. forest-like environments) are detailed in Section 4.3.2. In addition to the success rates, this second set of simulation experiments evaluates the approaches in terms of flight time, path length, and control effort. The Flightmare simulator [43] is used to generate photorealistic images and depth maps at 30 Hz. Simulations are run on a desktop computer with the planner running on a single thread of an Intel i9-14900K CPU. An NVIDIA RTX 4090 GPU is leveraged for photorealistic rendering within Flightmare.

4.3.1 Simulated Disaster Scenarios

Simulations are conducted in caves, a mine, abandoned industrial environments, and a sewer system (Fig. 4.5). Multiple start and goal locations are specified to evaluate the robustness of each approach. For each pair of start and goal locations each planner is run 5 times for a total of 1350 trials (see Table 4.1). A trial is considered successful

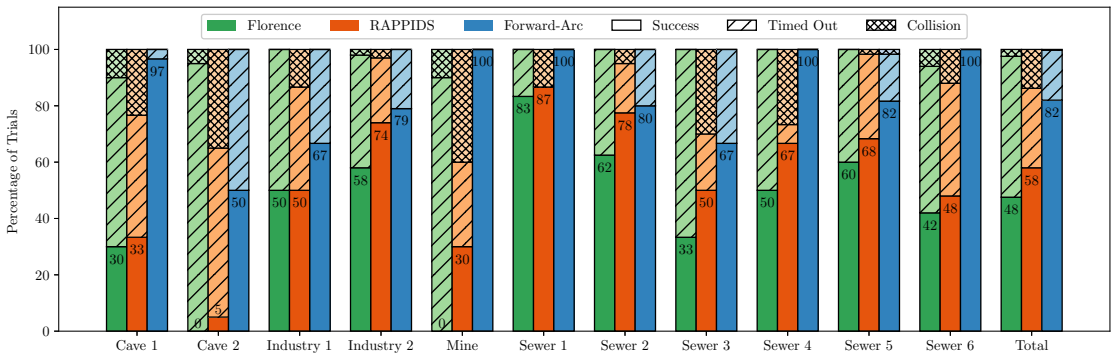


Figure 4.6: Planner success rate and failure modes across the environments detailed in Table 4.1. The proposed method (*Forward-Arc*) achieves the highest success rate and lowest collision rate compared to the baseline reactive planners [29, 31]. A total of 1350 trials are run (450 for each approach).

if the robot reaches the goal within a time limit. The sensing range is set to 10 m for all planning approaches.

The planner performance is evaluated by measuring the success rate and failure modes (collisions and timeouts). The results are reported in Fig. 4.6. *Forward-Arc* outperforms the baseline methods in terms of success rate (82%) and has the fewest collisions (1 out of 450 trials). In contrast, we find that many *Florence* collisions are caused by aggressive emergency stopping maneuvers, which results in a loss of stability. Figure 4.7 provides a representative figure for the performance of each of the approaches. *RAPPIDS* is unable to make sharp turns due to narrow pyramidal flight corridor constraints. *Florence* is unable to enter the passageway due to its conservative collision probability and uncertainty propagation estimates. *Forward-Arc* consistently navigates to the objective, demonstrating robust and safe operation in diverse environmental conditions.

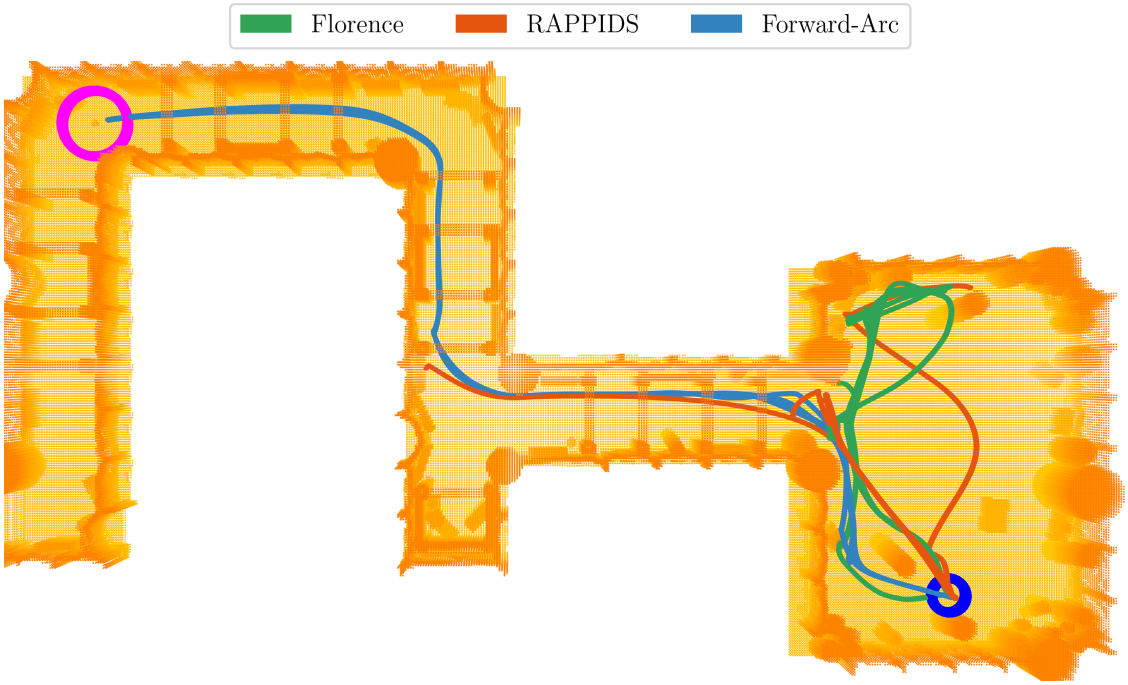


Figure 4.7: Cross-section of the mine environment with overlays of 5 trajectories for each planning approach. The start and goal locations are indicated in blue and magenta, respectively. *Forward-Arc* reaches the goal in all 5 trials. *RAPPIDS* collides in 4 trials and times out in 1 trial. *Florence* collides in 1 trials and times out in 4 trials.

Table 4.1: Unity Custom Environments

Scene	# Goals	Trials	Description
Cave 1	6	90	Natural, Corridors
Cave 2	8	120	Natural, Caverns
Industry 1	6	90	Man-made, Buildings
Industry 2	20	300	Man-made, Factory
Mine	2	30	Man-made, Corridor
Sewer 1	6	90	Man-made, Rooms
Sewer 2	8	120	Man-made, Rooms
Sewer 3	6	90	Man-made, Rooms
Sewer 4	6	90	Man-made, Tunnels
Sewer 5	12	180	Man-made, Rooms
Sewer 6	10	150	Man-made, Tunnels

4.3.2 Obstacle Density Experiments

Next, we study the effect of varying obstacle densities and desired speeds on planner performance. We design a forest-like environment using Poisson disk sampling with a uniform density and cylindrical obstacles with a diameter of 0.75 m. Start and goal locations are positioned 70 meters apart with 10 evenly spaced endpoints per random environment seed (see Fig. 4.8 for example). We run a total of 1800 trials with 50 trials per planner at each combination in desired speed and obstacle density.

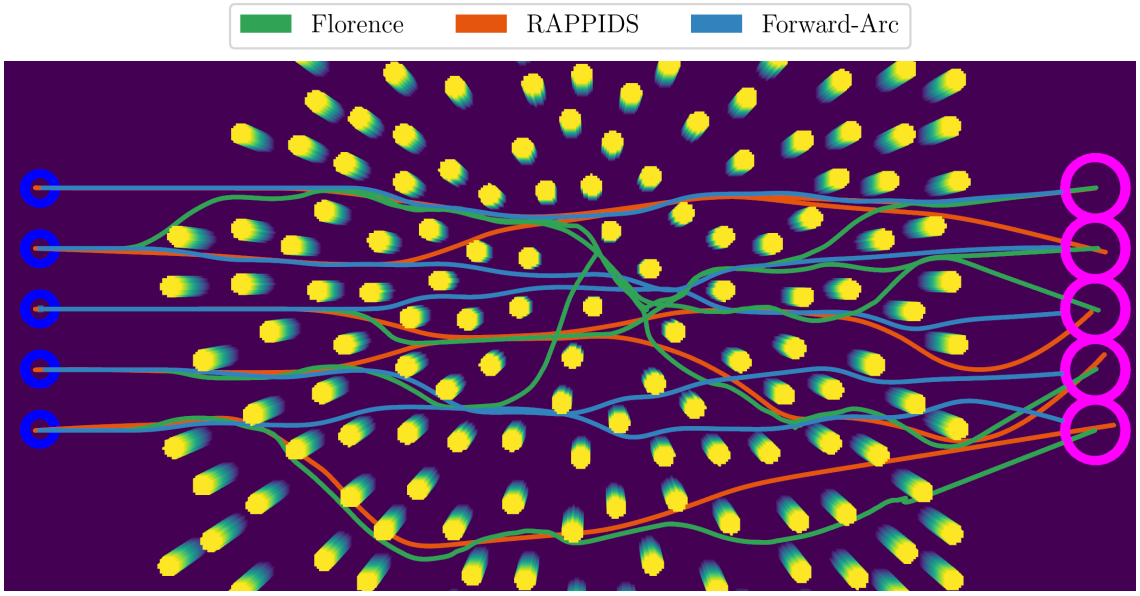


Figure 4.8: Overlay of 5 trajectories for each planner on the ground truth point cloud ($v_{max} = 3m/s$ and $\rho = 0.075$ obstacles/m^2). *Forward-Arc* takes the shortest path towards the goal compared to the baselines.

In Fig. 4.9a, *Florence* struggles with high density environments due to its aggressive stopping maneuvers and conservative collision probability propagation. *RAPPIDS* has an increased flight time at high obstacle densities due to the lack of explicit yaw angle control leading it to take longer paths (Fig. 4.8). As shown in Figs. 4.9a and 4.9b, *Forward-Arc* exhibits the lowest collision rate, flight time, path length, and control effort (integral of jerk squared) and has the highest success rate amongst all variations in obstacle densities and speeds.

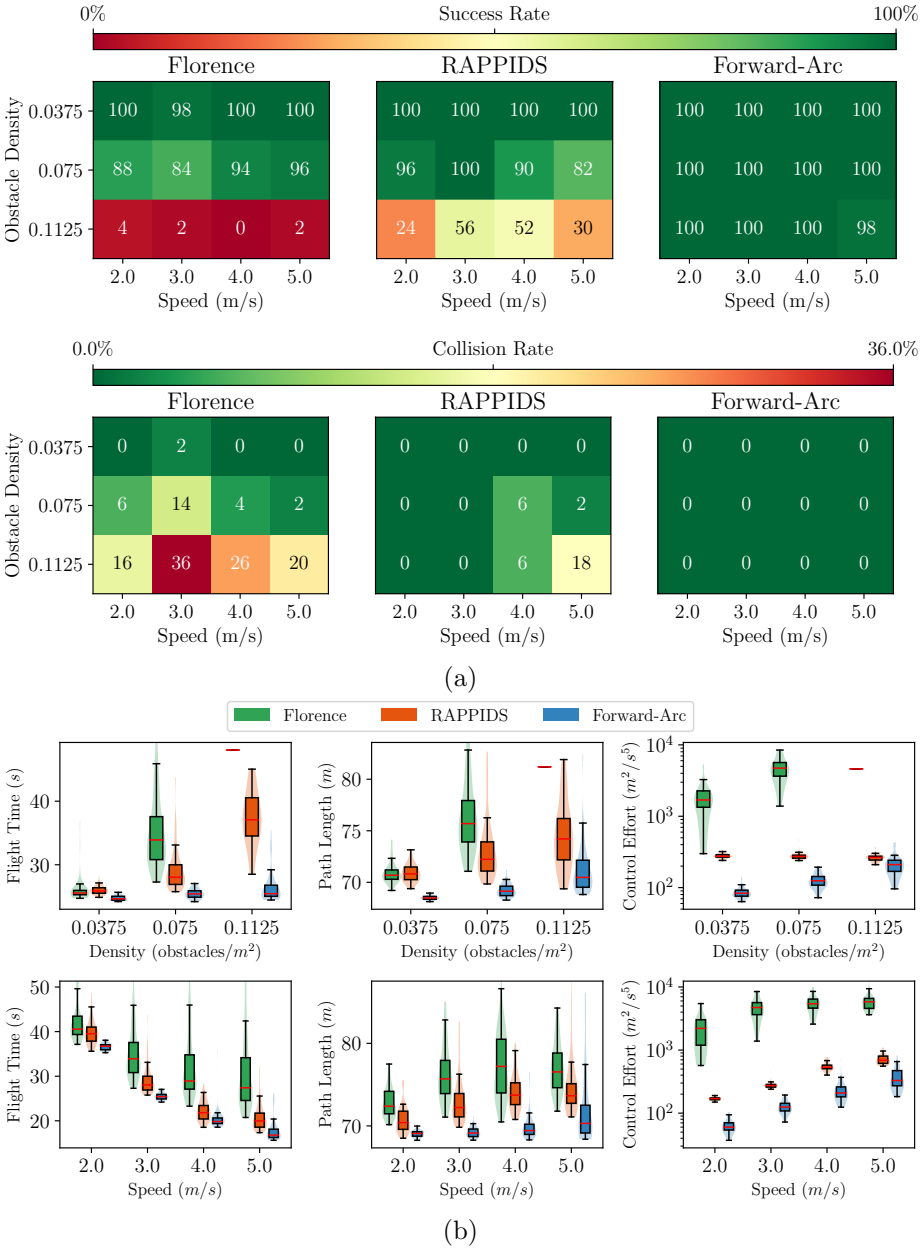


Figure 4.9: (a) success rate and collision rate matrices for simulations that vary obstacle densities and speeds. The *Forward-Arc* approach has higher rates of success at higher obstacle densities and speeds compared to the baseline approaches. (b) distribution of planning performance metrics from successful simulation trials across varying obstacle densities and speeds. Note that a log scale is used for the control effort plots. *Forward-Arc* reaches the objective with lower average flight time, path length, and control effort metrics compared to the baseline approaches.

4.3.3 Hardware Experiments

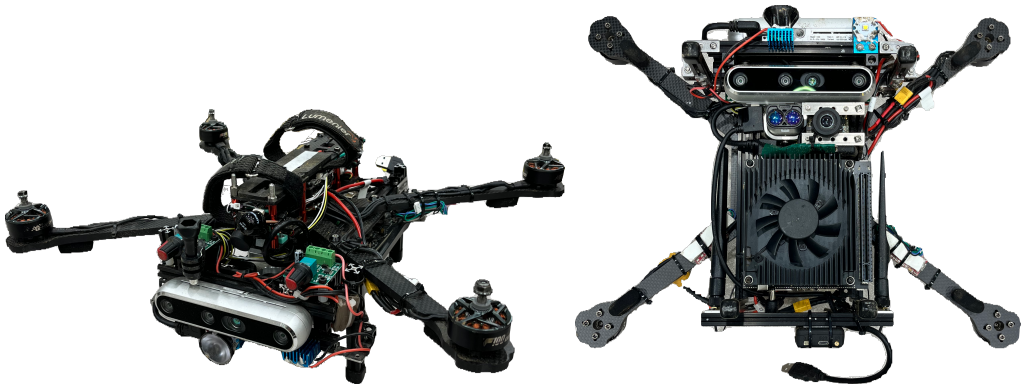


Figure 4.10: Isometric and bottom view of the custom aerial robot used in hardware experiments. The robot is equipped with a RealSense D455 and NVIDIA Orin AGX with 32 GB of RAM.

A custom quadrotor platform is developed for evaluation in hardware experiments. It is equipped with a forward-facing Intel RealSense D455 (Fig. 4.10), a downward-facing LW20 range finder, and a Matrix Vision BlueFox global shutter greyscale camera. The moncular visual inertial navigation system of Yao [42] is used for GPS-denied state estimation. The system is fully autonomous and all calculations are conducted onboard a NVIDIA Orin AGX equipped with 32 GB of RAM. The total system mass is 2.8 kg. The RealSense generate depth images at 30 Hz with a resolution of 424 $\times$ 240. The re-planning algorithm runs at 12 Hz, and maintain a 1 s history of frames. We utilize a cascaded controller (the outer loop runs at 100 Hz), which runs on the NVIDIA Orin. An mRo Pixracer flight controller runs the attitude control loop at a higher frequency with modified PX4 firmware [11].

Experiments are conducted in three environments: (1) an outdoor flight arena, (2) forest, and (3) cave.

Outdoor Drone Arena

11 high speed flight experiments were conducted in an outdoor space with an area of 18 m $\times$ 24 m. Foam boards with dimension 2 m high by 1 m wide are used as obstacles. The flight statistics for the 11 flight experiments are shown in Fig. 4.11

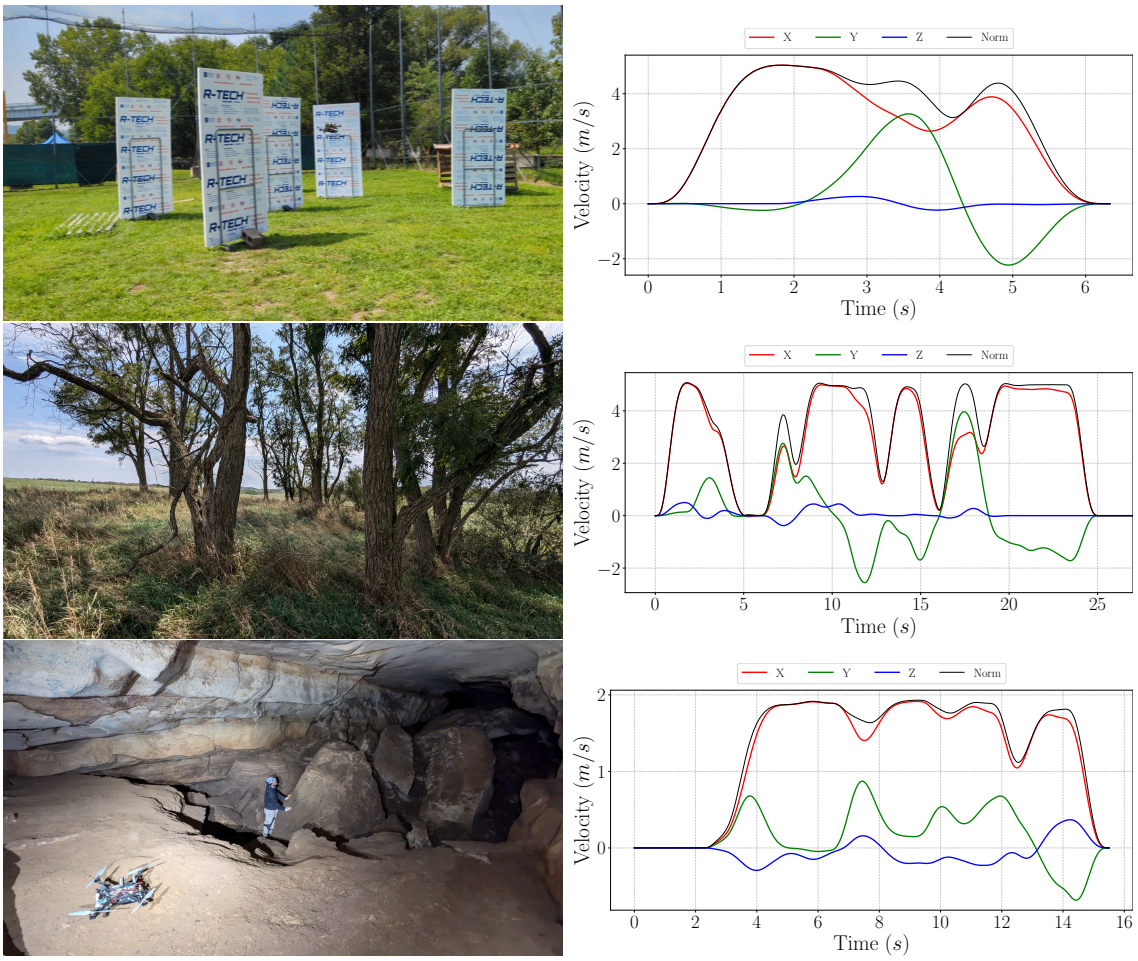


Figure 4.11: Left: flight arena, forest, and cave test environments. Right: reference trajectory velocity profiles for flights 10, 7 and 4, respectively, from Table 4.2. The maximum achieved speed across all trials is 6 m/s.

and Table 4.2. The goal location for all trials was set to be 18 m from the starting point, while desired speed and obstacle locations varied over the trials. The maximum speed achieved was 6 m/s. All 11 trials were successful (over 205 m total path length).

Forest

The approach was tested under the canopy of a stand of trees over 9 trials. Start and goal positions varied with a goal distance of 15 m up to 80 m. The maximum forward velocity was set to 5 m/s. A trial is visualized in Fig. 4.12

Cave

The final set of experiments deployed the robot to a cave in Kentucky. The cave features large rock outcrops and a deep chasm. The goal position varied between 10 m and 20 m away. Over 6 trials, four trials succeeded, one timed out, and one ended in collision. The cave features narrow sections with a vertical gap less than 1.8 m. The desired forward velocity was set between 2 m/s and 3.5 m/s.

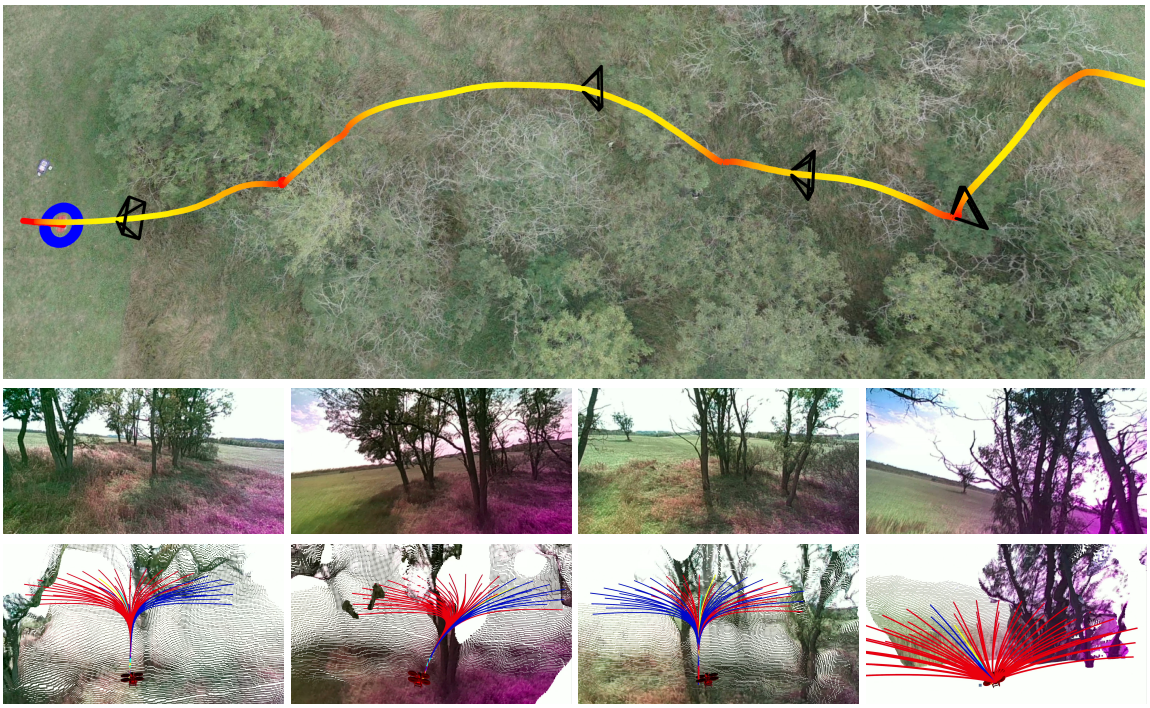


Figure 4.12: Outdoor obstacle avoidance test under tree canopy (forest flight 7 in Fig. 4.11 and Table 4.2). Top: VINS trajectory overlayed on terrain map (low speeds are shown in red and high speeds in yellow). Camera frustums are visualized at four time steps where the robot maneuvers to avoid obstacles. Middle: Onboard RGB image from RealSense D455. Bottom: Robot and forward-arc motion primitive library (red: non-safe primitives, blue: safe primitives, yellow: selected primitive, cyan: stopping primitive).

Table 4.2: Chapter 4 Hardware Flight Trials

Env.	Flight #	Flight Time s	Path Length m	v_{max} m/s	Max Speed m/s	Avg Speed m/s
Outdoor Flight Arena	1	12.08	16.45	2	2.21	1.32
	2	9.62	17.48	3	3.04	1.76
	3	7.01	17.89	4	4.09	2.29
	4	10.71	17.69	3	3.04	1.52
	5	9.40	18.71	4	4.12	1.83
	6	8.12	17.58	4	4.01	2.06
	7	6.94	17.85	5	4.85	2.46
	8	7.65	19.85	5	5.11	2.43
	9	9.22	20.49	5	5.09	2.11
	10	8.36	22.14	5	5.02	2.54
	11	7.13	18.23	6	6.06	2.38
Forest	1	9.73	19.88	3	3.11	1.90
	2	9.68	14.77	3	3.50	1.42
	3♣	11.57	29.91	3	3.30	2.53
	4	9.15	30.36	5	5.13	3.10
	5	18.43	64.20	5	5.20	4.34
	6♣	4.77	15.86	5	5.07	3.25
	7	33.62	91.28	5	5.22	2.58
	8	22.23	87.79	5	5.29	3.81
	9♣	36.40	29.47	5	5.05	0.71
Cave	1	7.79	10.30	2	1.91	1.23
	2	13.98	14.23	2	2.08	0.86
	3	11.05	12.98	2	2.05	1.11
	4	13.96	20.97	2	2.34	1.39
	5♦	11.69	13.45	3	3.43	1.06
	6♠	6.05	12.04	3	3.16	1.87

♣ indicates a crash due to thin obstacle not being detected.

♦ indicates robot did not find a safe route and timed out.

♠ indicates robot crashed due to VINS state estimate drift.

4.3.4 Discussion on failure modes

As shown in Table 4.2 we experienced three failure modes in hardware experiments. First, perception errors due to thin branches that were not detected by the depth sensor led to 3 collisions in the forest environment. Second, without a global planner, the robot got stuck in a dead end in the cave environment where navigation to the goal required backtracking. Lastly, cave flight 6 experienced a collision due to VINS state estimation drift. During execution of a stopping maneuver, particulates from rotor downwash caused incorrect feature matches in the downward-facing camera, which led to the robot tracking the stopping trajectory into the ground instead of at the correct altitude.

Chapter 5

End2End Reactive Navigation

5.1 Introduction

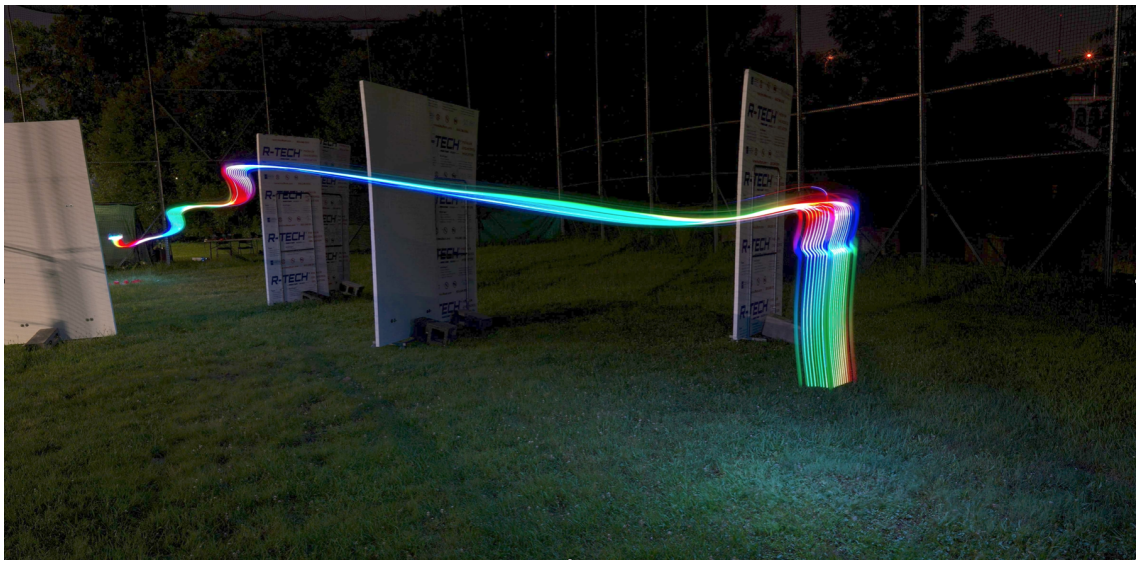


Figure 5.1: Long exposure photo of the end-to-end learning-based policy flying in the dark with foam obstacles. The quadrotor starts from the left and flies at 3 m/s towards a goal on the right 20 m away.

Learning-based navigation has emerged as a powerful alternative to classical module-based approaches, enabling autonomous agents to navigate complex environments

using neural networks. Unlike traditional approaches that decompose the navigation task into separate modules for perception, planning, state estimation, and control, learning-based methods use learned weights to convert raw sensor data into actions. This integration offers several advantages, including reduced planning latency and lower computational overhead, which make it feasible to deploy these systems on lightweight, low-cost, and scalable platforms. Recent work has demonstrated the effectiveness of such methods for high-speed autonomous flight in challenging environments [9, 32, 33].

5.1.1 Contributions

Building on recent progress in end-to-end learning for navigation, this work proposes a training and evaluation pipeline for learning policies that perform both obstacle avoidance and goal-directed flight. Our approach extends the current state-of-the-art differentiable reinforcement learning method by Zhang et al. [9], specifically addressing its limitations in navigating around large obstacles. We make the following contributions:

1. **Improved navigation around large obstacles:** We introduce a yaw prediction objective and leverage a time-of-arrival map as privileged information during training to guide the policy along shortest geodesic paths to the goal. Together, these components help the quadrotor reorient and navigate effectively around large obstacles without requiring an explicit map at test time.
2. **Bridging the sim-to-real gap:** We introduce two key strategies to address the sim-to-real gap. First, we compute desired angular rates from predicted actions, which helps bridge the gap between simplified point-mass dynamics used in simulation and the full rigid-body dynamics of a real quadrotor. Second, we apply domain randomization over parameters, including gravity, to enable the policy to learn a robust feedback loop that compensates for modeling errors, such as a 15% mismatch in nominal thrust.
3. **Simulation in diverse environments:** We evaluate our approach across a range of simulated cluttered environments, demonstrating that the learned policy generalizes well to settings that differ from those seen during training and outperforms the state-of-the-art method in goal-directed navigation. Our

approach achieves an 86% success rate, outperforming baseline strategies by 34%.

4. **Real-world deployment:** We deploy the policy onboard a custom quadrotor in an outdoor obstacle course, including in daytime and nighttime, and under forest canopy. The policy is validated across 20 flights, covering 589 m without collisions at speeds up to 4 m/s.

5.2 Methodology

We develop a navigation and control strategy which takes as input a depth image and the quadrotor state and uses a neural network to predict a thrust and yaw angle. The policy maintains a hidden state which compresses important features from current and past observations into an intermediate representation that is used for future action predictions. This framework enables a lightweight, end-to-end strategy that avoids obstacles while maintaining smooth actions.

5.2.1 Differentiable Dynamics

Following the approach of Zhang et al. [9], the quadrotor navigation problem is formulated as a Markov decision process with a discrete-time dynamical system. As shown in Fig. 5.2, observations, o_k, are generated at each time step from simulated sensor measurements based on the current state, s_k. The policy takes the observation o_k as input and outputs an action a_k. The action we predict is a mass normalized thrust, $a_k \in \mathbb{R}^3$. This thrust vector is the control input to the system dynamics $s_{k+1} = f(s_k, a_k)$, which determines the next state, s_{k+1}. At each time step, the agent also receives a cost function dependent on the current state and action $l_k = l(s_k, a_k)$. Notably, the system dynamics are differentiable with respect to s_k and a_k. This enables us to use Analytical Policy Gradient (APG) methods to train the model. APG methods compute the loss with respect to the policy parameters by applying the chain rule to backpropagate the loss through the dynamics function. This enables very sample efficient training, as even a single sample provides a gradient which can

be used to optimize the policy.

$$v_{k+1} = v_k + \frac{a_k + a_{k+1}}{2}\Delta t$$
$$p_{k+1} = p_k + v_k\Delta t + \frac{1}{2}a_k\Delta t^2 \tag{5.1}$$

While the full quadrotor dynamics model is differentiable, it has been shown that using point mass dynamics results in reduced training time without sacrificing on performance [44]. Therefore the system dynamics model selected is a point mass model that uses a velocity Verlet integration scheme (Eq. (5.1)). Since point mass dynamics does not maintain an explicit orientation, the orientation is determined uniquely by the simulator at every time step. The basis vectors for the body frame are defined such that the body z axis points along the predicted thrust vector a_k and the yaw angle is aligned with the target direction.

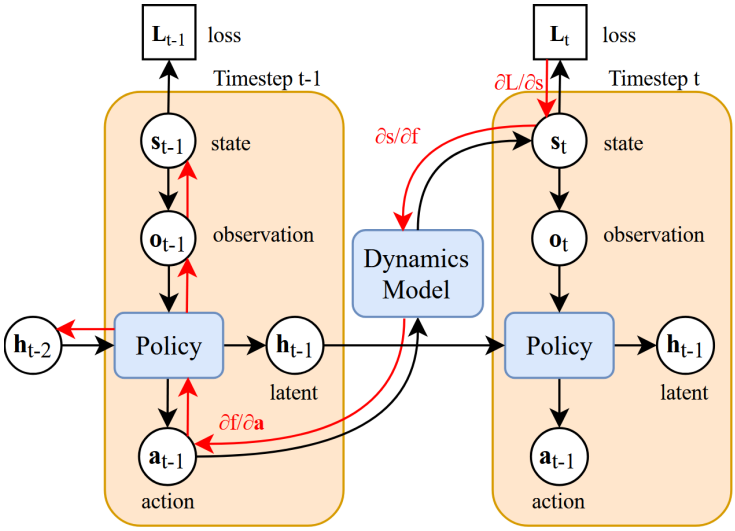


Figure 5.2: Differentiable dynamics enables direct policy updates by performing gradient descent on the loss function.

5.2.2 Network Architecture

The network takes as input a depth image, a target, and the robot's state. Each input is processed by a dedicated feature extractor and projected into a 192-dimensional vector. These vectors are layer-normalized and summed to form the input to a gated

recurrent unit (GRU) cell. The GRU combines this input with the previous hidden state to produce a new hidden state, which encodes relevant features from both current and past observations.

From this latent representation, a linear layer predicts the desired mass-normalized thrust, yaw angle, and current velocity estimate. By maintaining a memory of past observations and actions, the latent representation enables the policy to generate smooth and consistent control outputs.

The state consists of the current velocity and quaternion of the robot. The target is defined by a desired velocity vector v_{target} and a goal importance metric, g_{imp}, which reflects the inverse distance to the goal within a bounded range. g_{imp} provides contextual information about the goal's proximity relative to obstacles in the depth image, helping the network distinguish whether the goal lies in front of or behind an obstacle. All input vectors are expressed in the starting frame of the robot.

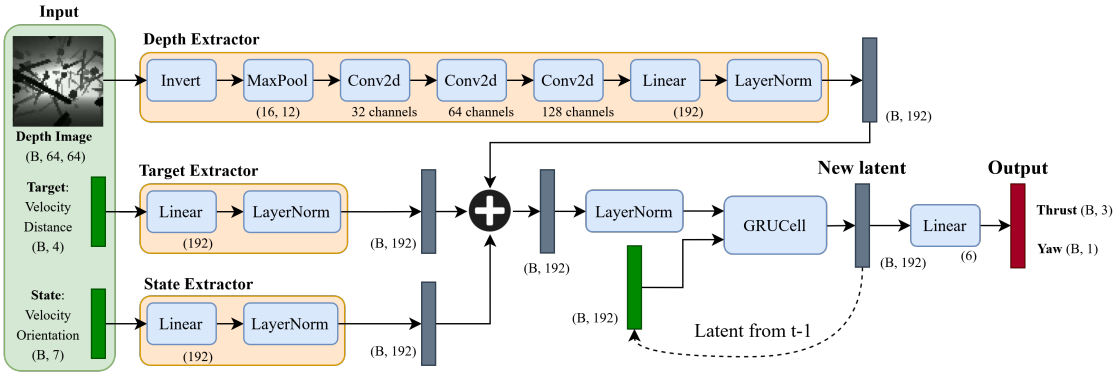


Figure 5.3: The end-to-end planning and control architecture is trained as a single neural network. Separate feature extractors are used to process each input before they are summed together. A GRUCell helps to maintain consistent action predictions over time, as only one action is predicted at each time step.

5.2.3 Loss Functions

This section describes the individual loss (or reward) functions applied at each timestep. Together, these terms guide the training of a stable and effective navigation policy.

Obstacle Avoidance Loss

$$L_{clearance} = \beta_1 \ln(1 + e^{\beta_2(d_k - r)}) \tag{5.2}$$

$$L_{collision} = \frac{1}{T} \sum_{k=1}^{T} v_k^c \max(1 - (d_k - r), 0)^2 \tag{5.3}$$

$L_{clearance}$ is a softplus penalty on the distance d_k to the nearest obstacle, encouraging the robot to maintain a safe clearance where r is the robot radius. In contrast, $L_{collision}$ penalizes the component of velocity v_k^c directed toward the closest obstacle. Together, these losses train the policy to keep a safe distance and reduce forward velocity when approaching obstacles.

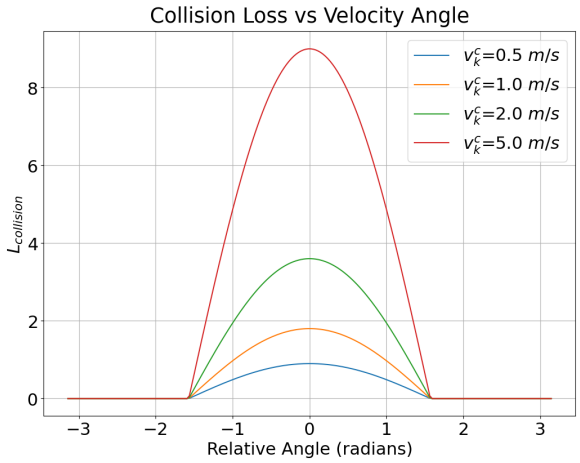


Figure 5.4: Collision loss as a function of the relative angle between v_k^c and the closest obstacle, with velocity and position held constant. The loss increases for states with higher speeds and smaller angles (i.e., when a large component of the velocity is directed towards the obstacle).

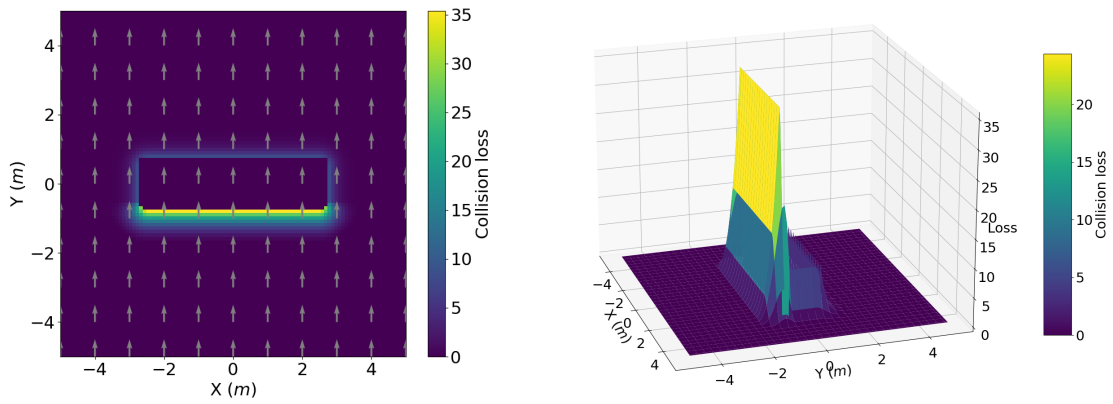


Figure 5.5: Visualization of the combined clearance and collision loss functions in a top-down view of an environment with a single wall obstructing forward motion. The left plot shows a heatmap of the loss values overlaid with uniformly spaced, upward-pointing velocity vectors (gray arrows). The loss increases as the agent approaches the wall. The right plot shows a 3D surface of the same loss slice, highlighting the steep gradients near the obstacle.

Smoothness Loss

$$L_{acc} = \frac{1}{T}\sum_{k=1}^{T}\|a_k\|^2 \tag{5.4}$$

$$L_{jerk} = \frac{1}{T-1}\sum_{k=1}^{T-1}\|\frac{a_{k+1}-a_k}{\Delta t}\|^2 \tag{5.5}$$

$$L_{\omega} = \|\boldsymbol{\omega}\|^2 \tag{5.6}$$

Since the policy only predicts a single action at a time, it is important that the sequence of actions over time form a smooth and stable trajectory. The loss function includes penalties on the L2 norm of linear acceleration and jerk in the inertial frame, as well as angular velocity in the body frame. Specifically, L_{acc} encourages stability near hover at the target (see Section 5.3.2), while L_{ω} reduces abrupt yaw changes by discouraging large angular accelerations.

Target Velocity Loss

$$L_v = \frac{1}{T}\sum_{k=1}^{T} \text{Smooth L1}\left(\|v_k^{set} - \bar{v}_k\|_2, 0\right) \tag{5.7}$$

$$L_{vmax} = \frac{1}{T}\sum_{k=1}^{T} \max(\|v_k\| - \|v_{max}\|)^2 \tag{5.8}$$

Equation (5.7) encourages the policy to track the target velocity v_k^{set}. To allow flexibility for obstacle avoidance while maintaining long-term progress, we use a 2-second moving average of the actual velocity, $\bar{v}_k$, in the loss. This smooths short-term deviations and helps the policy stay aligned with the overall velocity objective. Consistent with findings in Zhang et al. [9], this averaging also reduces high-frequency velocity oscillations during rollouts under full rigid body dynamics. Additionally, we introduce a new penalty term L_{vmax} for speeds exceeding the maximum target velocity v_{max}, which helps bound the predicted thrusts and prevents policy divergence.

Yaw Prediction Loss

$$L_{\text{yaw}} = -\frac{1}{T}\sum_{k=1}^{T} \hat{\mathbf{d}}_k \cdot \mathbf{d}_k^{\text{yaw}} \tag{5.9}$$

The yaw alignment loss L_{yaw} is defined as the negative inner product between the desired direction and the current yaw direction $\mathbf{d}_k^{\text{yaw}}$. The desired yaw direction $\hat{\mathbf{d}}_k$ is set to $\tilde{\mathbf{v}}_k$, which denotes the exponentially weighted moving average of the velocity.

Geodesic Gradient Loss

We observe that existing obstacle avoidance and collision loss terms are often insufficient for guiding the robot around large obstacles. Although the gradients from these losses push the robot away from surfaces, they do not provide meaningful guidance on how to reroute around obstacles. This limitation is particularly evident with large obstacles, where local collision losses primarily penalize proximity and speed toward the obstacle. While such losses may help the robot come to a stop before a collision, they do not encourage progress or escape, increasing the risk of the robot becoming

stuck.

To encourage the robot to safely navigate around obstacles, we look towards geodesic distance fields, defined as minimum time paths from anywhere in free space to a specific goal location. Similar to Meijer et al. [45], we use the geodesic distance field as a training signal. However, [45] leverages the gradient of this field indirectly by learning a weighted distribution over directional rays and combining the top-k predictions to produce an action. This strategy can struggle in multi-modal settings where the weighted average may fall between modes and fail to align with any high-likelihood direction. In contrast, we directly regress the velocity to follow the gradient of the geodesic field, providing a more consistent and training signal.

One limitation of the geodesic distance fields used in Meijer et al. [45] is that they do not account for proximity to obstacles. As a result, the generated fields can lead to overly idealistic solutions that pass through narrow gaps, which may be unsuitable for real robots. To address this, we introduce a cost map that heavily penalizes regions near obstacles. This biases the wavefront propagation so that the resulting time-optimal paths maintain safer distances from obstacles (Fig. 5.7).

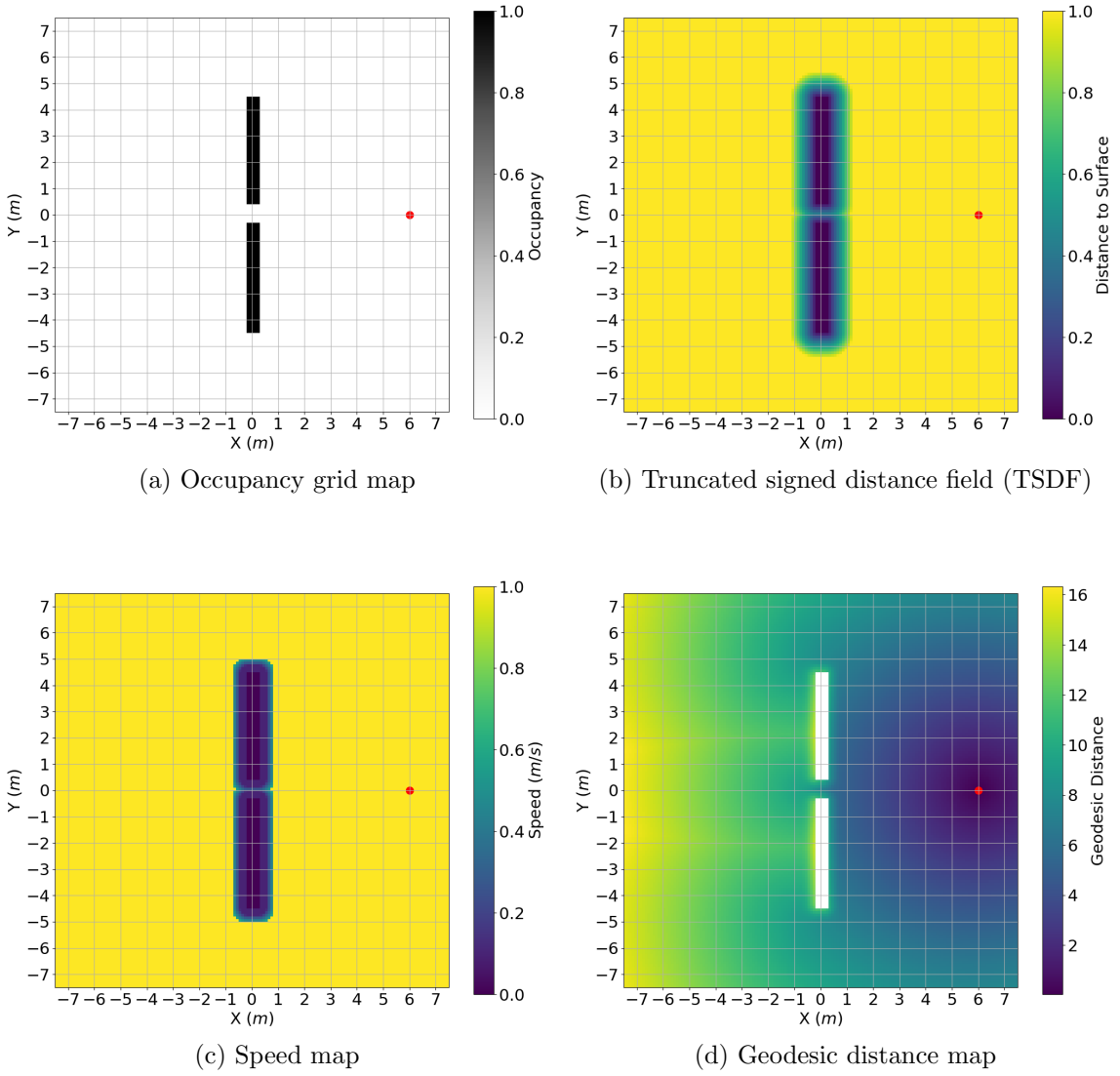
In Fig. 5.6, we explain the process of computing the geodesic distance map. First a 3D occupancy grid map is obtained by sampling the occupancy grid map at a fixed resolution. Next a Euclidean distance transform on the occupancy grid map is used to obtain a Truncated signed distance field (TSDF). The TSDF enables us to design a speed cost map as a function of the distance to the nearest obstacle. Using Eq. (5.10), we obtain an expression for how fast a wavefront can propagate through each cell. When the cell is at least d_{safe} from an obstacle, the wavefront can propagate at 1 m/s, whereas within d_{safe} distance of the obstacle, the travel speed is proportional to the distance to the obstacle. Finally we use Fast Marching Method [46] implemented in the *scikit-fmm* library [1].

[1] https://github.com/scikit-fmm/scikit-fmm

$$f(d) = \begin{cases} 0 & \text{if } d < 0 \\ md + (v_{slow} - m * r) & \text{if } d <= d_{safe} \\ 1 & \text{else } d > d_{safe} \end{cases} \tag{5.10}$$

$$m = \frac{d_{safe} - v_{slow}}{d_{safe} - r} \tag{5.11}$$

where r is the robot radius, d is the distance from the nearest obstacle, d_{safe} is the threshold distance from an obstacle which is considered safe, and v_{slow} is the speed to travel when in close proximity to an obstacle.



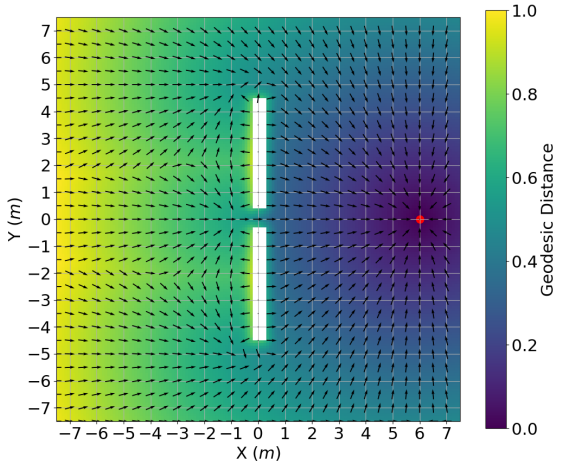
(a) Occupancy grid map

(b) Truncated signed distance field (TSDF)

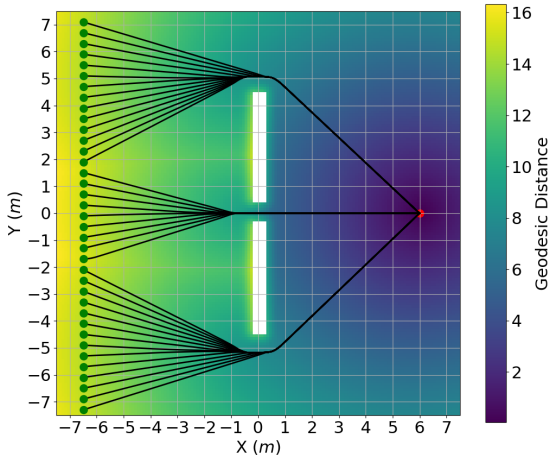
(c) Speed map

(d) Geodesic distance map

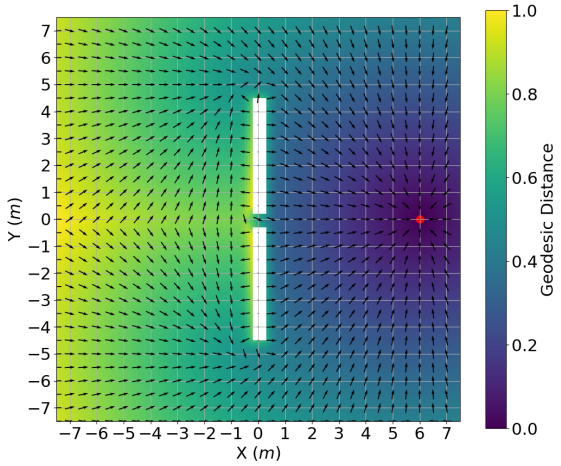
Figure 5.6: Computation of the geodesic distance field. (a) Occupancy grid map with a 9 m-long wall and a 70 cm gap, using a 10 cm resolution. (b) Truncated signed distance field (TSDF) to the nearest obstacle. (c) Speed map computed from the TSDF using Eq. (5.10). (d) Geodesic distance (arrival time) computed using the fast marching method (FMM) with respect to the goal position (red dot).



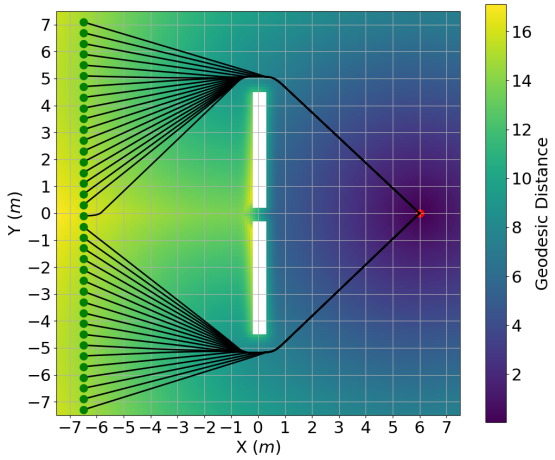
(a) FMM geodesic gradient (70 cm gap)



(b) Shortest paths along geodesic (70 cm gap)



(c) FMM geodesic gradient (30 cm gap)



(d) Shortest paths along geodesic (30 cm gap)

Figure 5.7: Geodesic distance fields with overlaid numeric gradients and shortest paths from multiple starting points (green dots) to a fixed target location (red dot). In the top row, where the gap in the wall is wider than the robot's radius, some shortest paths pass through the gap. In the bottom row, where the gap is narrower than or equal to the robot's radius, all shortest paths go around the wall.

Total Loss

Table 5.1: Loss function parameters

Parameter	Value
λ_{acc}	0.01
λ_{jerk}	0.001
λ_{ω}	0.3
λ_{v}	4.0
λ_{vmax}	1.0
$\lambda_{collision}$	6.0
$\lambda_{clearance}$	6.0
β_1	2.5
β_2	-6.0

$$
\begin{aligned}
L = \big(&\lambda_{acc} L_{acc} \\
&+ \lambda_{jerk} L_{jerk} \\
&+ \lambda_{\omega} L_{\omega} \\
&+ \lambda_{v} L_{v} \\
&+ \lambda_{vmax} L_{vmax} \\
&+ \lambda_{clearance} L_{clearance} \\
&+ \lambda_{collision} L_{collision}\big)
\end{aligned} \tag{5.12}
$$

The total loss (Eq. (5.12)) is a linear combination of all the loss terms using the coefficients from Table 5.1.

5.2.4 Training

The policy is trained entirely in simulation using reinforcement learning with a custom GPU accelerated simulator to render depth images and parallelize training. While several quadrotor simulators exist that suit our needs, we develop ours on top of

VisFly [47].

Rendering and collision checking is performed using Habitat-Sim [48] in an environment with randomly generated primitives as obstacles (spheres, cylinders, and cubes). The network is trained with a batch size of 32 for 10K iterations using back propagation through time (BPTT) to accumulate and propagate gradients. We found that using curriculum training with the collision loss initially set to $\lambda_c = 0$ and slowly increasing environment density ensures stable training. The policy loss converges within minutes but the entire policy takes several hours in order to become proficient at obstacle avoidance.

Training Level	Scene	λ_c	ρ_1	ρ_2	Iterations
0	Box 2	0	0	0	500
1	Box 2	2	0.1	0	5000
1	Box 2	2	0	0.03	5000
2	Box 2	2	0.15	0	5000

Table 5.2: Curriculum Training

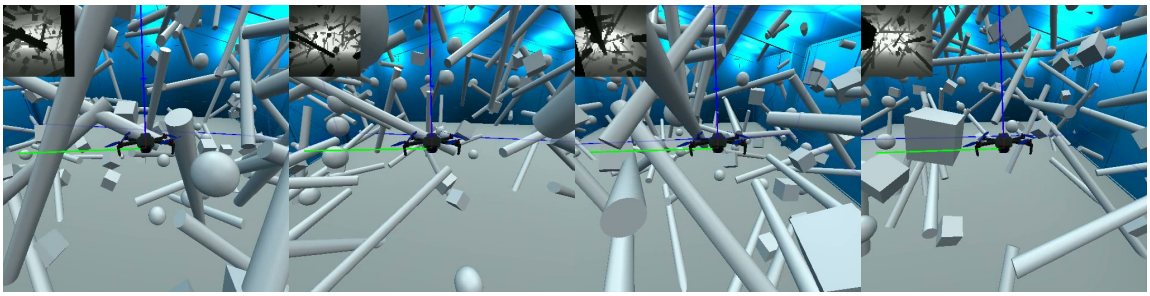


Figure 5.8: An example of four randomized training environments. Each agent receives a randomized start and goal location. The loss functions ensures the policy learns to predict a smooth sequence of actions that avoid obstacles in the depth image. The depth image observations (black and white) are inset in the top left of each scene. Demo Video: https://youtu.be/F4kVOh997k8.

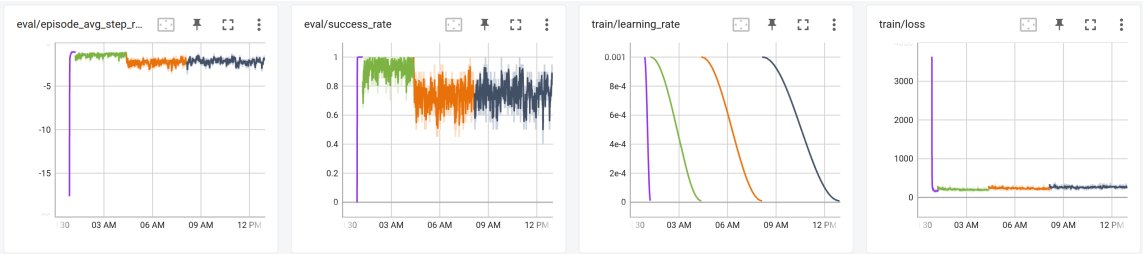


Figure 5.9: Plots of tensorboard curriculum training (reward, success rate, learning rate, and loss over time). Each block in the training curriculum is indicated by a different color (purple, green, orange, grey). As training progresses and gets more difficult with more obstacles, the success rate slowly decreases.

5.2.5 Bridging the Sim-to-real gap

A challenge in deploying policies trained exclusively in simulation to hardware is ensuring the approach is robust to modeling errors and other differences between the simulation environment and real-world environment. We bridge the sim-to-real gap by applying two key strategies. First, we account for the rotational dynamics differences between the simulated point mass dynamics and real-world rigid body dynamics by performing body rate attitude control. Second, we apply domain randomization over parameters, including gravity, to enable the policy to learn a robust feedback loop that compensates for modeling errors, such as a 15% mismatch in nominal thrust.

Body Rate Attitude Control

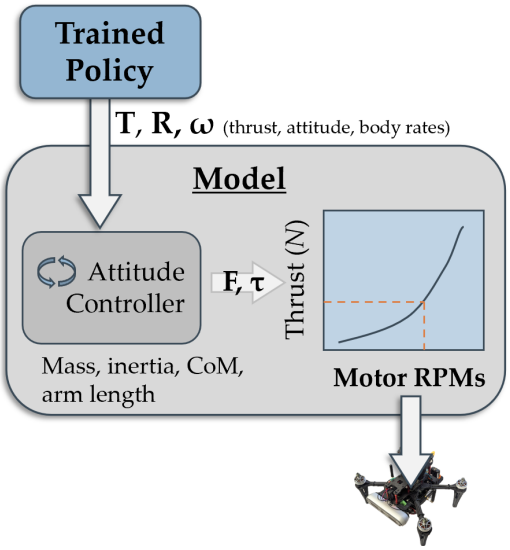


Figure 5.10: Diagram of the deployed policy integrated with the full quadrotor dynamics model. The trained policy outputs a predicted thrust vector and yaw angle, which are used to compute the desired collective thrust, attitude, and body rates. These targets are then tracked by an onboard attitude controller. Note that the deployment dynamics include full rotational dynamics, whereas the training environment assumes simplified point-mass dynamics.

In the standard cascaded control architecture, an outer-loop position controller computes reference trajectories for an inner-loop attitude controller. In our approach, the policy replaces the role of a PD-type position controller, directly predicting mass-normalized thrust and yaw relative to the robot's initial orientation that are converted into cascaded attitude commands.

The inner loop attitude control law [49] is implemented as a PD-type controller that tracks the desired collective thrust F_d, orientation $\mathbf{R}_d$, and body rates $\boldsymbol{\omega}_d$:

$$\boldsymbol{\alpha} = -K_R e_R(\mathbf{R}, \mathbf{R}_d) - K_\omega(\boldsymbol{\omega} - \mathbf{R}^T \mathbf{R}_d \boldsymbol{\omega}_d) + \boldsymbol{\alpha}_d - \boldsymbol{\alpha}_{dist} \tag{5.13}$$

$$\boldsymbol{\tau} = \mathbf{J}\alpha + \mathbf{r}_{cm} \times \mathbf{F} \tag{5.14}$$

Here, $e_R(\mathbf{R}, \mathbf{R}_d)$ is the attitude error function from [13], $\mathbf{J}$ is the moment of inertia matrix, and $\mathbf{r}_{cm}$ is the vector from the center of mass to the geometric center. The

feedforward and disturbance compensation terms $\boldsymbol{\alpha}_d$ and $\boldsymbol{\alpha}_{dist}$ are set to zero.

The desired rotation matrix $\mathbf{R}_d$ is constructed such that the body $\mathbf{z}_b$ axis aligns with the predicted thrust vector $\mathbf{F}$. The body $\mathbf{z}_x$-axis is then chosen to align with the projection of the desired yaw direction ψ_d onto the plane orthogonal to $\mathbf{z}_b$.

The desired body rates $\boldsymbol{\omega}_d$ are computed by approximating the Euler angle rates then computing body angular velocities using the Euler angle Jacobian. First a relative rotation between two consecutive desired attitudes $\mathbf{R}_{d1}$ and $\mathbf{R}_{d2}$ is used to approximate the time derivative of the Euler angle orientation. Specifically, we compute the relative rotation matrix and extract the corresponding ZYX Euler angle derivatives:

$$\mathbf{R}_{rel} = \mathbf{R}_{d1}^T \mathbf{R}_{d2} \tag{5.15}$$

$$\phi, \theta, \psi = \text{Euler\_ZYX}(\mathbf{R}_{rel}) \tag{5.16}$$

$$\dot{\phi} = \frac{\phi}{\Delta t} \tag{5.17}$$

$$\dot{\theta} = \frac{\theta}{\Delta t} \tag{5.18}$$

$$\dot{\psi} = \frac{\psi}{\Delta t} \tag{5.19}$$

These Euler angle derivatives are then mapped to body rates using the Euler angle Jacobian $\mathbf{J}_{Euler}(\phi, \theta)$:

$$\boldsymbol{\omega}_d = \mathbf{J}_{Euler}(\phi, \theta) \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{5.20}$$

$$= \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \sin(\phi)\cos(\theta) \\ 0 & -\sin(\phi) & \cos(\phi)\cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{5.21}$$

Including the derivative term $\boldsymbol{\omega}_d$ significantly improves control response, enabling the attitude controller to achieve the desired orientation with negligible latency in both simulation and real-world flights. In contrast, controllers that rely solely on

proportional feedback, such as in [9], exhibit noticeable control lag, which must be accounted for during policy training.

Domain Randomization

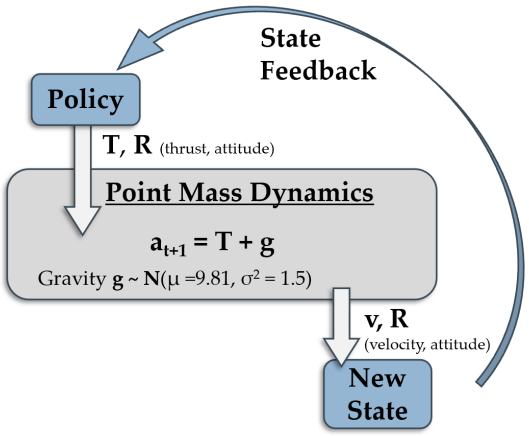


Figure 5.11: Closed-loop feedback during training via gravity randomization. By sampling the gravity constant from a normal distribution at the start of each rollout, the policy learns to adjust its thrust predictions based on its observed velocity, minimizing smoothness loss. This enables the policy to correct for quadrotor modeling errors, such as compensating for reduced thrust due to battery voltage drop-off.

In a cascaded control architecture, our policy replaces the outer-loop position controller and must compensate for modeling errors arising from imperfect system identification (e.g., inaccurate motor thrust characterization).

To ensure the policy is platform-agnostic, the policy is trained to output mass-normalized thrust, meaning we predict the collective thrust in units of acceleration m/s^2. Converting this normalized thrust to actual motor commands depends on characterization of parameters such as mass, torques, thrust-to-RPM coefficients, and battery voltage. Inaccuracies in these parameters can introduce steady-state errors between expected and actual thrust outputs.

To increase robustness to such parametric uncertainties, we apply domain randomization during training. Specifically, we randomize the gravitational constant g at the start of each rollout by sampling from a normal distribution (Table 5.3). This

forces the policy to adapt its thrust outputs dynamically based on velocity observations, as smoothness loss penalizes unintended accelerations. Through this process, a closed-loop feedback mechanism emerges, enabling the policy to compensate for modeling discrepancies, such as errors in mass and thrust coefficients. Policies trained without gravity randomization fail to exhibit this adaptive behavior (Section 5.3.2).

Beyond gravity randomization, we further broaden the training distribution to enhance generalization across domain shifts. We perturb the robot's initial position using a cylindrical distribution and vary the target velocity across rollouts. We also perturb the state inputs to the policy to simulate sensor noise. The randomized parameters and their distributions are summarized in Table 5.3.

Value	Distribution	Parameters
Gravity	$g \sim \mathcal{N}(\mu, \sigma^2)$	$\mu = 9.81,\ \sigma = 1.5$
Initial height	$z \sim \mathcal{U}(z_{\min}, z_{\max})$	$z_{\min} = 0.5,\ z_{\max} = 2.5$
Initial x, y	$(x, y) = (r\cos\theta, r\sin\theta), \theta \sim \mathcal{U}(0, 2\pi)$	$r = 12.5$
Target speed	$v_{target} \sim \mathcal{U}(v_{\min}, v_{\max})$	$v_{\min} = 1, v_{\max} = 5$
Velocity noise	$v_{noise} \sim \mathcal{N}(\mu, \sigma^2) \in \mathcal{R}^3$	$\mu = 0, \sigma = 0.05$
Rotation noise	$r_{noise} \sim \mathcal{N}(\mu, \sigma^2) \in \mathcal{R}^3$	$\mu = 0, \sigma = 0.02$

Table 5.3: Domain randomization and parameters.

5.3 Experiments

5.3.1 Simulation Experiments

We perform simulation experiments using the simulation environments from Section 4.3.1. The environments include indoor and outdoor environments including areas with narrow passageways and corridors. We compare our proposed planner against two baselines. First is Zhang et al. [9], which we call Back to Newton's Laws (BNL). We train their policy using the publically available codebase in [9]. Second, we compare against an ablated version of our policy (yaw w/o geodesic) which is trained without the geodesic distance loss.

The planner performance is evaluated by measuring the success rate and failure

modes (collisions and timeouts). The outcome of a trial is considered a success if the planner reaches the target location within a fixed time duration and without collisions. Otherwise the flight is considered a timeout or in collision. The results are reported in Fig. 5.12.

Our proposed approach outperforms the baseline policies in terms of success rate (86%) and has the lowest collision rate. In contrast, both baseline approaches have a significantly higher proportion of collisions and timeouts. Figure 5.13 provides a representative figure demonstrating some of the failure modes for the baseline approaches. BNL [9] is unable to avoid large, flat obstructions that require flying sideways or turning. The ablated model (yaw w/o geodesic) can sometimes get stuck in complex environments. In contrast, our model is able to yaw around large obstacles and use the geometric information learned from the geodesic loss to find paths through free space that lead to the goal.

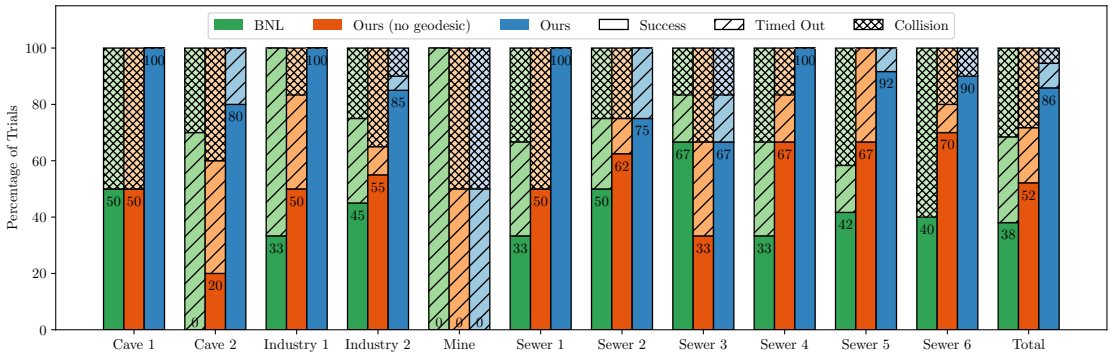
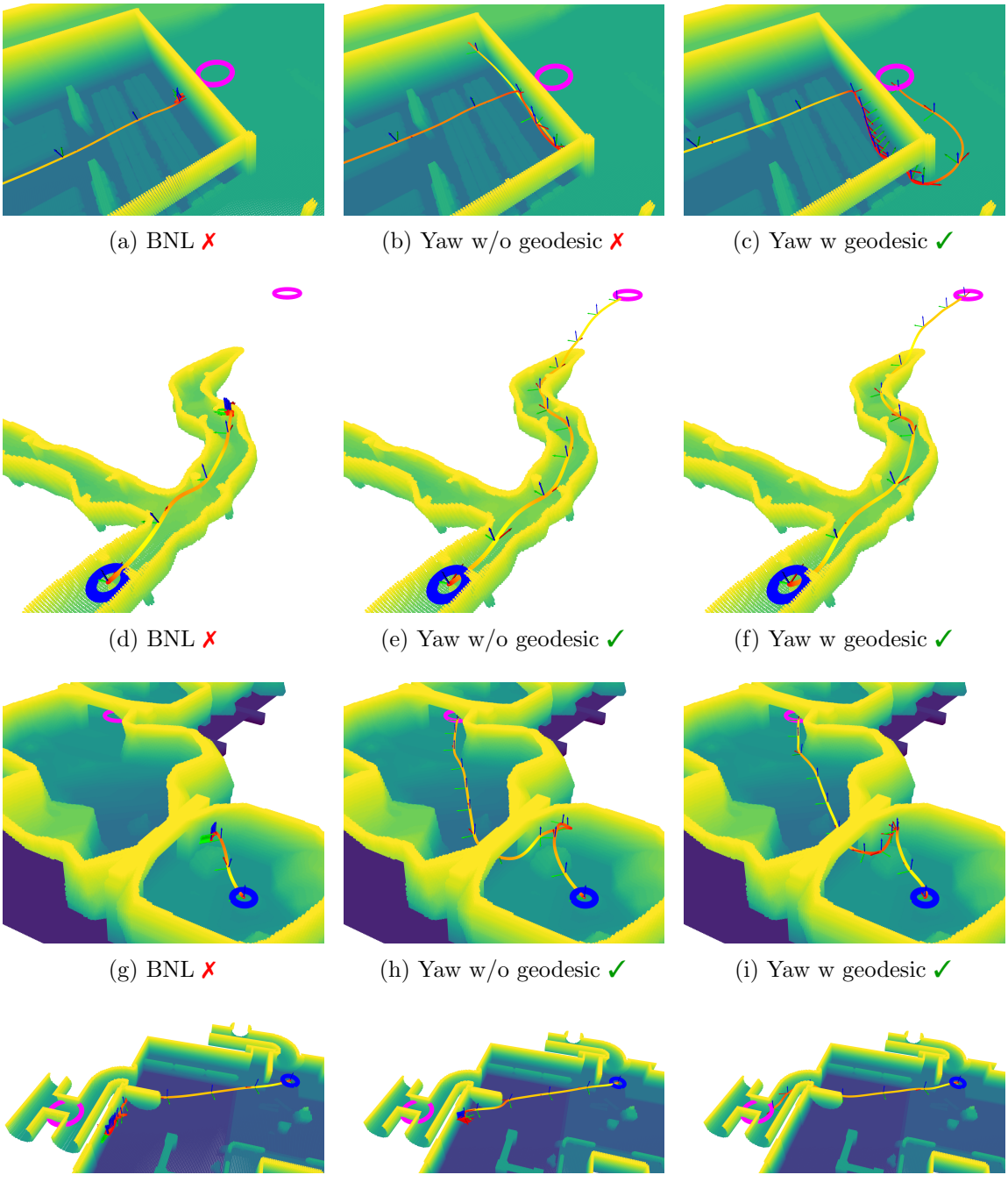


Figure 5.12: Planner success rate and failure modes across the environments detailed in Table 4.1. The proposed method (*Ours*) achieves the highest success rate and lowest collision rate compared to the baseline [9] and the ablated policy trained without geodesics.



(a) BNL ✗ (b) Yaw w/o geodesic ✗ (c) Yaw w geodesic ✓

(d) BNL ✗ (e) Yaw w/o geodesic ✓ (f) Yaw w geodesic ✓

(g) BNL ✗ (h) Yaw w/o geodesic ✓ (i) Yaw w geodesic ✓

(j) BNL ✗ (k) Yaw w/o geodesic ✗ (l) Yaw w geodesic ✓

Figure 5.13: Trajectory profiles overlaid on ground-truth point clouds, with a cross-section shown for clarity. Trajectories are colored by speed (red to yellow, up to $v_{\max} = 3$ m/s), with body frames every 2 seconds. Start and goal are marked in blue and magenta, respectively. Successful trials are marked with a ✓. Environments (top to bottom): *Industry 2*, *Cave 1*, *Cave 2*, *Sewer 1*.

5.3.2 Hardware Experiments

A custom quadrotor platform was developed to support hardware experimentation. The vehicle spans 15 cm from rotor to rotor and is equipped with a forward-facing Intel RealSense D456 depth camera, a downward-facing LW20 range finder, and a Matrix Vision BlueFox global shutter greyscale camera. All onboard computation is performed by an NVIDIA Orin NX module with 16 GB of RAM. The system has a total mass of 1.7 kg. A TBS Lucid H7 flight controller running customized Betaflight firmware [2] provides IMU data at 1000 Hz over a 1M baud serial connection. This high-rate communication enables the onboard attitude controller, running on the Orin NX, to transmit motor commands to the flight controller at the same 1000 Hz frequency. For GPS-denied state estimation, we employ a monocular visual-inertial navigation system developed by Yao Yao [42]. The RealSense camera generates depth images at 60 Hz with a resolution of 640 × 480 pixels.

Correcting for Modeling Error

To evaluate the effectiviveness of domain randomization in compensating for modeling errors, we perform an ablation study comparing policies trained with and without randomized gravity during training. The goal is to assess whether gravity randomization allows the policy to correct for quadrotor modeling errors as hypothesized in Section 5.2.5, such as thrust loss due to battery voltage drop-off or inaccuracies in system identification.

A key source of modeling error lies in the thrust-to-RPM mapping function. This function is empirically derived by mounting a motor and propeller onto a thrust stand, measuring thrust at varying RPMs, and fitting a quadratic curve to the data. However, benchtop conditions often differ from in-flight conditions due to factors such as battery voltage drop and airflow disturbances near the quadrotor frame. These unmodeled discrepancies result in steady-state thrust errors. For instance, to hover in place, the quadrotor ideally requires a collective thrust of $1g$, yet we observe that it needs approximatelya $1.15g$ (Fig. 5.15) to maintain altitude, a 15% mismatch.

To address these errors, we introduce gravitational constant randomization during training, sampling g from a normal distribution (see Section 5.2.5 for distribution

[2]Betaflight Open Source Flight Controller: https://github.com/betaflight/betaflight

details). We hypothesize that this domain randomization forces the policy to learn a closed-loop feedback mechanism that can adapt thrust outputs based on observed system states.

Both policies (trained with and without gravity randomization) are evaluated in a task where a finite state machine governs the take-off, autonomous hover, and landing phases. During the autonomous phase, the policy is provided a target velocity vector proportional to the quadrotor's distance from a fixed goal setpoint.

In Fig. 5.14, the policy trained without gravity randomization initially outputs exactly $1g$ of thrust, which is not enough to hover at the goal setpoint, resulting in a large control error and the quadrotor hovering near the ground.

Conversely, the policy trained with gravity randomization (Fig. 5.15) dynamically adjusts its thrust, increasing output up to 1.3 as necessary, successfully maintaining hover at the desired altitude.

These results validate our hypothesis: gravity randomization during training induces a feedback mechanism in which the policy adaptively adjusts its thrust outputs based on observed system states (e.g., velocity), allowing it to correct for modeling inaccuracies during deployment.

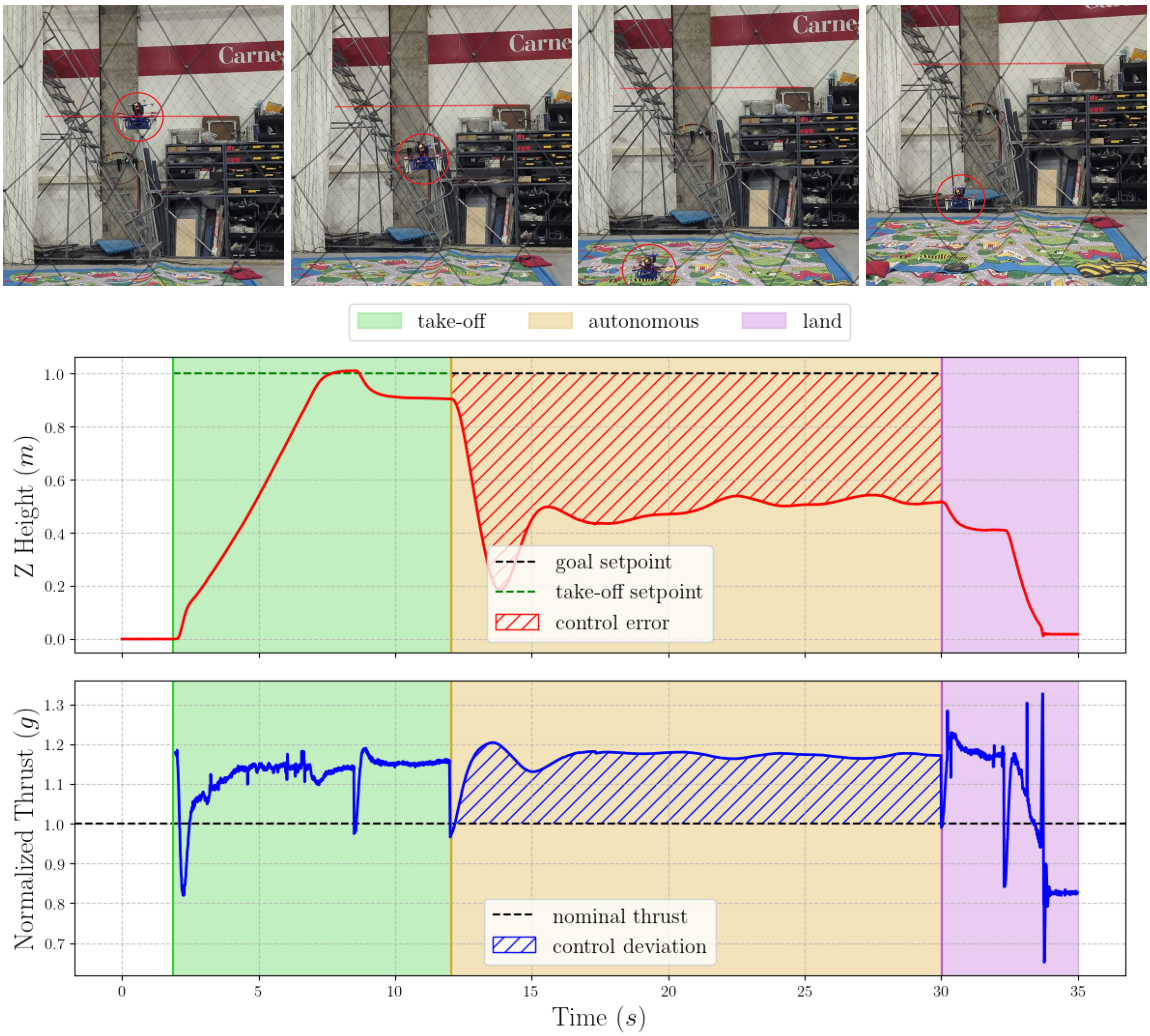


Figure 5.14: Flight using policy trained without gravity randomization. Top row: snapshots of the quadrotor initially starting at the goal setpoint in the takeoff phase, but when entering autonomous mode it quickly loses altitude and stabilizes near the floor. Middle row: The z height of the quadrotor as measured by a motion capture system with a large control error as shown by the red shaded region. Bottom row: The normalized thrust predicted by the policy initially is not high enough to hover, resulting in a loss in altitude.

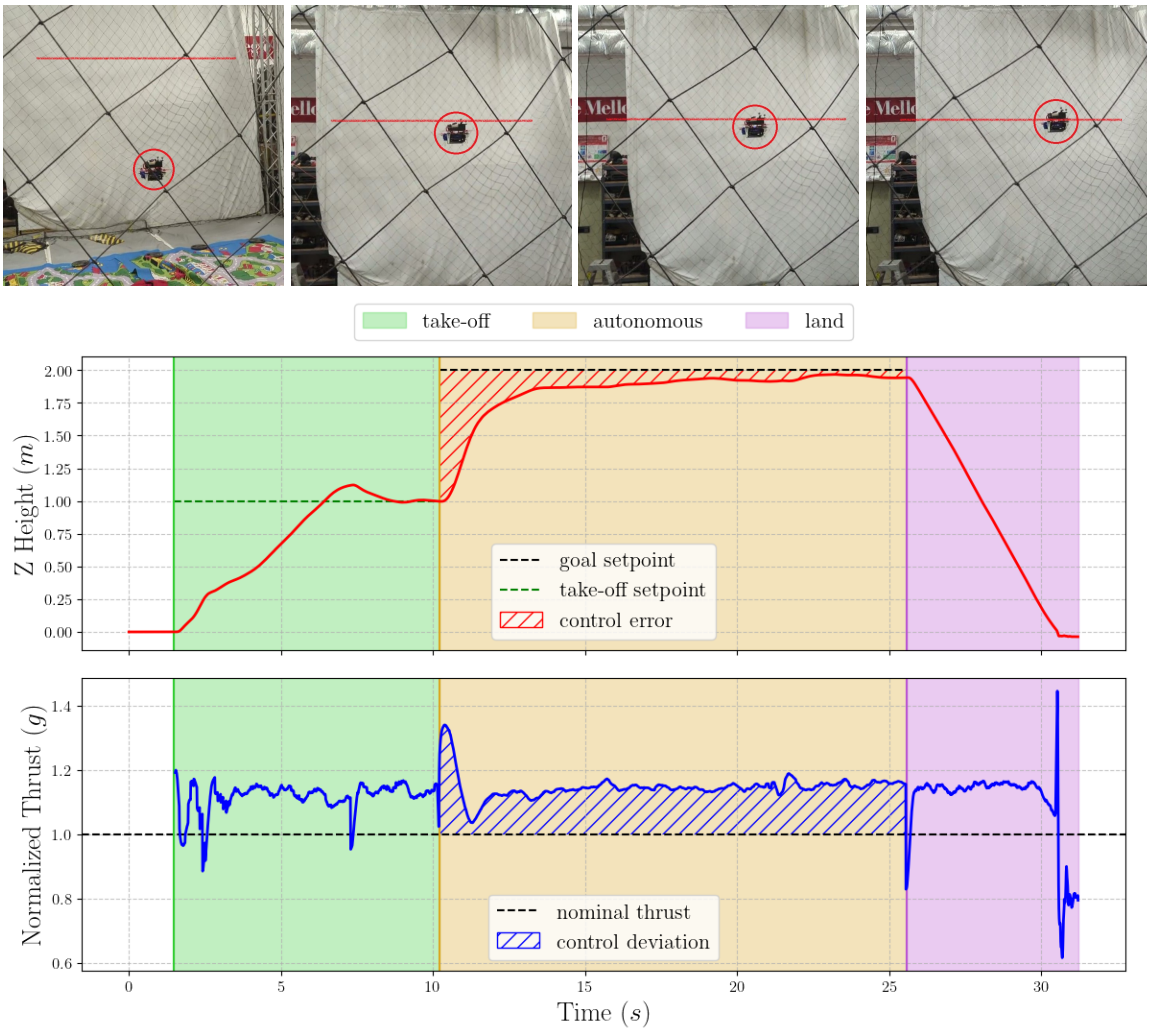


Figure 5.15: *Flight using policy trained with gravity randomization.* Top row: snapshots of the quadrotor initially at the takeoff setpoint, then quickly increasing altitude to stabilize at the goal setpoint. Middle row: The z height of the quadrotor as measured by a motion capture system with a diminishing control error as shown by the red shaded region. Bottom row: The normalized thrust predicted by the policy has a high initial thrust around $1.3g$ and then the control deviation stabilizes.

Outdoor Drone Arena

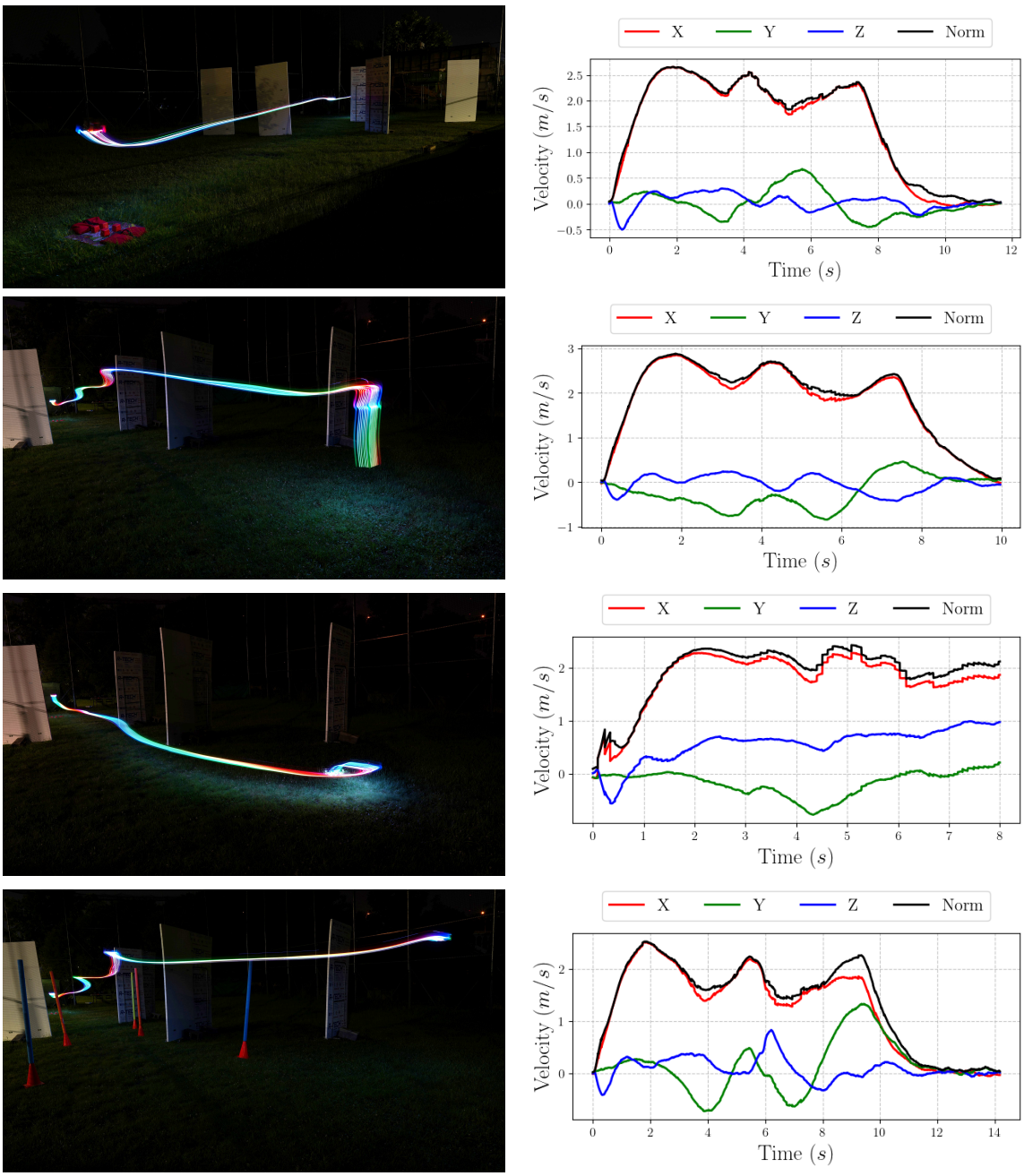


Figure 5.16: Outdoor obstacle avoidance tests at night with long exposure.

Forest

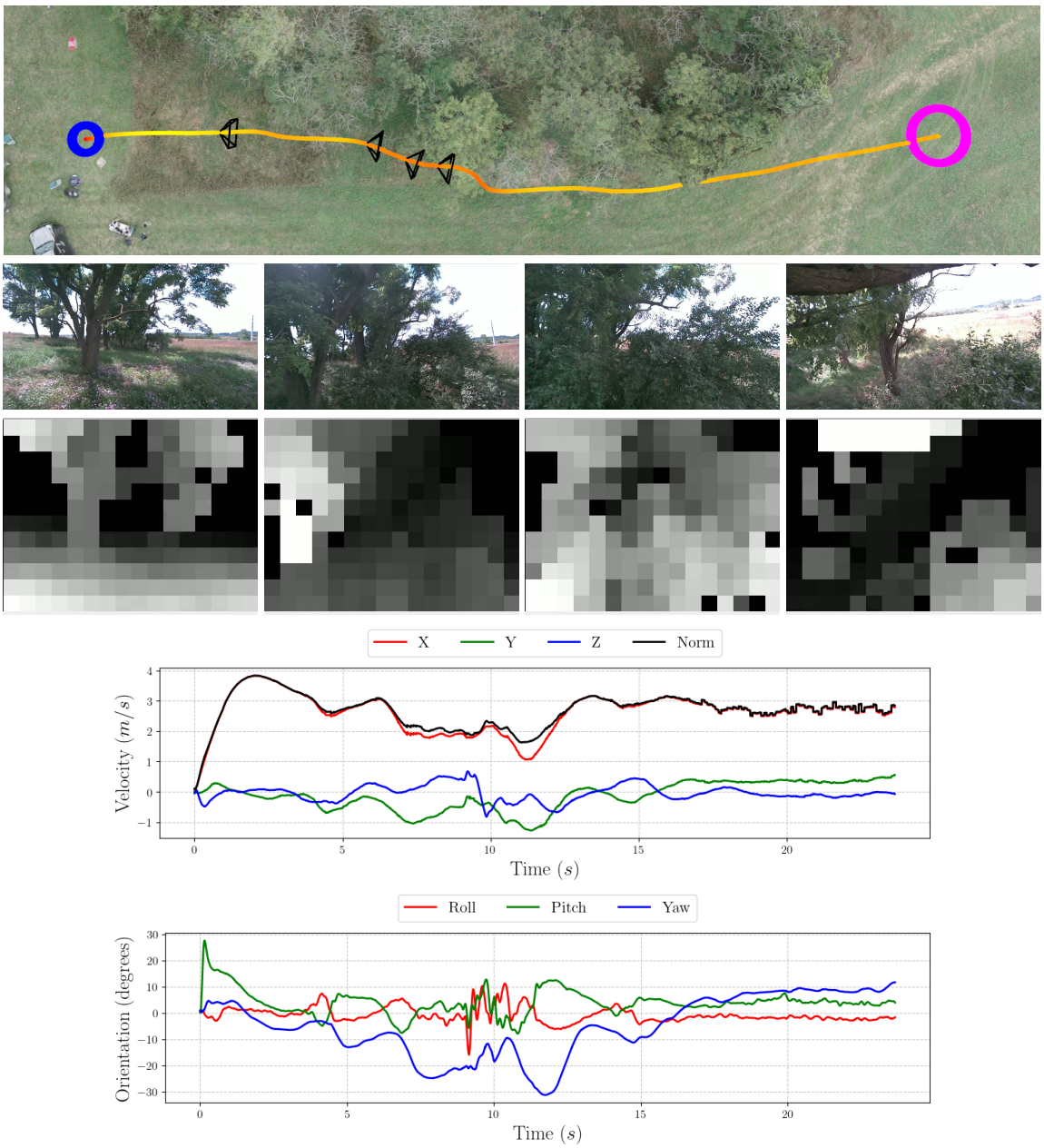


Figure 5.17: Outdoor obstacle avoidance test under tree canopy (forest flight 4 in Table 5.4). From top to bottom: VINS trajectory overlayed on terrain map, onboard RGB images, inverted and max pooled depth image, timeseries plot of linear velocity, timeseries plot of orientations.

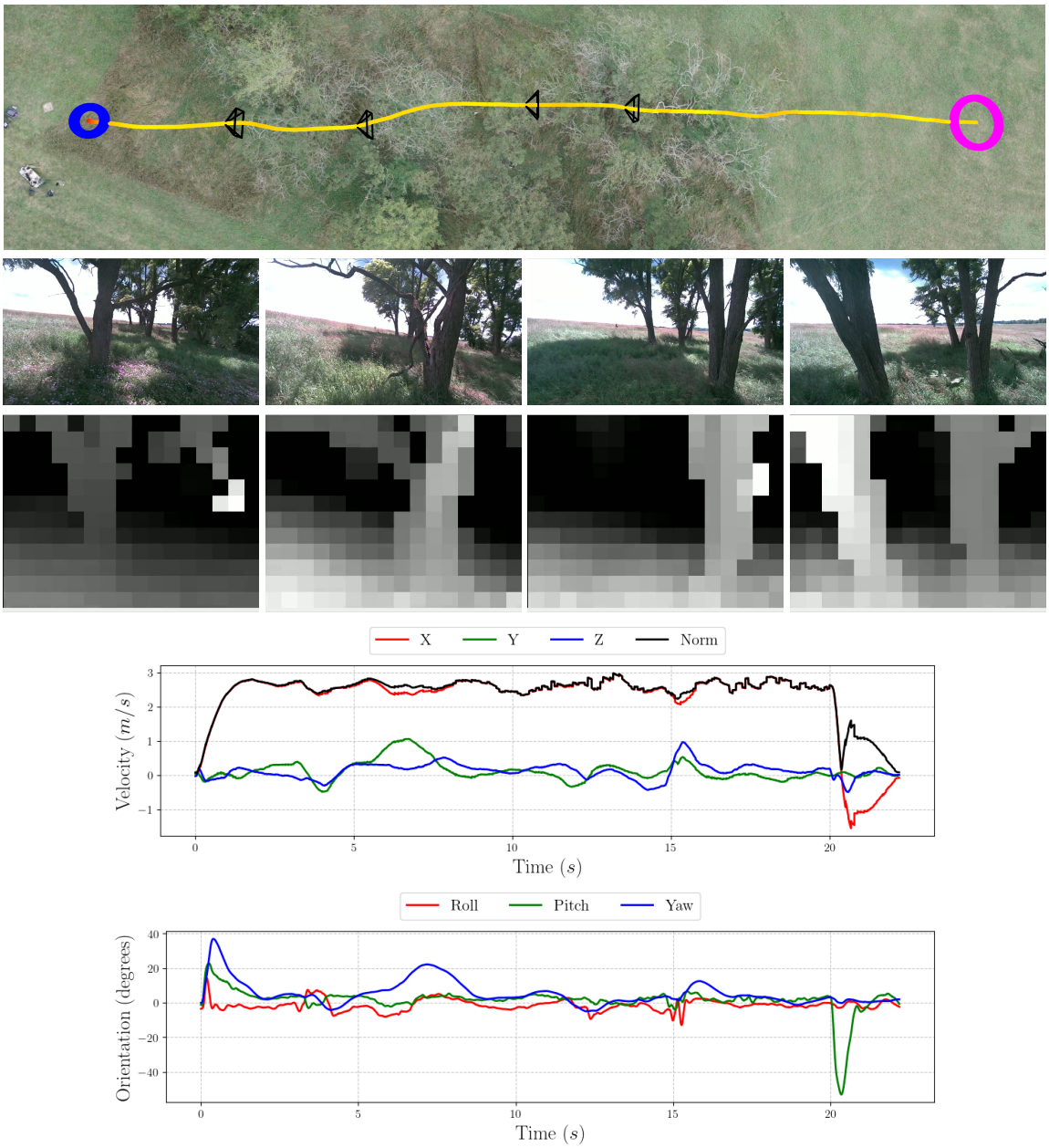


Figure 5.18: Outdoor obstacle avoidance test under tree canopy (forest flight 5 in Table 5.4). From top to bottom: VINS trajectory overlayed on terrain map, onboard RGB images, inverted and max pooled depth image, timeseries plot of linear velocity, timeseries plot of orientations.

Table 5.4: Chapter 5 Hardware Flight Trials

Env.	Flight #	Flight Time s	Path Length m	v_{max} m/s	Max Speed m/s	Avg Speed m/s
Outdoor Flight Arena	1	14.4	12.1	2.0	1.5	0.7
	2	13.2	17.9	2.0	2.0	1.2
	3	11.0	16.6	3.0	3.0	1.4
	4	17.3	26.9	3.0	2.8	1.4
	5	13.0	24.1	4.0	3.5	1.7
	6	12.0	24.5	5.0	3.7	1.9
	7	9.3	17.2	3.0	2.9	1.7
	8	11.7	22.3	3.0	2.9	1.8
	9	11.8	24.9	3.0	4.0	2.0
	10	11.1	22.5	3.0	2.8	1.8
	11	11.7	19.6	3.0	2.7	1.5
	12	11.3	20.4	3.0	2.7	1.6
	13	10.0	19.8	3.0	2.9	1.9
	14	9.5	17.9	3.0	2.4	1.9
	15	14.2	21.0	3.0	2.5	1.3
Forest	1	12.7	22.5	3.0	2.8	1.6
	2	31.1	52.9	3.0	2.8	2.2
	3	26.0	74.6	4.0	3.1	2.5
	4	23.7	70.1	5.0	3.8	2.7
	5	20.0	61.3	3.0	3.0	2.4

Chapter 6

Conclusion

This thesis addresses reactive motion planning and control strategies with limited field of view depth sensors across unknown environments.

6.1 Summary of Contributions

Chapter 3: Forward Arc Navigation. We present a reactive planner based on forward-arc motion primitives that uses a short history of RGB-D observations to navigate safely near obstacles. Instead of building a fused local map, the planner performs collision checks directly over recent sensor views. To ensure safety, the quadrotor always maintains a backup trajectory that allows it to stop and hover within known free space. Our approach is extensively evaluated in photorealistic simulation environments, achieving an 82% success rate, 24% higher than baseline reactive methods, with a near-zero collision rate of 0.2%. We also validate the method with a custom quadrotor deployed across three environments, completing 24 flight trials and covering 571 m without collisions, reaching speeds up to 6 m/s.

Chapter 4: End2End navigation. We build on recent advances in reinforcement learning with differentiable physics to develop a navigation policy that predicts thrust and heading directly from depth and state observations. By leveraging privileged information during training, our approach successfully navigates around large obstacles that cause baseline policies to fail. To address the sim-to-real gap, we introduce two

key strategies. First, we compute desired angular rates from predicted actions, which helps bridge the gap between simplified point-mass dynamics used in simulation and the full rigid-body dynamics of a real quadrotor. Second, we apply domain randomization over parameters, including gravity, to enable the policy to learn a robust feedback loop that compensates for modeling errors, such as a 15% mismatch in nominal thrust required to hover. Our approach achieves an 86% success rate, outperforming baseline strategies by 34%. The policy is also validated on hardware across 20 flights, covering 589 m without collisions.

6.2 Future Work

Future work could focus on reducing sensor dependencies to enable the development of smaller and more cost-effective platforms. For example, replacing the depth sensor with a monocular color camera would reduce payload requirements. Hu et al. [50] use optical flow from a single camera as input, although their learning-based optical flow model currently requires offboard processing due to computational demands. Alternatively, recent monocular depth prediction networks have shown promise for onboard augmentation of partial depth observations at real-time rates [51].

Another opportunity is to remove the visual odometry module used for velocity and position estimation. Our current setup requires a downward-facing camera and a laser rangefinder for visual odometry. Instead, the policy could predict velocity directly, as in Zhang et al. [9], eliminating the need for separate sensors. This shift introduces new challenges, such as how to specify goal-driven tasks without reliable position estimates. Consequently, future work could explore other tasks with more flexible definitions such as exploration or dynamic target tracking.

Further improvements are possible in both the training pipeline and the policy architecture. Our current approach remains near hover orientation and leverages point-mass dynamics to simplify optimization, but this limits the quadrotor's ability to execute more agile maneuvers. Incorporating full rigid-body dynamics during training, potentially through a hybrid model such as in Heeg et al. [44], which uses separate dynamics models for the forward and backward passes, could unlock more acrobatic capabilities. Finally, enhancing the policy's global awareness without relying on an explicit map remains an open challenge. Replacing the GRU with a more

expressive learned-memory architecture could improve spatial reasoning and long-term decision-making.

Appendix A

Forward-Arc Motion Primitives

A.0.1 Forward Arc Motion Primitives

Forward arc motion primitives use velocity control where the quadrotor is desired to have state $\mathbf{x}_{t+T}$ where T is the duration of the motion primitive.

$$\mathbf{x} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v_x \cos(\theta_{t+T}) \\ v_x \sin(\theta_{t+T}) \\ v_z \\ \omega \end{bmatrix} \tag{A.1}$$

where v_x is the forward x velocity and lateral velocity $v_y = 0$.

Then integration gives us an expression for position. For instance, integrating $\dot{x}$

is as follows

$$\begin{aligned}
\int_t^{t+T} \dot{x} dt &= \int_t^{t+T} v_x \cos(\theta_{t+T}) dt \\
&= v_x \frac{\omega}{\omega} \int_t^{t+T} \cos(\theta_{t+T}) dt \\
&= \frac{v_x}{\omega} \int_t^{t+T} \cos(\theta_{t+T}) \omega dt \\
&= \frac{v_x}{\omega} \int_t^{t+T} \cos(\theta_{t+T}) d\theta \\
&= \frac{v_x}{\omega} \sin\theta \Big|_{\theta_t}^{\theta_{t+T}} \\
&= x_t + \frac{v_x}{\omega}(\sin(\theta_{t+T}) - \sin(\theta_t)) \\
&= x_t + \frac{v_x}{\omega}(\sin(\omega T + \theta_t) - \sin(\theta_t))
\end{aligned}$$

$$\mathbf{x}_{t+T} = \mathbf{x}_t + \begin{bmatrix} \frac{v_x}{\omega}(\sin(\omega T + \theta_t) - \sin(\theta_t)) \\ \frac{v_x}{\omega}(-\cos(\omega T + \theta_t) + \cos(\theta_t)) \\ v_z T \\ \omega T \end{bmatrix} \tag{A.2}$$

When expressed relative to the quadrotor body frame this expression becomes

$$\mathbf{x}_{t+T} = \begin{bmatrix} \frac{v_x}{\omega}(\sin(\omega T)) \\ \frac{v_x}{\omega}(-\cos(\omega T)) \\ v_z T \\ \omega T \end{bmatrix} \tag{A.3}$$

Satisfying constraints

The trajectory is represented by a set of 7th order polynomials for velocity $v_k(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 + c_4 t^4 + c_5 t^5 + c_6 t^6 + c_7 t^7$ for each dimension of the the flat output space $[x, y, z, \psi]$.

The unknown polynomial coefficients $\mathbf{c}$ is obtained by solving a linear system for each axis independently with endpoint constraints on the derivatives. The initial constraints v_0, a_0, j_0, s_0 must match the endpoints of the prior reference trajectory to prevent discontinuities between trajectory segments.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 0 & 2 & 6 & 12 & 20 & 30 & 42 \\ 0 & 0 & 0 & 6 & 24 & 60 & 120 & 210 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 T^{-1} \\ c_2 T^{-2} \\ c_3 T^{-3} \\ c_4 T^{-4} \\ c_5 T^{-5} \\ c_6 T^{-6} \\ c_7 T^{-7} \end{bmatrix} = \begin{bmatrix} v_0 \\ a_0 T \\ j_0 T^2 \\ s_0 T^3 \\ v_T \\ a_T T \\ j_T T^2 \\ s_T T^3 \end{bmatrix} \tag{A.4}$$

Finally, the coefficients for position is obtained by integration.

$$p_k(t) = c_0 t + \frac{c_1}{2} t^2 + \frac{c_2}{3} t^3 + \frac{c_3}{4} t^4 + \frac{c_4}{5} t^5 + \frac{c_5}{6} t^6 + \frac{c_6}{7} t^7 + \frac{c_7}{8} t^8 \tag{A.5}$$

Bibliography

[1] Daniel Mellinger, Alex Kushleyev, and Vijay Kumar. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In *2012 IEEE International Conference on Robotics and Automation*, pages 477–483, 2012. doi: 10.1109/ICRA.2012.6225009. 2.1

[2] Helen Oleynikova, Zachary Taylor, Marius Fehr, Roland Siegwart, and Juan Nieto. Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1366–1373, 2017. doi: 10.1109/IROS.2017.8202315. 2.1, 3.1.1

[3] Charles Richter, Adam Bry, and Nicholas Roy. Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments. In *Robotics Research: The 16th International Symposium ISRR*, pages 649–666. Springer, 2016. 2.1, 3.1.1

[4] Vladyslav Usenko, Lukas Von Stumberg, Andrej Pangercic, and Daniel Cremers. Real-time trajectory replanning for MAVs using uniform B-splines and a 3D circular buffer. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 215–222, 2017. doi: 10.1109/IROS.2017.8202160. 2.1, 3.1.2

[5] Fei Gao, William Wu, Yi Lin, and Shaojie Shen. Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 344–351, 2018. doi: 10.1109/ICRA.2018.8462878. 2.1

[6] Jesus Tordesillas and Jonathan P. How. Minvo basis: Finding simplexes with minimum volume enclosing polynomial curves. *Computer-Aided Design*, 151: 103341, 2022. 2.1

[7] Zhepei Wang, Xin Zhou, Chao Xu, and Fei Gao. Geometrically constrained trajectory optimization for multicopters. *IEEE Transactions on Robotics*, 38(5): 3259–3278, 2022. 2.1

[8] Davide Falanga, Philipp Foehn, Peng Lu, and Davide Scaramuzza. Pampc:

Perception-aware model predictive control for quadrotors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. IEEE, 2018. 2.1, 2.2

[9] Yuang Zhang, Yu Hu, Yunlong Song, Danping Zou, and Weiyao Lin. Learning vision-based agile flight via differentiable physics. *Nature Machine Intelligence*, pages 1–13, 2025. 2.1, 3.2, 3.2.3, 5.1, 5.1.1, 5.2.1, 5.2.3, 5.2.5, 5.3.1, 5.12, 6.2

[10] Dominic Clifton and Betaflight Contributors. Betaflight: Open source flight controller firmware. https://github.com/betaflight/betaflight, 2024. URL https://github.com/betaflight/betaflight. Version 4.x, accessed July 2025. 2.2

[11] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 6235–6240. IEEE, 2015. 2.2, 4.3.3

[12] 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. ISBN 978-1-61284-386-5. doi: 10.1109/ICRA.2011.5980409. 2.2

[13] Taeyoung Lee. Geometric tracking control of the attitude dynamics of a rigid body on so (3). In *Proceedings of the 2011 American Control Conference*, pages 1200–1205. IEEE, 2011. 2.2, 5.2.5

[14] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34:189–206, 2013. doi: 10.1007/s10514-012-9321-0. 3.1.1

[15] Steven M. LaValle and James J. Kuffner Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001. 3.1.1

[16] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 2002. 3.1.1

[17] 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. 3.1.1

[18] Luxin Han, Fei Gao, Boyu Zhou, and Shaojie Shen. Fiesta: Fast incremental euclidean distance fields for online motion planning of aerial robots. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4423–4430. IEEE, 2019. 3.1.1

[19] Yue Pan, Yves Kompis, Luca Bartolomei, Ruben Mascaro, Cyrill Stachniss, and Margarita Chli. Voxfield: Non-projective signed distance fields for online planning and 3d reconstruction. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5331–5338, 2022. doi: 10.1109/IROS47612.2022.9981318. 3.1.1

[20] Xin Zhou, Zhepei Wang, Hongkai Ye, Chao Xu, and Fei Gao. Ego-planner: An esdf-free gradient-based local planner for quadrotors. *IEEE Robotics and Automation Letters*, 6(2):478–485, 2020. 3.1.1

[21] Robin Deits and Russ Tedrake. Computing large convex regions of obstacle-free space through semidefinite programming. In *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, pages 109–124. Springer, 2015. 3.1.1

[22] 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. 3.1.1

[23] Qianhao Wang, Zhepei Wang, Mingyang Wang, Jialin Ji, Zhichao Han, Tianyue Wu, Rui Jin, Yuman Gao, Chao Xu, and Fei Gao. Fast iterative region inflation for computing large 2-d/3-d convex regions of obstacle-free space. *IEEE Transactions on Robotics*, 2025. 3.1.1

[24] 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, 2016. doi: 10.1109/ICRA.2016.7487284. 3.1.1

[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. 3.1.1

[26] Jialin Ji, Zhepei Wang, Yingjian Wang, Chao Xu, and Fei Gao. Mapless-Planner: A Robust and Fast Planning Framework for Aggressive Autonomous Flight without Map Fusion. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6315–6321. IEEE, 2021. ISBN 978-1-72819-077-8. doi: 10.1109/ICRA48506.2021.9561460. 3.1.1, 3.1.2

[27] Jesus Tordesillas, Brett T Lopez, and Jonathan P How. Faster: Fast and safe trajectory planner for flights in unknown environments. In *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 1934–1940. IEEE, 2019. 3.1.1

[28] 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. 3.1.1, 3.1.2, 4.2.2

[29] Nathan Bucki, Junseok Lee, and Mark W Mueller. Rectangular pyramid partitioning using integrated depth sensors (rappids): A fast planner for multicopter navigation. *IEEE Robotics and Automation Letters*, 5(3):4626–4633, 2020. 3.1.2, 4.3, 4.6

[30] Vaibhav K. Viswanathan, Eric Dexheimer, Guanrui Li, Giuseppe Loianno, Michael Kaess, and Sebastian Scherer. Efficient Trajectory Library Filtering for Quadrotor Flight in Unknown Environments. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2510–2517, 2020. doi: 10.1109/IROS45743.2020.9341273. 3.1.2

[31] Pete Florence, John Carter, and Russ Tedrake. Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps. In *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*, pages 304–319. Springer, 2020. 3.1.2, 4.2, 4.2.2, 4.3, 4.6

[32] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 6(59):eabg5810, 2021. doi: 10.1126/scirobotics.abg5810. URL https://www.science.org/doi/abs/10.1126/scirobotics.abg5810. 3.2, 3.2.1, 5.1

[33] Junjie Lu, Xuewei Zhang, Hongming Shen, Liwen Xu, and Bailing Tian. You only plan once: A learning-based one-stage planner with guidance learning. *IEEE Robotics and Automation Letters*, 9(7):6083–6090, 2024. 3.2, 3.2.2, 5.1

[34] Minwoo Kim, Geunsik Bae, Jinwoo Lee, Woojae Shin, Changseung Kim, Myong-Yol Choi, Heejung Shin, and Hyondong Oh. Rapid: Robust and agile planner using inverse reinforcement learning for vision-based drone navigation. *arXiv preprint arXiv:2502.02054*, 2025. 3.2, 3.2.1

[35] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011. 3.2.1

[36] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021. 3.2.1

[37] Fei Gao, Yi Lin, and Shaojie Shen. Gradient-based online safe trajectory generation for quadrotor flight in complex environments. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 3681–3688. IEEE, 2017. 3.2.2

[38] Xuning Yang, Koushil Sreenath, and Nathan Michael. A framework for efficient teleoperation via online adaptation. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5948–5953. IEEE, 2017. 4.2.1

[39] 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 *Proceedings of the 2018 International Symposium on Experimental Robotics*, pages 524–535. Springer, 2020. 4.2.1

[40] Xuning Yang, Ayush Agrawal, Koushil Sreenath, and Nathan Michael. Online adaptive teleoperation via motion primitives for mobile robots. *Autonomous Robots*, 43:1357–1373, 2019. 4.2.1

[41] Wennie Tabib, Kshitij Goel, John Yao, Curtis Boirum, and Nathan Michael. Autonomous cave surveying with an aerial robot. *IEEE Transactions on Robotics*, 38(2):1016–1032, 2021. 4.2.1

[42] John W. Yao. *Resource-Constrained State Estimation with Multi-Modal Sensing*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, April 2020. 4.2.2, 4.3.3, 5.3.2

[43] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. In *Conference on Robot Learning*, pages 1147–1157. PMLR, 2021. 4.3

[44] Johannes Heeg, Yunlong Song, and Davide Scaramuzza. Learning quadrotor control from visual features using differentiable simulation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2025. URL `https://arxiv.org/abs/2410.15979`. To appear. 5.2.1, 6.2

[45] Isar Meijer, Michael Pantic, Helen Oleynikova, and Roland Siegwart. Pushing the limits of reactive navigation: Learning to escape local minima. *IEEE Robotics and Automation Letters*, 2025. 5.2.3

[46] James A Sethian. A fast marching level set method for monotonically advancing fronts. *proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996. 5.2.3

[47] Fanxing Li, Fangyu Sun, Tianbao Zhang, and Danping Zou. Visfly: An efficient and versatile simulator for training vision-based flight. *arXiv preprint arXiv:2407.14783*, 2024. 5.2.4

[48] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9339–9347, 2019. 5.2.4

[49] Alexander E Spitzer. *Dynamical Model Learning and Inversion for Aggressive Quadrotor Flight*. PhD thesis, Carnegie Mellon University Pittsburgh, PA, 2021. 5.2.5

[50] Yu Hu, Yuang Zhang, Yunlong Song, Yang Deng, Feng Yu, Linzuo Zhang, Weiyao Lin, Danping Zou, and Wenxian Yu. Seeing through pixel motion: learning obstacle avoidance from optical flow with one camera. *IEEE Robotics and Automation Letters*, 2025. 6.2

[51] Alessandro Saviolo, Niko Picello, Jeffrey Mao, Rishabh Verma, and Giuseppe Loianno. Reactive collision avoidance for safe agile navigation. *arXiv preprint arXiv:2409.11962*, 2024. 6.2