Learning Reactive Flight Control Policies: From LIDAR Measurements to Actions

Sam Zeng

CMU-RI-TR-18-58

August 2018

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Sebastian Scherer, Chair
Kris Kitani
Sankalp Arora

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

Keywords: Reactive Control, Imitation Learning, Reinforcement Learning, Field Robotics, Simulation, Environment Generation, Tunnels

Abstract

The end goal of a quadrotor reactive flight control pipeline is to provide control outputs, often in the form of roll and pitch commands, to safely navigate the robot through an environment based on sensor inputs. Classical state estimation and control algorithms break down this problem by first estimating the robots velocity and then computing a roll and pitch command based on that velocity. However, this approach is error-prone in geometrically degenerate environments which do not provide enough information to accurately estimate vehicle velocity. Recent work has shown that learned end-to-end policies can unify obstacle detection and planning systems for vision based systems. This work applies similar methods to learn an end-to-end control policy for a lidar equipped flying robot which replaces both state estimator and controller while leaving long term planning to traditional planning algorithms. Specifically, this work demonstrates the feasibility of training such a policy using imitation learning and RNNs to map directly from lidar range measurements to robot accelerations in realistic simulation environments without an explicit state estimate. The policy is fully trained on simulated data using procedurally generated environments, achieving an average of 1.7km mean distance between collisions. Additionally various real world flight tests through tunnel and tunnel-like environments demonstrate that a policy learned in simulation can successfully control a real quadcopter.

Acknowledgments

I would like to begin by first thanking my adviser, Dr. Sebastian Scherer for his encouragement and support in addition to providing me with many opportunities for developing and deploying robotic systems in real world environments. I would also like to thank Dr. Kris Kitani and Wen Sun for their feedback and suggestions during the early stages of this work. To Weikun Zhen and Sankalp Arora, I owe a great deal for their significant mentorship and advice during my time in the master's program. Additionally, Ai Yunfeng, John Keller, Yaoyu Hu, and Henry Zhang were all incredibly helpful with development of hardware and running field experiments, not only limited to this work but also on a wide variety of projects during my time in the master's program. Next, I would also like to thank my good friend Ratnesh Madaan for his feedback, collaboration, and support. Furthermore, I would like to thank the Department of Energy for the funding and resources that made this project possible. Finally, I am extremely grateful to my parents and to Laura Sullivan, for being there since the beginning to offer their love and support. This accomplishment would not have been possible without them. Thank you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Advancements in autonomous systems for small UAVs have resulted in research on a growing number of applications for autonomous quadcopters ranging from infrastructure inspection to package delivery.

Many of these applications require flying robots to navigate close to obstacles through cluttered, GPS-denied, environments. In such cases, safe control and navigation can be particularly challenging since a single crash can cause catastrophic failure. Furthermore, safe flight requires a robust velocity estimate since in the event of state estimation failure, there is no option to just stop and wait.

Due to the inherent instability of the system, constant control commands must be applied to prevent a collision. In the event the robot enters an environment where accurate state estimation is computationally impossible due to lack of features, we still want the system to do its best given available information to prevent immediate collision.

This work is motivated by the need to traverse tunnel and corridor-like environment, but the methods described here could be easily extended to another environment as long as enough training data can be supplied, likely though procedural environment generation in simulation.

1.1 Tunnel Environments

Tunnel environments, such as mines, are very challenging for robot navigation due to a combination of low light, constrained space, geometrically degenerate sections, and strong magnetic interference. Much research has been done on developing robots for autonomous operations in underground tunnel environments. Shaffer and Stentz [1992], Scheding et al. [1999], and Nuchter et al. [2004] all demonstrate ground robots operating in tunnels in underground mines with some form of state estimation requirement. More recently, with advancements in UAV technology, there has been a growing interest from researchers such as Kanellakis and Nikolakopoulos [2016] in operating autonomous UAVs in underground tunnels found in mine environments.

In particular, we will be focusing on developing systems for a flying robot equipped with LIDAR as the main navigational sensor to safely traverse the tunnel like environments of a mine. LIDAR technology has become significantly cheaper and lightweight enough to mount onto small quadcopters providing accurate range measurements which can be applied to a number of

Figure 1.1: Quadcopter flying in a coal mine.

inspection tasks.

Although LIDAR can directly observe obstacles in the environment, geometrically degenerate sections of environment can prove to be exceptionally challenging for LIDAR based state estimation algorithms since they rely on geometric structure to estimate velocity.

1.2 Geometrically Degenerate Environments

Degenerate environments in the context of state estimation refers to environments where the there is not enough observable information in the environment for a given sensor to solve for a full state estimation solution. Geometrically degenerate environments in particular are environments where the geometrical structure is not sufficient for a range based sensor to measure any information in at least one or more directions.

LIDAR based localization methods require corresponding environment structure which constrain the measurements in each dimension in order to accurately determine the robot's pose. We call an environment where the pose estimation problem is unsolvable due to lack of geometric structure geometrically degenerate. Zhen et al. [2017] introduced a method for measuring the degeneracy of the geometry in an environment called localizability of an environment.

Additionally, many environments with very small, sparse features or low localizability are essentially impossible to navigate safely through since a localization system will produce results

with extremely high uncertainty which when wrapped in a control loop trying to match a specific velocity will result in a very unstable system.

In practice, we often want to navigate areas where only some small subsections are unlocalizable and so it can be hard to predict the performance of a robot operating in an unmapped area. Tunnels and corridor-like environments often contain subsections of unlocalizable regions. This risk is amplified by flying robots since inherent instability can result in rapid and often catastrophic failure.

1.3 Reactive Control for Safe Flight

Due to the challenging nature of the desired environments, we focus on safe flight and avoidance of obstacles by looking to compute a control action as directly as possible given the current sensor information. This means we want to compute control actions based only on recent sensor readings and ignore any global long term history in order to achieve faster reaction time and lower computational cost. This removes the dependency on accurately mapping the environment and increases safety but at the cost of global optimally. Reactive flight can be prone to planning hysteresis and other forms of sub-optimal global behavior.

In practice on larger systems with global objectives, such as described in a work by Scherer et al. [2007], reactive flight control is often used as a low level layer operating at high frequencies so that a the high level global mapping and planning does not have to re-plan for small changes in local obstacles. On smaller systems with simpler missions, such as in a work by Ross et al. [2012] and Daftry et al. [2016] where a small UAV is navigating through a forest, reactive control may be the primary navigation solution.

The system we will focus on will fall into this second category of using reactive control as its primary form of navigation, although this does not preclude our method from being used underneath some high level global system. Figure 1.2 depicts the details of our reactive flight control problem.

We will sense obstacles in the environment with a 2D 270 degree FOV LIDAR sensor and use these measurements to compute roll and pitch control actions which correspond to an accelerating on a quadcopter. Additionally, we want to make progress in a goal direction which will be defined as the x-direction of the robot. Finally, we would also like the robot to traverse the environment at a fixed target velocity.

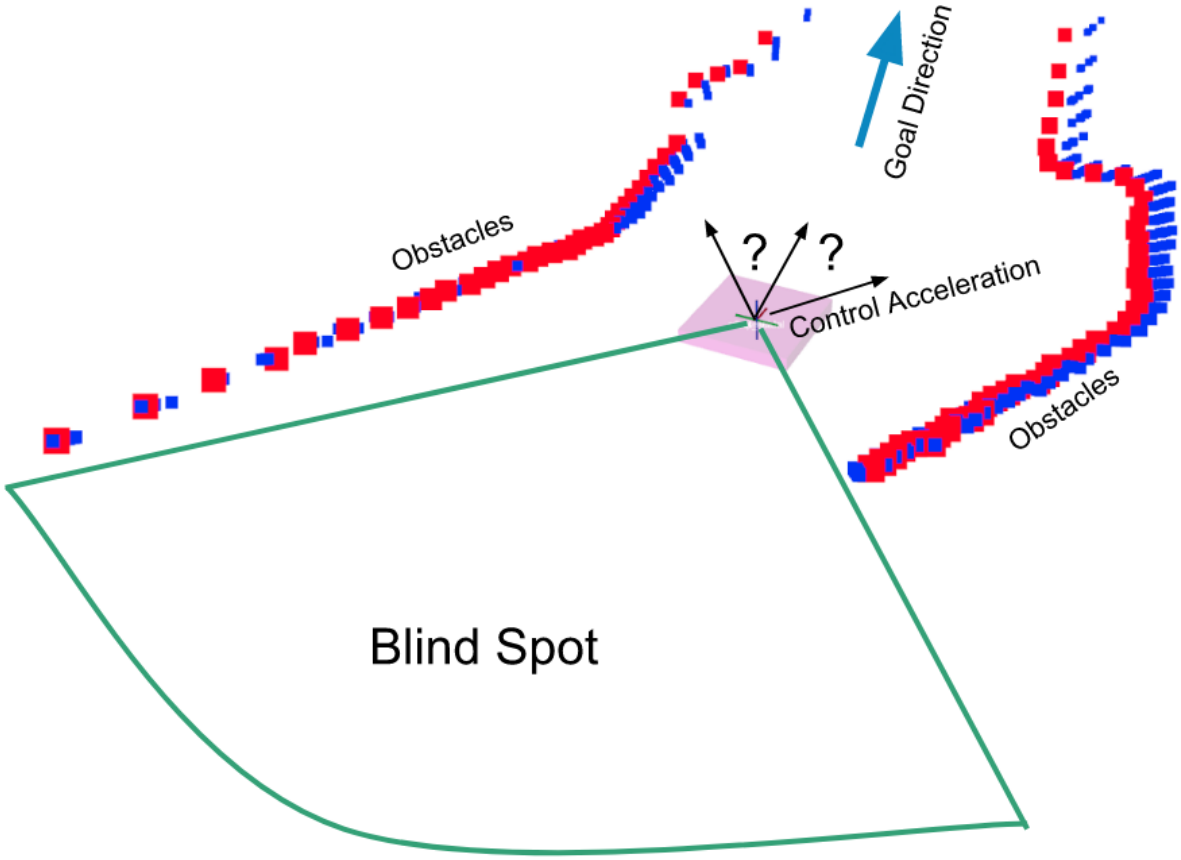

Figure 1.2: The Reactive Flight Control Problem: This diagram depicts the goal direction, sensor measurements and control actions available to the robot.

Chapter 2

Related Work

2.1 Reactive Navigation and Control

Reactive control is a common approach for mobile robot navigation in challenging environments. Initially in part inspired by biological systems, Arkin [1995], recently many researchers such as Daftry et al. [2016] and Gandhi et al. [2017] have sought to develop reactive control algorithms to directly compute control actions from local sensor data. In short, reactive control aims compute a viable control action based on local sensor information. Often these actions are not required or expected to be optimal, but they must be safe and viable.

Generally, many control pipelines for mobile robots follow the structure shown in Figure 2.1. Often, reactive control approaches have attempted to substitute or replace many aspects of this system pipeline depending on the constraints of the specific robot, sensors, constraints, and environment they are operating in. For example, Borenstein and Koren [1991] developed what they call a ”reflexive control” system for sonar equipped ground robots to simplify the planning and position control elements of the problem. They demonstrated this system on a sonar equipped ground robot which directly compute target velocities based on range measurements.

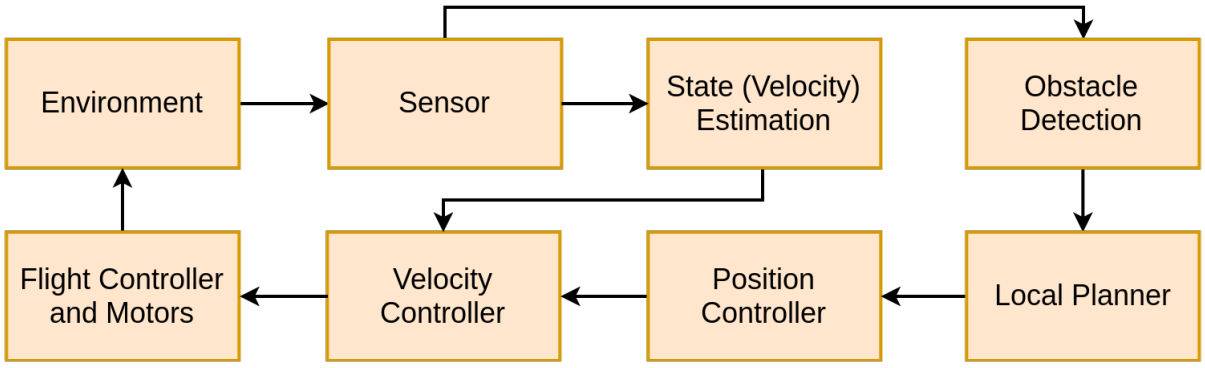

Figure 2.1: Classical System Architecture For Flight Control Pipeline

There exist many different tools for filling in each of these boxes with either a reactive system or a traditional method. Much of reactive control research is usually focused on developing

reactive systems for removing the need for a dedicated subsystem by devising a reactive control algorithm based on the challenges unique to the robot and target environment.

More recently, researchers have been exploring the use of machine learning to replace different parts of this generalized control pipeline with learned control policies in what is called end-to-end learning. We can divide up these systems based on the main navigational sensor used onboard each robot into two main categories, camera based systems and lidar based systems.

2.2 Camera Vs. Lidar Comparison

We make this division since the decision to use camera or LIDAR for navigation results on a system pipeline creates many different challenges for different algorithms based on each sensors strengths and weakness. Given this, it is worth considering the possibility of using both sensors on a vehicle which is definitely done on some systems, but it requires sacrificing additional payload and computational resources to support the added weight and processing required to be able to use both sensors which are often extremely limited on a quadcopter.

Table 2.2 emphasizes the differences between LIDAR and camera based reactive flight control pipelines by exploring the strengths and weakness of both sensors. As discussed earlier, we have chosen to use LIDAR as the main navigational sensing mode. We made this decision based on a number of factors.

First, since LIDAR is an active sensor, it does not require any external lighting to function since it is provided by part of the sensing process. Next, the large field of view offered by many LIDAR sensors is very useful for tunnel environments since there are almost always obstacles on all sides of the robot. A vision based system would require cameras in every direction and associated lighting and processing power to resolve obstacle locations. Finally, since LIDAR is able to precisely measure obstacle locations, it greatly simplifies the obstacle detection task. Additionally, LIDAR data can be recorded for offline post processing and reconstruction allowing for the construction of maps of the environment. Although SLAM may be difficult to run on-board the robot for navigation, post processing mapping can be done without real-time constraints producing significantly better results.

Lidar Vs Camera Tradeoffs		
	Camera	LIDAR
Pros:	-High Resolution -High Frequency -Inexpensive	-Directly Observes Obstacles -Large Horizontal FOV -Easier to Simulate
Cons:	-Generally Small Horizontal FOV -Cannot Directly Observe Obstacle Location -Difficult to Simulate -Requires external Lighting	-Low Resolution -Low Frequency -Expensive

Table 2.1: Tradeoffs between LIDAR and Camera systems for Reactive Flight Conrol on UAV in Tunnel Environments

However, the selection of LIDAR as the main navigational sensor for this system also presents

some unique challenges, especially with regards to velocity estimation if the system is required to fly in GPS denied environments. Since reactive control does not maintain a global state, there is often little need for position estimation which can be very computationally expensive. For these systems, state estimation mainly consists of estimating velocity, which is often very system dependant, and estimating orientation which is often done with an IMU. Velocity estimation on ground robots is almost always done in part by wheel encoders while flying robots equipped with cameras almost always use some visual odometry algorithm such as PTAM from Klein and Murray [2007].

On the other hand, LIDAR based systems do not have an easily transferable equivalent of visual odometry since they often offer much lower resolution and framerate. As a result, when performing feature tracking with LIDAR, the features cannot be as precisely matched between frames due to lower resolution and frame to frame differences are much larger due to higher framerate. Although systems for LIDAR SLAM exist such as hector_mapping Kohlbrecher et al. [2011] or for spinning 2D LIDAR, Zhen et al. [2017] demonstrates a combination of ESKF and GPF for full state estimation. However, they cannot overcome the fundamental sensor limitations and fail at high speeds and require significant geometric structure to be truly robust in practice.

2.3 End-to-End Learning for Camera Based Systems

A large amount of recent work focused on learning reactive control policies for vision based systems. Specifically, work from Ross et al. [2012]. demonstrates using imitation learning to train a reactive flight control policy for flying in forest environments by mapping directly from images to target velocities. The target velocities are then tracked by a downward facing camera. Additionally, Gandhi et al. [2017] uses almost a very similar framework for flying in indoor environment but instead this policy was trained using real world reinforcement learning. This work was particularly unique since the robot was actually allowed to crash many times during the training process to generate the required reward signal for reinforcement learning.

In general, many of these systems share a common framework shown in Figure 2.2 where they use a learned policy to build a tightly coupled subsystem that combines obstacle detection, local planning, and position control. Furthermore, these visual systems have demonstrated significant progress in being able to perform on real world robots.

The term End-to-End in the literature can be a bit misleading since, in reality, many of these systems keep many aspects of the system pipeline that work very well in practice as depicted in Figure 2.2. This makes a great deal of sense to focus the research on a constrained sub-problem of combining only the most failure prone subsystems in practice.

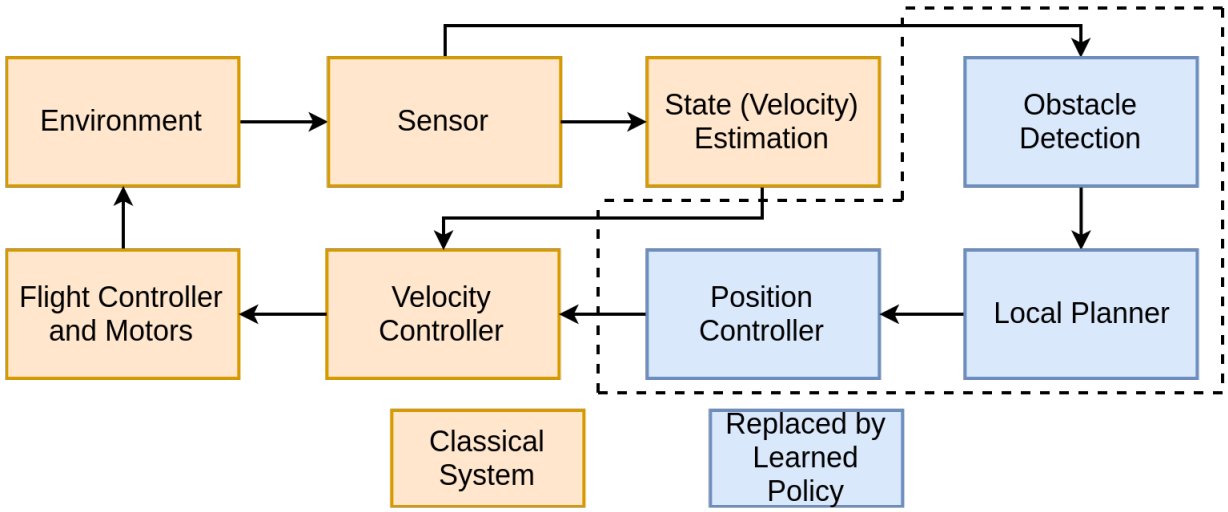

Figure 2.2: Camera Based System Architecture For Reactive Flight Control Pipeline

2.4 End-to-End Learning for LIDAR Based Systems

Less research has been done for reactive flight control with LIDAR based systems. One particularly relevant work done by Zhang et al. [2015] features learning a completely end-to-end control policy that maps LIDAR data directly to motor actions in simulation. The approach used in this paper is shown in Figure 2.3 in terms of the previously mentioned generalized flight control pipeline.

Their results show successful flight around single obstacles and simple hallways in simulation. Here, this policy is trained with imitation learning using an MPC expert to compute optimal control commands for a given dynamic system and is trained with a fully supervised learning process. However, this approach is limited to simulation due to difficulty generalizing to additional dynamics models as shown by their results.

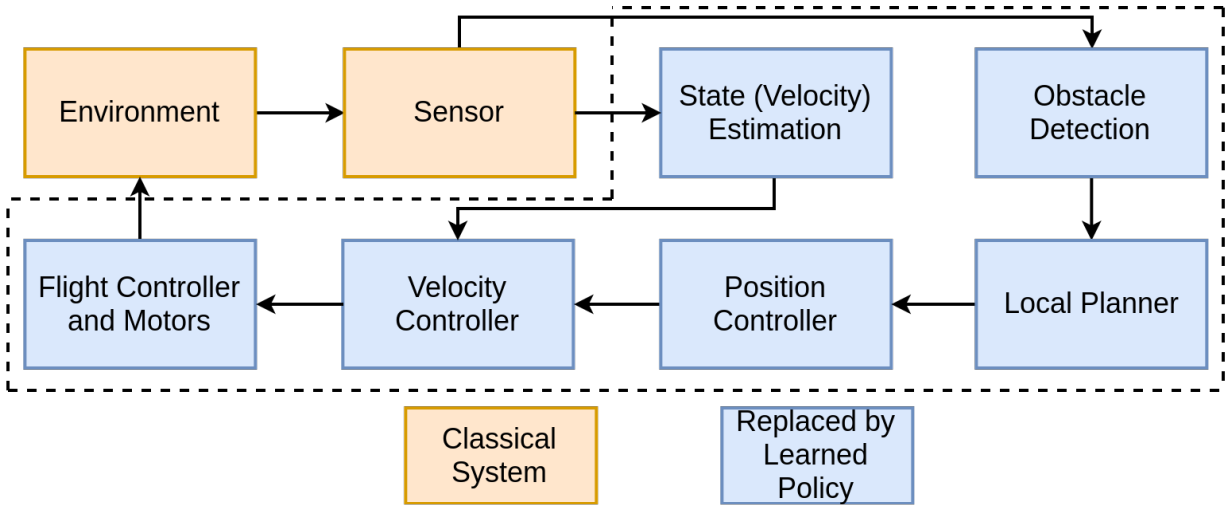

Figure 2.3: System Architecture For Fully End-to-End LIDAR Based Reactive Flight Control Pipeline

2.5 LIDAR to Actions

Based on these related works, we designed an approach to make the best use of many existing methods for some aspects of the system pipeline while replacing the most challenging areas for a LIDAR based system with a learned policy. This is because it may be difficult to compute exact solutions some of the sub-problems for any given subsystem, such as state estimation, but overall there still is enough environment information to determine the best control action based on the state.

To our knowledge, there has not been any successful work done with learning a reactive flight control policy for a small LIDAR equipped UAV capable of controlling a real robot. This work presents a LIDAR to Actions pipeline and a method for training such a flight control policy in simulation. Furthermore, we demonstrate that the LIDAR to Actions policy trained only in simulation can be directly transferred to a real robot.

Chapter 3

Approach

3.1 LIDAR to Actions System Design

The main challenge for a LIDAR based navigation on a quadcopter is accurate velocity estimation due to the limitations of small, lightweight, LIDAR sensors. However, the end goal of a reactive flight control systems is to produce control actions that keep the robot safe based on sensor inputs. There is no strict requirement to generate accurate velocity readings. Velocity estimation is a useful tool when it is working well, but classical approaches based on feature tracking can be very brittle since the state estimation and velocity control loop can become unstable, especially in geometrically degenerate environments.

Specifically, to address this challenge, we construct a subsystem that tightly couples velocity estimation and control to prevent this instability. Ideally, this subsystem would directly compute velocity control actions based on laser inputs. Since this is difficult and non-intuitive to design, we can apply state of the art deep learning methods to learn a control policy that directly maps from recent laser measurements directly to velocity control actions (accelerations). Figure 3.1 shows in blue how such a policy would fit into the rest of the flight control system architecture.

Additionally, since existing methods for flight control, local planning, and obstacle detection are quite robust given our system configuration, we can reduce the learning problem to only address the key difficulties with existing methods as shown in Figure 3.1. By combining only the velocity estimation, position controller, and velocity controller into a single learned policy, it minimizes the complexity the policy needs to learn while still providing maximum benefit by replacing the weakest link in the traditional system pipeline.

3.2 Model Architecture

Policy design can be split into two sections. First, the network must process the input of several recent laserscans into a vector of hidden units. Next, we will combine the output from the previous layers with the target way-point from the planner to produce the desired roll and pitch output. This separation process the time dependant input separately from the target point which should make it easier to learn.

In order to capture the motion of the robot, the input must be not only the current laserscan,

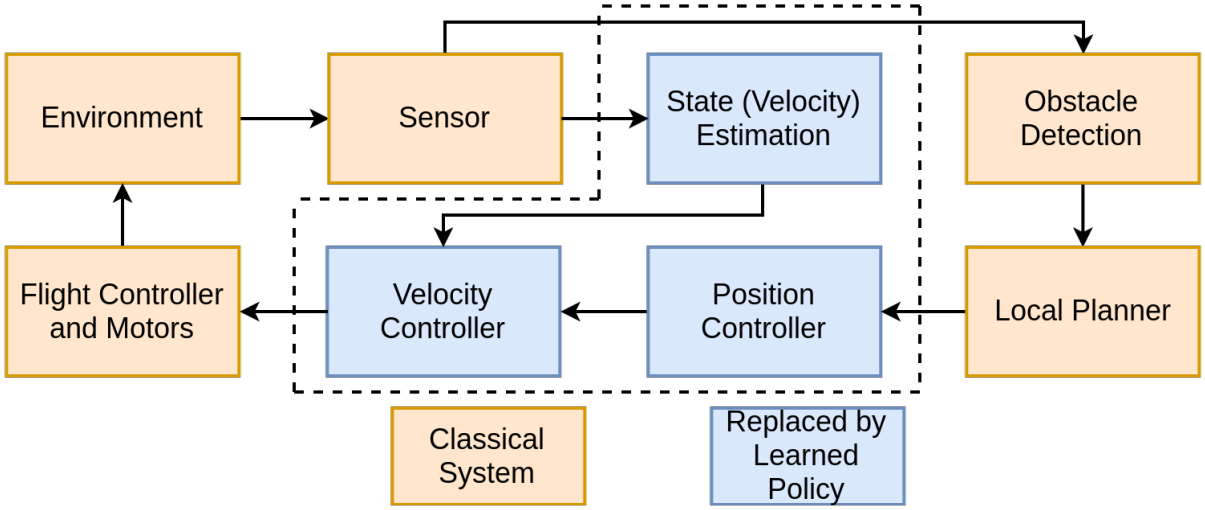

Figure 3.1: LIDAR to Actions System Architecture

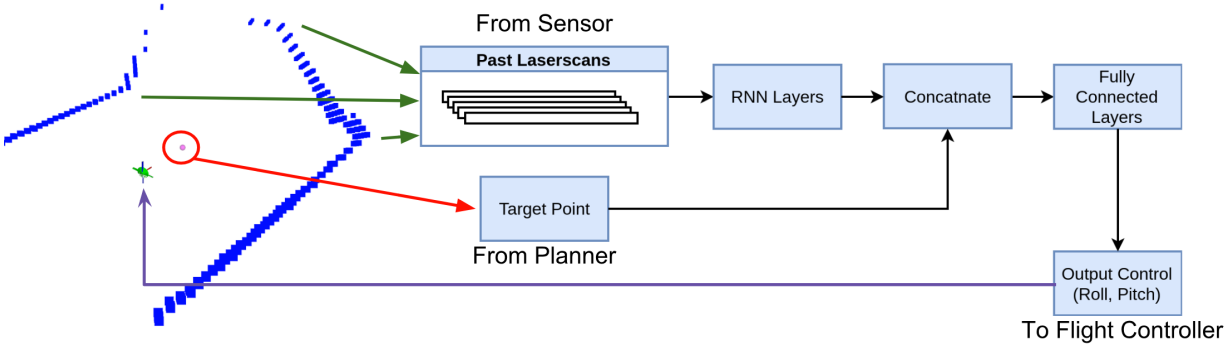

Figure 3.2: Model Architecture

but also some of the recent past laserscans over a some window size. State of the art deep learning approaches have provided significant advances in representation of time dependant data. In reinforcement learning, DQN Mnih et al. [2013] concatenate the past images of the environment and pass them into several CNN layers. However, Hausknecht and Stone [2015] found that using RNNs for the input layers improved performance over directly concatenating past inputs.

Based on the results in these papers, we choose to use RNNs to encode the laser input for this problem as shown in Figure 3.3. RNNs are particularly useful here since we can also adjust the window size without increasing the number of model parameters as would be the case when just concatenating past laserscans. Initial tests showed that a window size of 10 with a laser sampling frequency of 10hz worked well for this problem. This would encode information about the past 1 second of the robot's motion.

To further improve the robustness and transferability of the system, we also significantly down-sample the laserscan input to only 90 points for the 2D pipeline. The rationale behind this is to prevent more complex models from over-fitting to the simulated training environment.

Additionally, since the inputs are greatly down-sampled to 90 points, we found that CNNs were not necessary for such a small input and that directing inputting into fully connected RNN

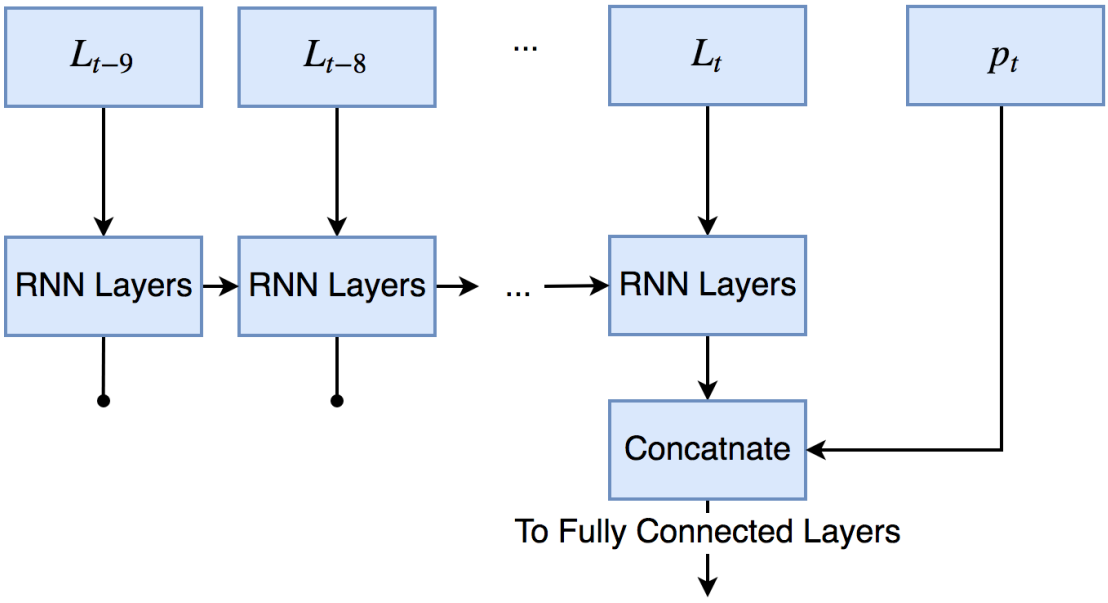

Figure 3.3: RNN Encoder Architecture

layers performed just as well.

Next, we combined the RNN output with the target waypoint and pass them into a fully connected layers producing roll and pitch output as shown in Figures 3.3 and 3.2. There was some initial concern that directly concatenating a single point with a large vector may result in problems during training due to the significant importance of the target point being only a very small part of the feature vector. However, in practice, this did not appear to be a problem and once training started it quickly produced output correlated with the target point.

3.3 Training

One of the main advantages of learning a control policy for a LIDAR based navigation sensor is the ability to accurately simulate LIDAR range measurements on a wide variety of different environments. Specifically, through simulation we were able to train policies on over 10,000 unique environments accumulated over 3 million expert labeled datapoints.

To do this, we constructed procedurally generated simulation training environments to train our policy which is able to provide a very large training dataset. Additionally, training in simulation allows us to query the true robot state for computation of expert control actions at any given timestep or forward simulate a single trajectory to compute the cost of executing a specific action.

3.3.1 Imitation Learning

Imitation learning has been used for learning control policies many times in the past. Ross et al. [2012] used imitation learning to train a reactive control policy for a camera based drone

flying through forest environments. More recently, Kaufmann et al. [2018] demonstrated using imitation learning to navigate through a drone racing environment at very high speeds for a camera based system.

Although it works very well in practice, imitation learning is limited by the fact that it cannot train a policy to have greater than expert performance and it often requires a human in the loop to act as an expert. Since we train the full policy in simulation, we have full access to the systems true state at any time allowing us to craft an expert capable of labeling training data for all situations. As a result we were able to generate arbitrary large amounts of training data since it was not necessary for a human expert in the training loop.

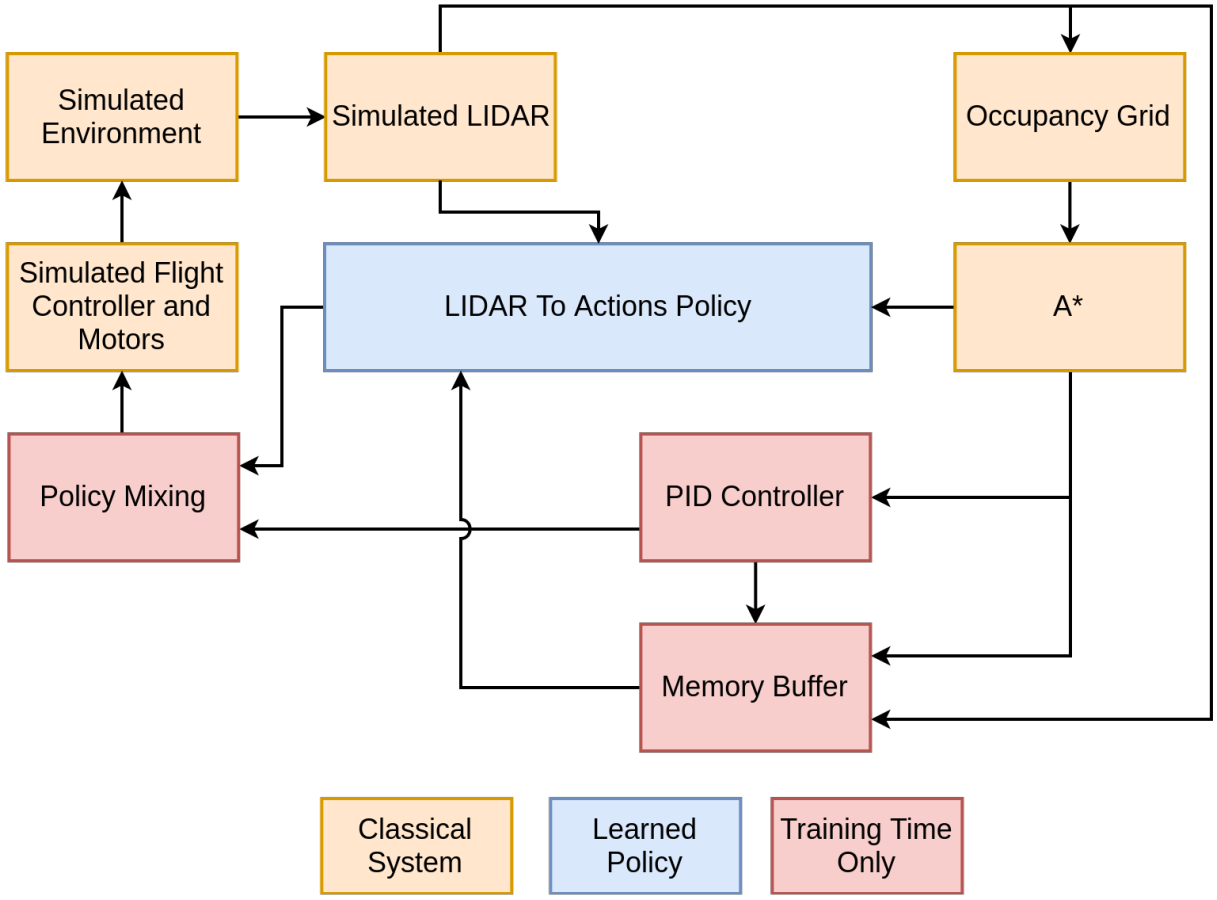

Figure 3.4: Imitation Learning Training Pipeline

In detail, policies were trained on over 11,000 episodes in less than 2 days accumulating over 2 million training iterations on 8 million training examples. The loss was computed using an L2 norm between the expert action and the policies action. This was then fed to the Adam optimizer which uses an adaptive learning rate to update the model parameters.

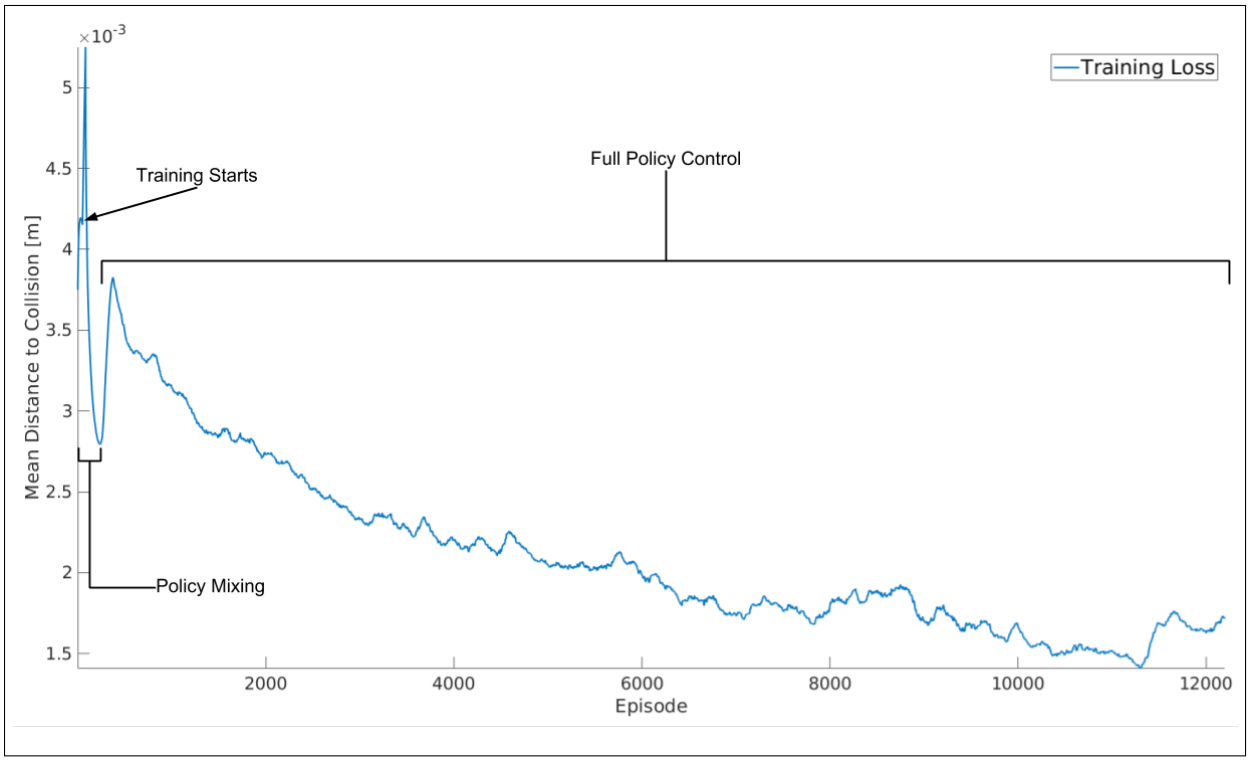

Figure 3.5: 2D Simulated Imitation Learning Training Loss

3.3.2 Near-Optimal Policy Expert

In order to compute the expert control action in simulation, we employed a standard cascaded PID controller for determining the the target action for a given set of laser inputs. We begin with a position controller where we compute the target velocity vector as shown in Equation 3.1 where p^s is the simulated true position of the robot. The superscript s denotes that this variable is the ground truth from the simulation environment.

$$v = v_{target}(p^s - p_t) \tag{3.1}$$

However, since in the planning section, we compute a target waypoint, p_t given the current pointcloud making it relative to the robots frame, this error will remain constant. As a result, the target velocity for a given laser state, v_t is just the projection of the desired velocity onto the waypoint from the planner.

Next, we use a PD controller to determine the necessary roll and pitch angles to meet a target velocity shown in equation 3.2.

$$pitch = p_x(v_x^s - v_x) + d_y a_x^s \tag{3.2}$$

$$roll = p_y(v_y^s - v_y) + d_y a_y^s$$

3.3.3 Policy Mixing

We apply policy mixing to start off the training as is done in by reinforcement learning algorithms in a similar fashion to Sun et al. [2017] in Deeply AggreVaTeD. The main difference from standard reinforcement learning algorithms here is that since an expert is available, we decay from expert policy to learned policy instead of from random policy to learned policy.

This way, during training time, the simulated robot will start by being fully controlled by the expert policy. This ensures that the initial training examples will be generated states seen during "nominal" flight conditions. Ideally, after training on these states, the policy should be able to replicate them in practice. However, when the trained policy is first used to control the robot, it often leads to a different unseen state distribution. Policy mixing slowly phases in these new unseen states which leads to more stable training.

In detail, we used a linear decay rate starting with full expert control and start shifting towards full policy control over a duration of 40,000 simulation steps. Additionally, there is a burn in period of 10,000 steps to partially fill the memory buffer before training begins.

3.3.4 Memory Buffer

To improve the training performance, we implemented a DQN style Mnih et al. [2013] memory buffer to record training data during an episode instead of immediately computing a loss. Here the memory records the laserscan, target waypoint, and expert action at every timestep. Figure 3.4 depicts how this fits into the system pipeline at training time. Due to memory size limitations, we limit the memory to 500,000 training examples after which the oldest samples are overwritten.

This memory buffer has several major benefits over directly training when the data is observed. First, it allows for IID sampling from the recorded dataset during each training iteration. Additionally, when doing imitation learning, this is essentially very similar to using DAGGER, from Ross et al. [2011], and aggregating at every timestep. This ensures that the state distribution in the training data will be similar to the state distribution at test time since the training data is generated from the robot using the policy for its control actions.

3.3.5 Reinforcement Learning

In addition to imitation learning, state of the art reinforcement learning approaches have the promise of learning greater than expert results. One approach in particular from Sun et al. [2017] called Deeply AggreVaTeD describes a training procedure for using an expert to achieve reinforcement learning level performance with exponentially reduced the required training data training given a stochastic policy and a reward function. This approach computes the cost-to-go for any given robot state and control action in combination with the expert policy. Basically, this computes how much reward the expert could still get given it first took the policies action. This cost is used to then updates the policy to minimize the cost-to-go of the policy's action.

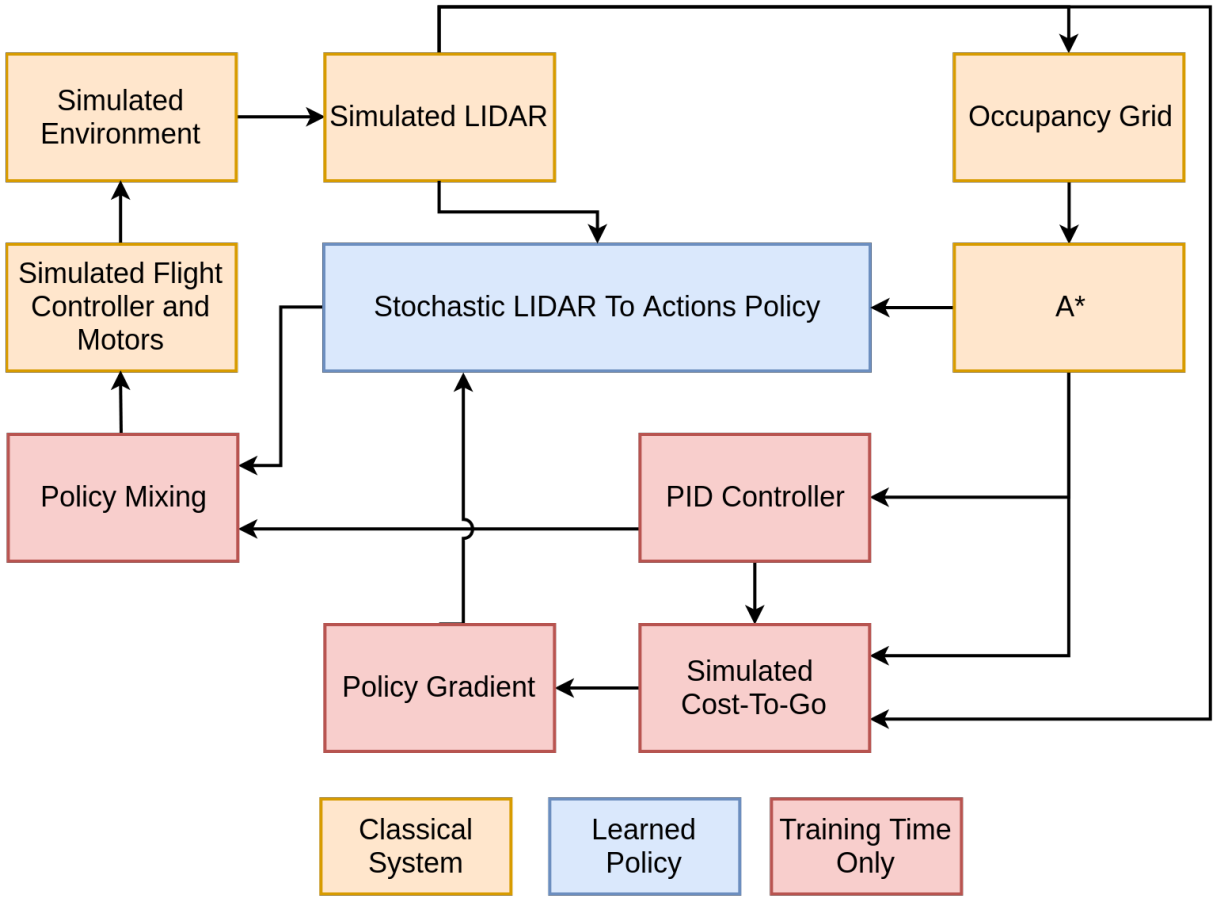

Figure 3.6: Deeply AggreVaTeD Training Procedure Applied to Our System

3.3.6 Metrics and Reward

We measure the progress of training and validate the policies with several different metrics to estimate if the policy would behave desirably during flight. Additionally, for computing cost-to-go, we need a reward function which is a combination of factors based off of several metrics.

The first metric we can look at is the $L2$ loss which measures the difference between the policy's action and expert action for any given state. This can be either computed on unseen data during the current episode which we call the episode loss or it can be computed at training time on the data sampled from memory which is the training loss.

Next, we can measure the mean distance to collision when accumulated over episodes. For a sequence of n episodes, this is computed as shown in equation 3.3

$$d_{mc} = \frac{\sum_{i=0}^{n} d_i}{\sum_{i=0}^{n} c_i} \tag{3.3}$$

Here, d_i is the distance traveled in the x direction in episode i and c_i has a boolean value of 1 if the episode terminated with a collision or 0 if the full 60 seconds pass with no collision. The value of d_{mc} ranges from $-\infty$ to ∞, but in practice, this a positive number for all of the approaches evaluated and approaches ∞ for the expert at low target velocities.

The final metric is the mean x velocity over a number of episodes to estimate the rate of progress the robot makes in the goal direction at any given episode. We expect this velocity to be lower than the target velocity since the robot cannot just travel in a straight line.

Finally, we compute the cost-to-go by forward simulating the robot from a given state as a sum of the future action cost of the expert, the distance from obstacles, final distance from target waypoint, and weather or not the robot is expected to crash as shown in Equation 3.4

$$C(s,a) = w_{act}||a|| + \sum_{n=1}^{H} (w_{act}||a_{e,i}|| + w_{obs}d_{obs,i}) + w_{err}||x_{final} - p_t|| + w_{crash}c \tag{3.4}$$

Here, we weight the value of each aspect of the cost according to the parameters w_{act}, w_{err}, w_{obs} and w_{crash}. The forward simulation is done over a finite horizon of H steps.

3.4 Local Planner

Traditional motion planning algorithms are capable of finding collision free paths given depth information of obstacles in real time. In order to simplify the navigation problem, we apply tradition planning techniques to find a safe path given the local obstacle information from just the latest laser scan. The goal of this planner is to provide information on what direction the policy should currently be moving in to successfully navigate the environment within the robots field of view. This information will be passed to the policy as a single waypoint, p_t, which we will refer to as the target point or target waypoint.

In this work, we use two different planners for 2D and 3D environments shown in Figure 3.7.

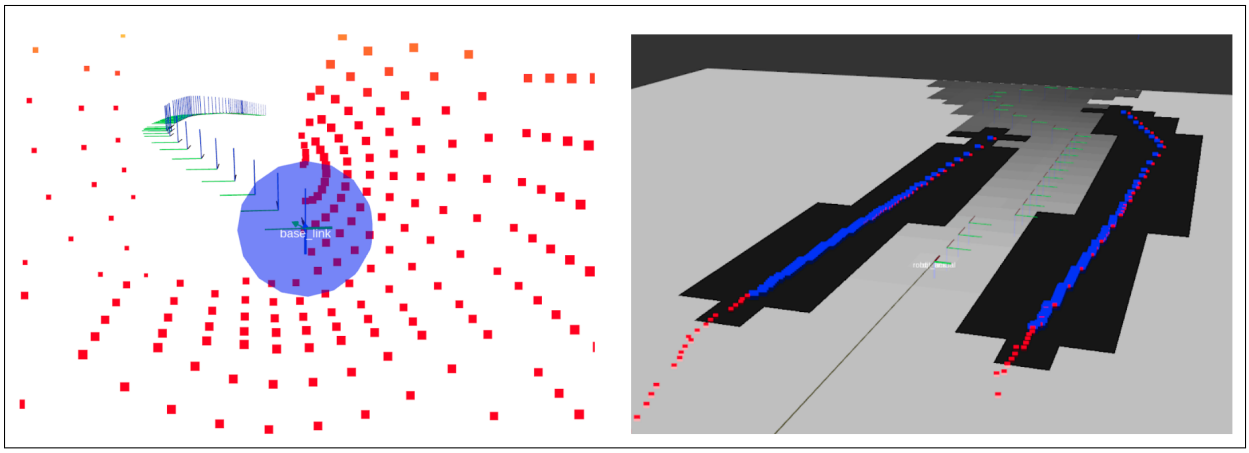

Figure 3.7: Left: Informed RRT* Planner in 3D Environment. Right: A* Planner in 2D Environment

For 2D environments, we use A* since the planning environment can be simplified with a very low resolution 8 connected occupancy map resulting in quick planning times and deterministic performance. Furthermore, we compute the cost of traversing a cell as follows:

$$c_{cell} = d_{previous} + min(\frac{\alpha}{0.25 + d_{obs}}, 2)$$

Here $d_{previous}$ is the distance from the center of the previous cell in the path. d_{obs} is the distance from the nearest obstacle and the final cost is inversely proportional to the distance from the nearest obstacle. After we are a save distance away from obstacles $(> 2m)$ this part of the cost is fixed since we do not care how far away the obstacles are after that.

After we have planned a path, we can divide up the path into a series of waypoints and use the first waypoint to provide information to the policy on what direction to move locally. We will treat this first waypoint as the target for our controller and it will be denoted as p_t. This reduces the complexity and allows the policy to learn a much simpler local control policy without sacrificing global performance.

3.5 Flight Controller

Many off the self quadcopters come with a high frequency built in flight control system to convert attitude control commands into motor velocities using IMUs, barometers, and magnetometers for feedback. These systems are well-tuned and are very capable in many common flight regimes. This ensures that attitude commands produced by the model are accurately tracked by the robot in flight. Additionally, we model the tracking error in simulation as discussed in detail in the robot dynamics simulation section.

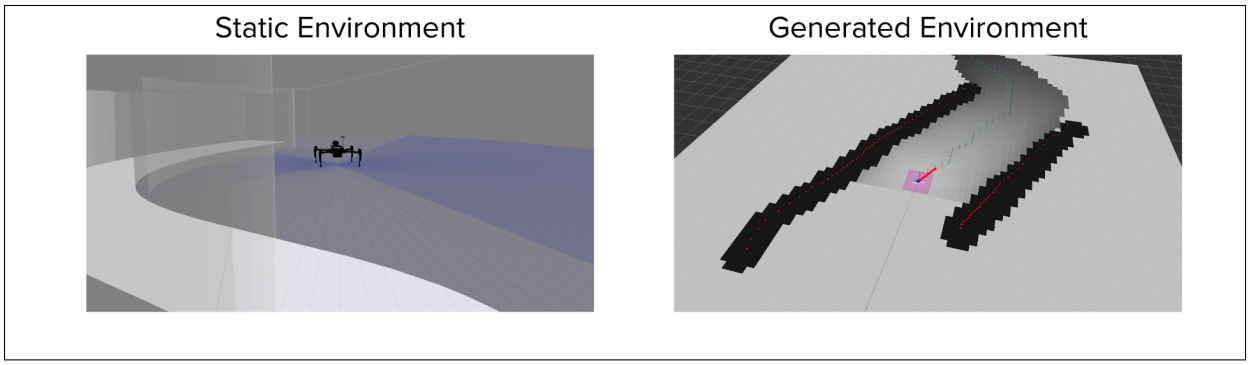

Figure 3.8: Example 2D Environments used for training the policy

3.6 Simulation

Simulation can be broken down into three parts. First, we need an environment for the robot to traverse. Next, we need to simulate what sensor measurements the robot would receive based on its state inside the simulated environment. Finally, the robot produces a control action to affect the robots state in the simulated environment based on the simulated robot dynamics.

3.6.1 Environment Simulation

A number of different simulation environments were used to train different policies. These can be broken down in two ways, static vs procedural and 2D vs 3D environments. Policies were successfully trained on each combination of environment type.

One important thing to note is for the quadcopter velocity control problem in particular, moving from 2D to 3D environments is not significantly more challenging. This is due to the fact that onboard sensors such as a barometer can accurately compute vertical velocity estimates for most flight conditions. This is independent of environment geometry and will work fine even in geometrically degenerate environments. As a result, in 3D, we can directly compute the target Z velocity and acceleration with our sensors and thus we do not need to learn a control policy for the Z direction.

Static Environments

Static environments remain the same during each episode of training, but with randomized start positions within a starting area. The main purpose of these environments is to confirm that a given model is capable of learning to model the time depend input data and produce reasonable outputs. After training on these environments, a good model will produce very near expert performance. However, we do not expect it to exceed expert performance with imitation learning alone.

Procedural Environment Generation

Procedural environment generation allowed training on a much wider variety of environments that change between each episode during training so that the policy is trained on a much wider

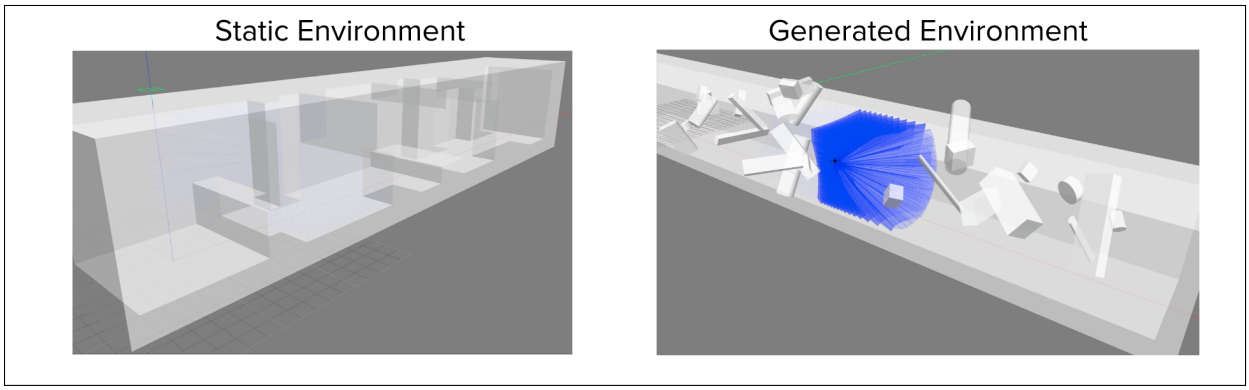

Figure 3.9: Example 3D Environments used for training the policy in 3D. On the left is the static training environment. On the right shows a single configuration of the generated training environment. After every episode, the objects are randomly rearranged.

variety of data which drastically improves generalizability of the final policy, but requires significantly more training time. Additionally, we do not expect a policy to achieve expert level performance on these environments.

For the 2D generated environment, we generate tunnel environments section by section by breaking down the environments into global and local environment structures. Global structure consists of 2 parameters governing the direction of the tunnel and the tunnels average width at the current section. On generation of each section, we randomly select a new angle and new tunnel width and produce a new section based on those parameters. The local structure determines what kind of texture is placed on the walls for a given section. In our environments, there are three types of local structure, smooth, small bump, and intersecting hallway. On generation of the new section, we pick a random local structure type for each wall with randomized parameters governing the specific structure.

Additionally, the environment generation has no requirement to generate an environment with a feasible path. The path may be too narrow or even completely block of and it is the planners responsibility to determine where to place the waypoints. The desired behavior in this case is for the robot to move to the point furthest away from obstacles.

3.6.2 Sensor Simulation

The next step in simulation is to compute the simulated measurement based on the robots pose inside the simulation environment. This is done by first computing an exact measurement and then adding gaussian noise to each laser pixel shown in equation 3.5.

$$r_{sim,\theta} = r_{raycast,\theta} + N(0, \sigma_{laser}) \tag{3.5}$$

Here, $r_{raycast,\theta}$ is the true distance between the simulated sensor pose and the simulated environment in direction θ and $r_{sim,\theta}$ is the value used in the simulated laserscan for the direction θ. This approximation of 2D LIDAR simulation does not take into account the roll and pitch angles and any change in measurement these may cause. Instead, during test time on the real

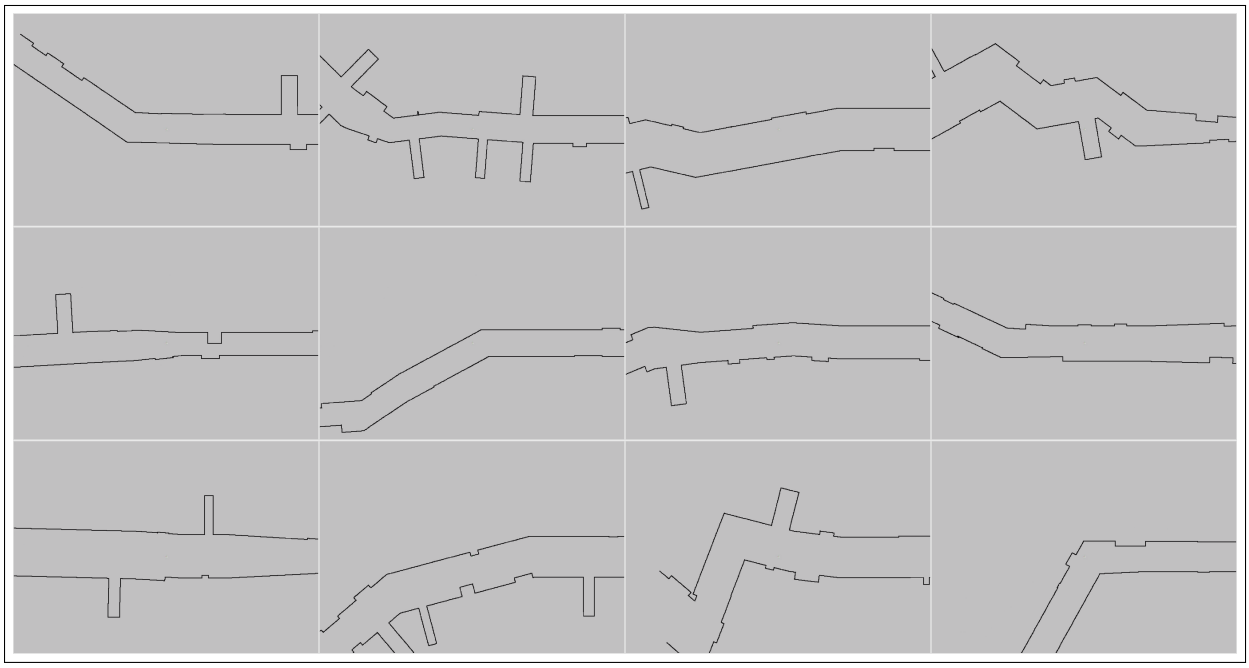

Figure 3.10: Sample sections of procedurally generated 2D environments

Figure 3.11: Sensor Model

robot, we assume the walls are vertically symmetrical within the ± 5 degrees of horizontal and use roll and pitch from the IMU to compensate the measurements for the attitude change.

3.6.3 Robot Dynamics Simulation

Finally, after passing the simulated laserscan into the LIDAR to Actions pipeline, we output a control action in the form of a target roll and pitch angles. On the actual vehicle, these command go to a flight controller where they are used to compute motor velocities. However, in order to simplify the system dynamics, we treat the flight controller as a back box and model its dynamics separately. This leaves us with a two piece system, first consisting of the flight control tracking dynamics which governs the quadcopter's orientation and the next the transnational dynamics which governs how the robot pose in space.

We simulate the flight controller dynamics with a second order model of the error dynamics as described by Sa et. al. Sa et al. [2017]. In their work, they treat the flight control system for the DJI Matrice 100 as a back box with second order dynamics and preform system identification to determine its parameters. For our simulation, we use the second order controller error model described in this paper with the identified parameters as baseline values for m, b and k in equation 3.6.

$$m\ddot{e} + b\dot{e} + ke = 0 \tag{3.6}$$

Here, e is the tracking error between the target attitude and the true attitude. By numerically

integrating during simulation, we can reconstruct the true attitude at every time step as follows:

$$\ddot{e}_{t+1} = -\frac{b}{m}\dot{e}_t - \frac{k}{m}e_t \tag{3.7}$$

$$\dot{e}_{t+1} = \dot{e}_t + \ddot{e}_t \Delta t \tag{3.8}$$

$$e_{t+1} = e_t + \dot{e}_t \Delta t + \ddot{e}_t \Delta t^2 \tag{3.9}$$

$$x_{t+1} = T_{t+1} + e_{t+1} \tag{3.10}$$

However, since these parameters may not correspond directly to the flight control on our robot due to a different payload and additional minor differences, we would like to learn a policy that is able to generalize to many similar, but different quadcopter dynamic systems. To do this, at the start of every episode, we generate a random offset within 10% of the parameters value and add it onto the parameter for that entire episode as shown in equation 3.11.

$$(m + m_{offset})\ddot{e} + (b + b_{offset})\dot{e} + (k + k_{offset})e = 0 \tag{3.11}$$

$$m_{offset} \sim Uniform(-0.1m, -0.1m)$$

$$b_{offset} \sim Uniform(-0.1b, -0.1b)$$

$$k_{offset} \sim Uniform(-0.1k, -0.1k)$$

Now that we have the attitude dynamics, we can also compute the x-y linear dynamics. This is done by first assuming the robot's thrust controller maintains a constant altitude regardless of roll and pitch angles. This is a reasonable assumption to make since a combination of feed-forward and a inner high frequency feedback loop maintain altitude for small angles of roll and pitch quite well in practice.

Given this assumption, we can view the forces acting on the robot shown in Figure 3.12. This Figure shows that the robot will accelerate at a rate of given a roll or pitch angle of $\theta_t = x_t = Tt + e_t$. From this we can integrate the accelerations to simulate the velocity and position of the robot.

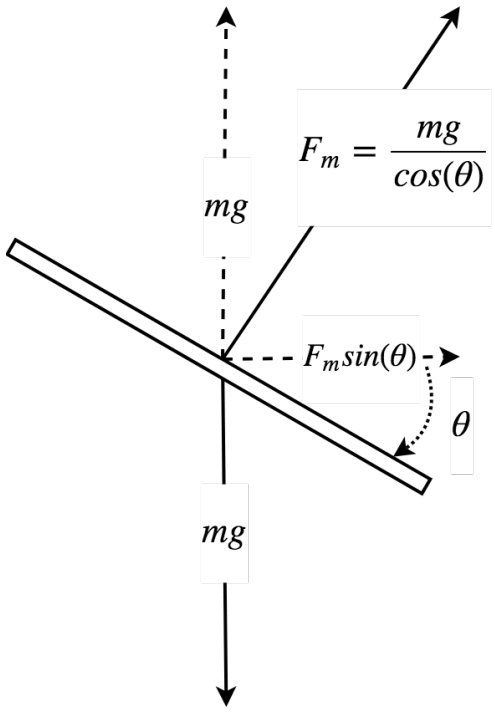

Figure 3.12: Dynamics Model

Chapter 4

Experiments and Results

The experiments and results shown here are split into two parts. First we will demonstrate successful policy performance in simulation on the generated environments. Furthermore, we will compare the 2D implementation of this method to off the shelf SLAM algorithms in 2D simulation environments. Additionally, we demonstrate a simple extension of this approach working in 3D. Next, we test policies trained only in simulation controlling a real robot in a variety of real world environments. Initial results show policies learned in simulation can be directly transferred to the robot.

4.1 Imitation on Simulated 2D Environments

Figure 4.1 depicts successful training with imitation learning as the mean distance between collisions is rises throughout the training process. Since near the end of training, the policy would average collisions in approximately 1% of episodes, the data in this plot is averaged with a large window of 3000 episodes to provide a better understanding of the true performance.

In addition to measuring the mean distance to collision, we also can examine the training loss and difference between the expert and policy during the episode, or mean episode loss as shown in Figure 4.2. Since this mean episode loss is computed on data that the policy has not yet been trained on, the episode loss is similar to the validation loss and we should expect it to always be above the training loss.

Overall, imitation learning on these environments was largely successful in learning a generalizable policy that works well on most of the training environments.

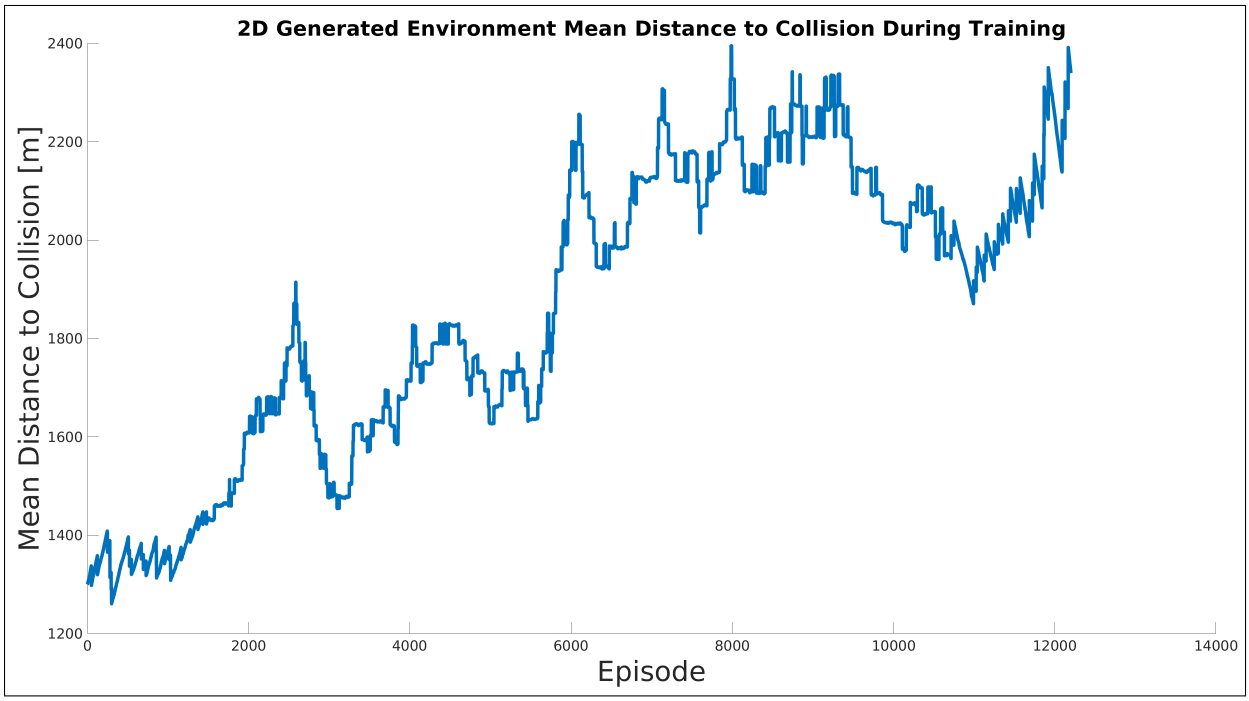

Figure 4.1: 2D Simulated Imitation Learning Mean distance to Collision on Generated Environments

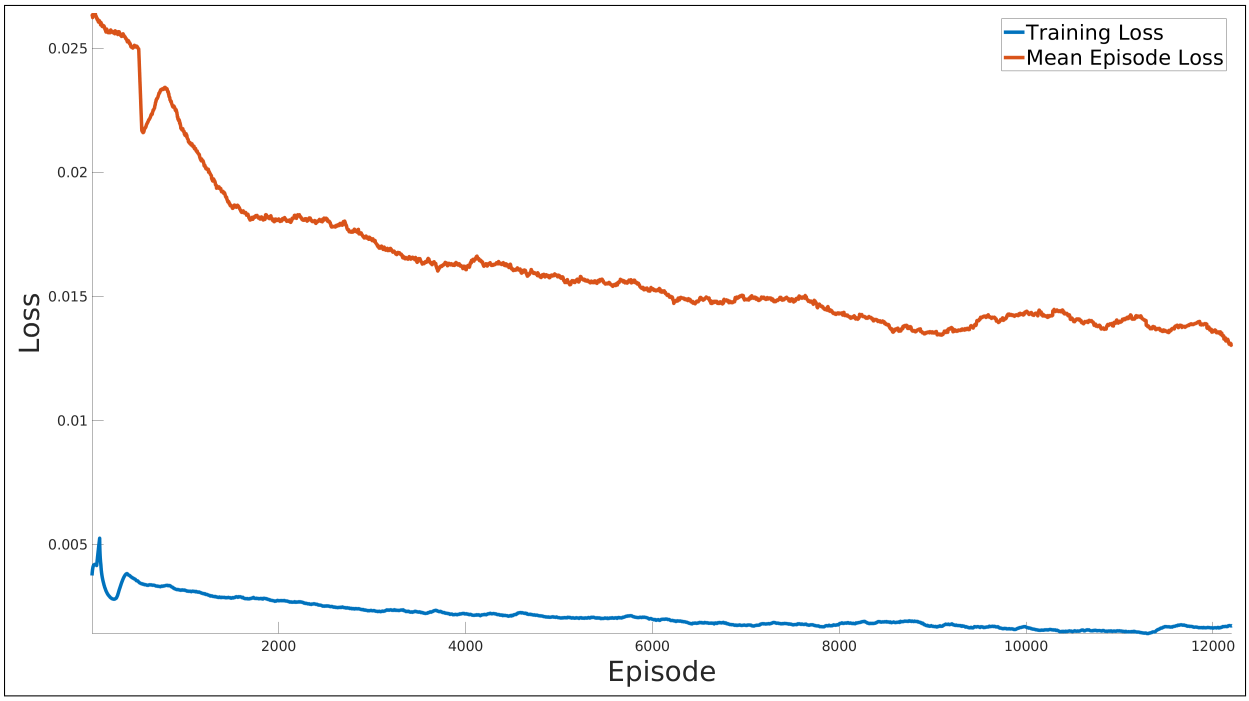

Figure 4.2: Training and mean episode loss during training time. Both losses decrease on average with additional episodes, but the mean episode loss is consistently higher than the training loss.

4.2 Reinforcement Learning on Simulated 2D Environments

We also compare the results of training a policy with Deeply AggreVaTeD to see if we can achieve better performance with reinforcement learning. Specifically, we took a set of weights pre-trained for 2m/s which started off averaging around 300m between collisions and attempted to improve the performance by additionally training with Deeply AggreVaTeD. The results are shown in Figures 4.3 and 4.4.

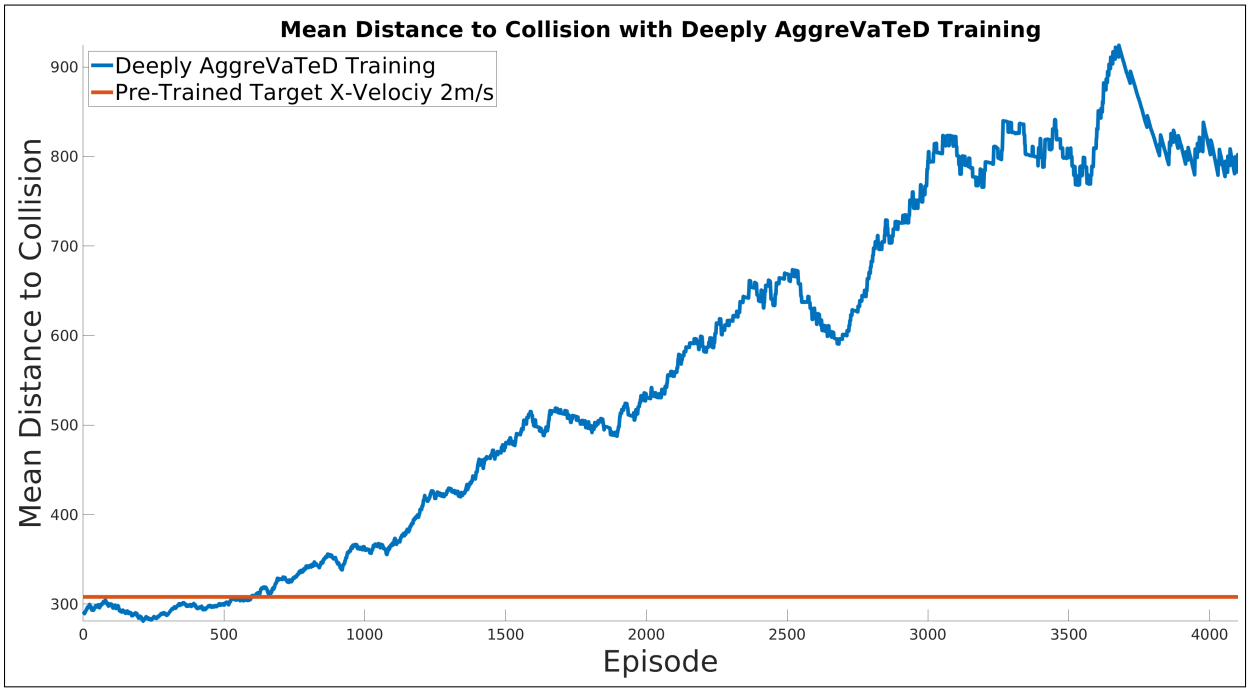

Figure 4.3: 2D Simulated Reinforcement Learning Training Mean Distance to Collision

Upon examining these results, we can see that reinforcement learning significantly improved the mean distance collision but failed to maintain the desired velocity of 2m/s. This implies that since the reward weighting scheme valuing safety over meeting the target velocity. Table 4.2 shows how reinforcement learning compares to the policies trained by imitation learning. Based on the rewards weighting of costs, it suggests that around 1.1 m/s is the maximum X-velocity the robot should travel at to improve safety. However, in terms of the velocity to safety trade-off, the reinforcement learning trained policy does not seem to be significantly superior to the imitation learning results.

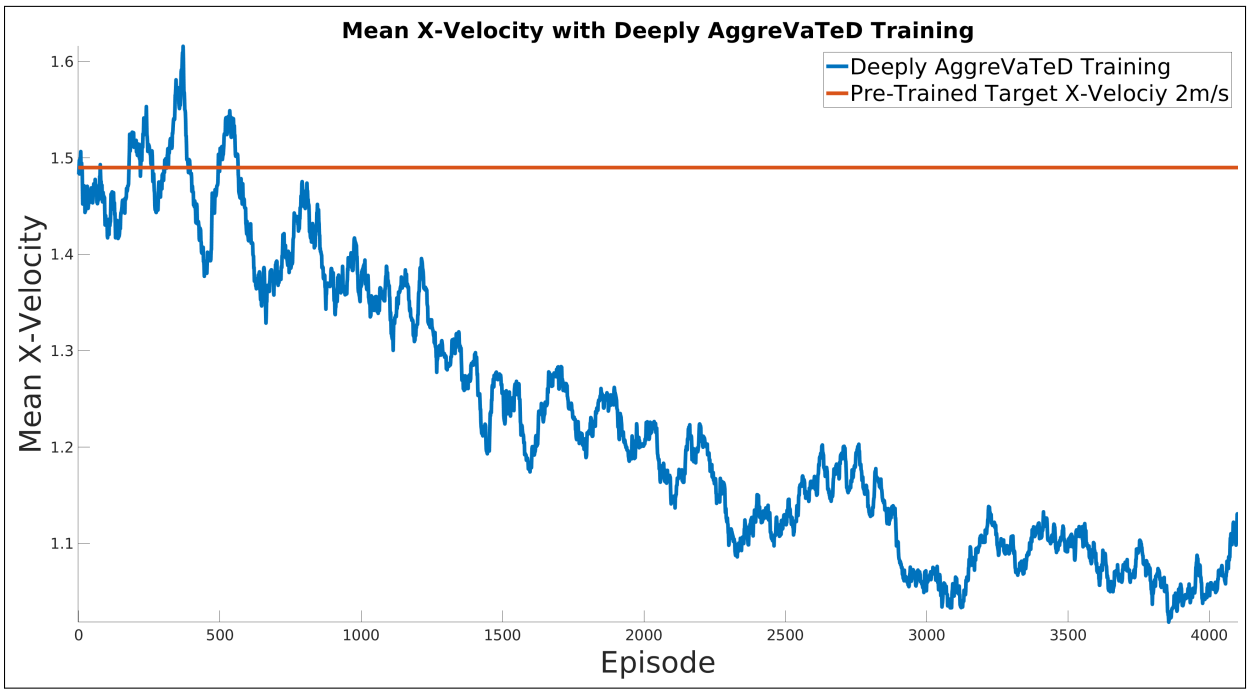

Figure 4.4: 2D Simulated Policy Gradient Training Mean Velocity Per Episode

Comparison of Imitation and Reinforcement Learning				
	Imitation Learning 0.5 m/s	Imitation Learning 1.5 m/s	Imitation Learning 2.0 m/s	Reinforcement Learning
Mean Distance to Collision	1852.3m	448.1m	416.0m	924.2 m
Average X-Velocity	0.461 m/s	1.23 m/s	1.49 m/s	1.08 m/s

Table 4.1: Comparison Between Imitation Learning to Reinforcement Learning in Generated Simulation Evnironments

4.3 Baselines on Simulated 2D Environments

This work will compare the learned policy to a number of different existing solutions for state estimation and control. First, the common solution to velocity estimation for LIDAR based navigation on quadcopters is to use SLAM to estimate the velocity and then wrap a control loop around it with a PID controller. Next, we could also consider the control solution when there is not a state estimate available. In such as situation no feedback is possible so we would simply use a feedforward only system.

4.3.1 SLAM

For the SLAM baseline, we used the off the shelf hector-mapping package from Kohlbrecher et al. [2011] and ran it on the simulated laserscan and then used the velocity estimate from SLAM to provide feedback for a velocity controller. There is no need to apply the very best state of the art SLAM algorithm here since the fundamental limitation here is the sections of geometrically degenerate environment where it is mathematically impossible to estimate the state. However, with higher input resolution, more features may be resolvable resulting in better performance but slower computation with increased resolution.

To make this comparison, we run two different implementations of this, one with the exact same input data as the learned policy which consists of 90 points at 10hz which we will call SLAM with equal resolution. The other is with a much larger laserscan consisting of 540 points at 10hz which we will call SLAM with full resolution.

4.3.2 Feedforward

The feedforward baseline executes roll and pitch actions in the direction of the target waypoint from the planner with a small magnitude tuned to reach a near target average velocity. As expected, this leads to uncontrolled acceleration in the forward direction and does not preform well at all.

4.3.3 Comparisons with Learned Policy

Figure 4.5 depicts an improvement of almost 10 times the mean distance to collision increase by using the learned policy over even the full resolution SLAM on the simulated 2D generated environments. Given

Additionally, although Figure 4.5 suggests that we are meeting our first objective of not crashing we need to confirm that the learned policy is still making significant progress in the desired goal direction. Table 4.3.3 compares the mean distance traveled in each episode by the policy and each of the baselines. Interestingly, the baselines all travel further in each episode, but since they are often out of control, they also crash significantly more often.

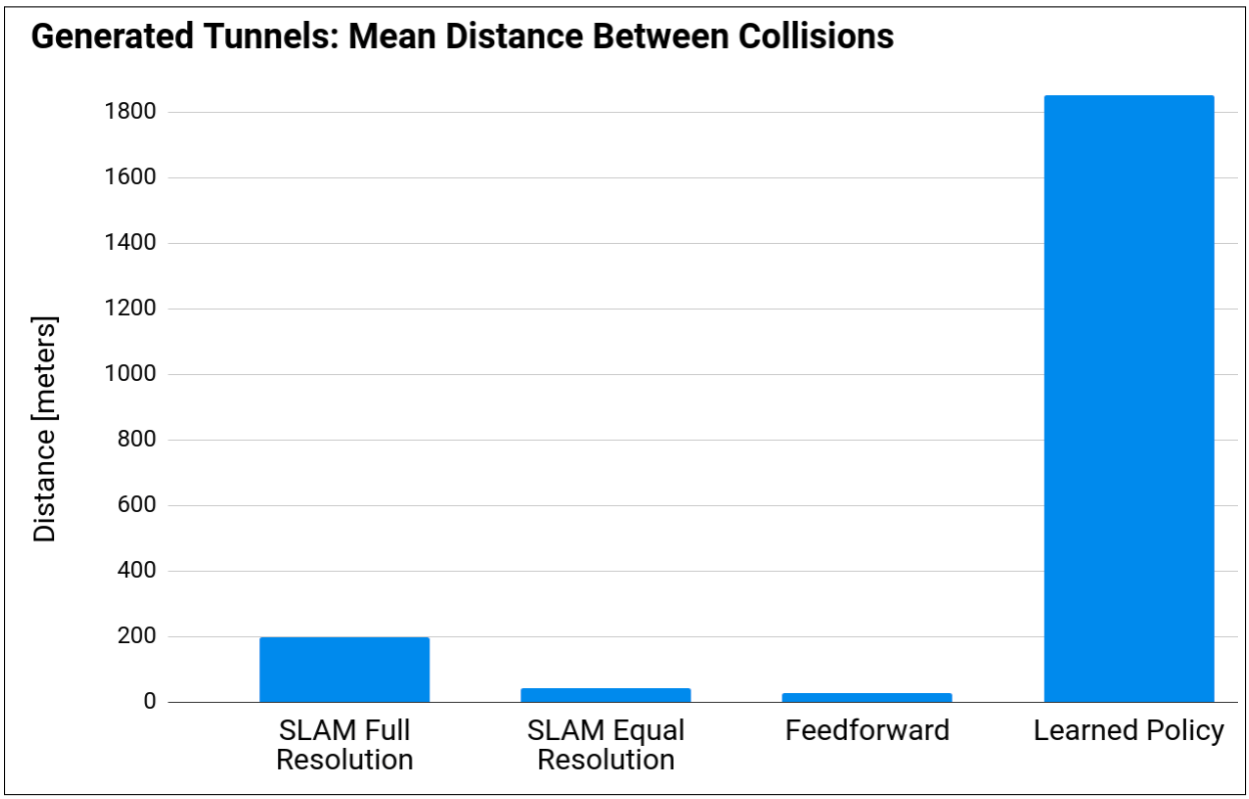

Figure 4.5: Comparison of mean distance between collisions of baselines and learned policy with a target velocity of 0.5 m/s. Averages were computed over 300 randomly generated episodes on unique environments. (PID expert not depicted since it achieves extremely high distances between collisions)

Comparison of Baseline to Learned Policy at 0.5m/s Target Velocity					
	Learned Policy	SLAM Full Resolution	SLAM Equal Resolution	Feedforward	PID Expert
Mean Distance To Collision	1852.3m	198.7m	42.6m	28.1m	N/A
Average Distance Traveled	27.7m	38.1m	42.6m	28.1m	28.75m

Table 4.2: This table compares the learned policy to each of the baselines with a target velocity of 0.5 m/s averaged on over 300 episodes.

4.4 Imitation Learning on Simulated 3D Environments

Furthermore, we also show that the same framework can be applied to learn a policy which also works on 3D environments if the robot is equipped with a 3D LIDAR as well. Figure 4.6 shows the policies reward increases as the training progresses. Since the training progress was a bit more difficult to measure in 3D with the previous metrics, we combined them into a single reward signal to monitor the the training progress.

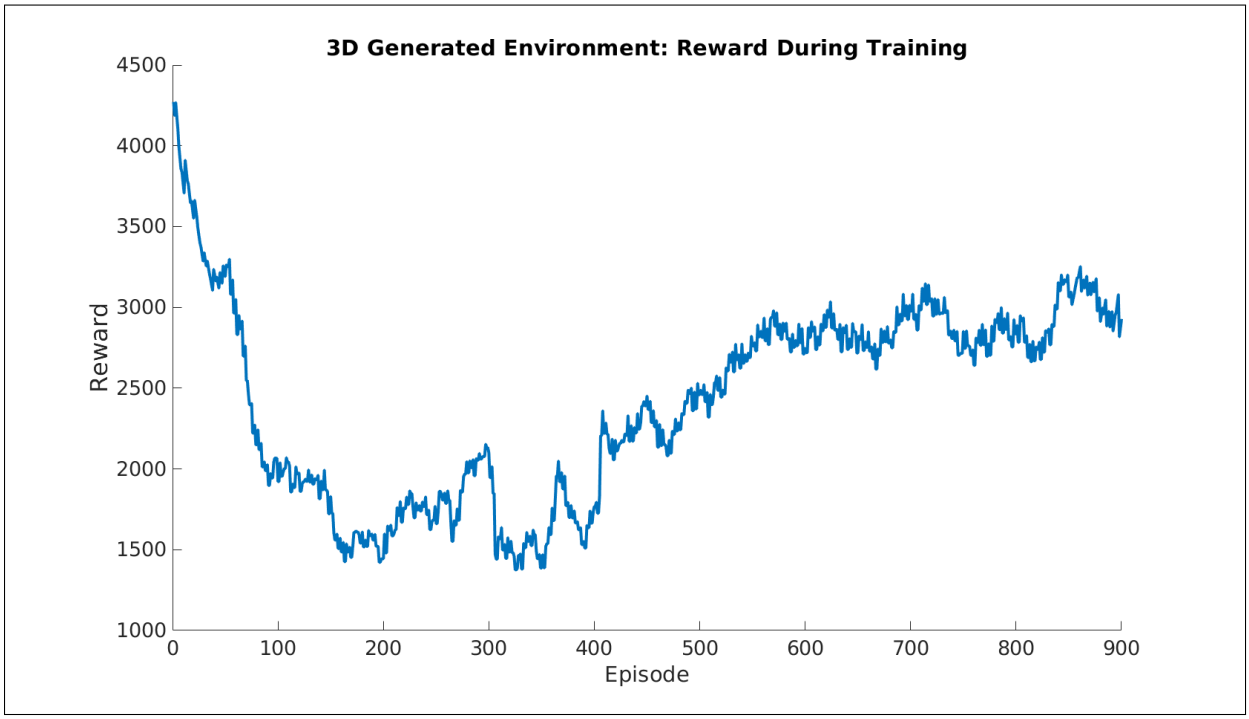

Figure 4.6: 3D Simulated Imitation Learning Training Results on Generated Environments

4.5 Robot Testing with 2D LIDAR

In addition to simulated experiments, we also explored how well the policy trained in simulation could be directly transferred to running on board a real robot. For this, we did experiments with flights in two different environments show in Figure 4.7. We used a DJI Matrice 100 shown in Figure 4.8

Figure 4.7: Left: Quadcopter flying with learned policy in a coal mine. Right: Quadcopter flying with learned policy in hallway environment.

Experiments were done with several different sets of trained weights for different target velocities. Table 4.3 shows a comparison between simulation results and initial flight tests results in the test environments.

Although the initial results demonstrate significant amounts of successful flight onboard a real robot, there is still a clear difference in performance between simulation and real world flight. One of the main causes for this difference is that there was an unmodeled controller deadzone discovered during testing that was not incorporated into simulation. This was particularly problematic for policies trained for lower velocities such as the 0.5 m/s policy since it commanded very small roll and pitch angles that were actually ignored by the controller in real life. Additionally, at higher velocities, the simulated dynamics model begins to break down and performance also decreases.

Furthermore, the mine environment proved to be particularly challenging due to strong magnetic field interference. The current policy does not provide any yaw control and assumes that the flight controller can successfully maintain a global heading. However, magnetic interference from various sources in the environment caused significant yaw in almost every flight resulting in early termination of the test due to collision or no path being found.

Figure 4.8: Robot equipped with 2D Lidar and Nvidia TX2

Comparison Of Simulation and Real World Performance						
	Simulation 0.5 m/s	Simulation 1.5 m/s	Simulation 2.0 m/s	Real 0.5 m/s	Real 1.5m/s	Real 2.0m/s
Mean Distance To Collision	1852.3m	448.1 m	416.0	40m	450m	73 m
Average Velocity	0.461 m/s	1.23 m/s	1.49 m/s	0.3 m/s	0.9 m/s	1.9m/s

Table 4.3: Table Comparing Simulation Results to Real World Performance

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In summary, the first part of this work defines how a learned LIDAR based reactive flight control policy can be integrated into a system pipeline to unify the state estimation, position and velocity control subsystems. This allows for direct computation of control accelerations without the need for an explicit velocity or position estimate.

Next, we demonstrated that an RNN based policy architecture can encode recent laserscan measurements and a target waypoint to produce robust control outputs. This design additionally incorporated a heavily down-sampled input state in order to prevent overfitting and to help maintain performance between simulated and real world environments.

Additionally, we explored the two training paradigms of imitation learning and reinforcement learning for training this policy. To acquire sufficient training data for these methods, a procedurally generated over 10,000 simulation environments in combination with a PID controller expert were employed to label over 3 million data points over the course of the training process.

Finally, we evaluated the learned polices both in simulation and on a real robot. In simulation, the learned policies out preformed all state of the art alternatives by a wide margin. Furthermore, initial real world results demonstrated that these policies learned in simulation could be directly transferred to real world robots capable of flying through tunnels and corridors for hundreds of meters without crashing. Future research and experiments should be able to quickly improve the robustness of these learned policies.

5.2 Future Work

5.2.1 Learning to Generate Training Environments

Perhaps the most interesting option for future work would be to extend the environment generation process to produce very realistic environmental structures. Recent work with Generative Adversarial Nets (GANs) by Goodfellow et al. [2014] demonstrate the ability of generative models to produce very realistic images. Applying such an approach to LIDAR data, we could learn a generative terrain model based on real world data. Then we could procedurally generate re-

alistic environments by using a generative model to generate realistic depth textures and global structure for the simulator.

This would not only provide a closer match between the training environment and testing environment, but it would also allow the robot to learn to fly in a new environment just by being walked through similar environments.

5.2.2 More Realistic Simulation Dynamics

In the experiments section, we observed that real world performance was lower than simulation performance due to a number of unmodeled dynamics, such as the controller deadzone. Adding details such as this into the simulation should greatly improve the gap between simulation and real world performance.

5.2.3 Advanced Control

As we push the limits and increase the target velocity, the simplified dynamics assumptions made by the grid based A* planner and expert PID controller break down and performance decreases. In order to train for faster target velocities, we would need to use a much more advanced planner and expert controller such as what was used by Kaufmann et al. [2018] for their drone racing problem.

Bibliography

Ronald C Arkin. Reactive robotic systems. 1995. 2.1

Johann Borenstein and Yoram Koren. The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, 7(3):278–288, 1991. 2.1

Shreyansh Daftry, Sam Zeng, Arbaaz Khan, Debadeepta Dey, Narek Melik-Barkhudarov, J Andrew Bagnell, and Martial Hebert. Robust monocular flight in cluttered outdoor environments. *arXiv preprint arXiv:1604.04779*, 2016. 1.3, 2.1

Dhiraj Gandhi, Lerrel Pinto, and Abhinav Gupta. Learning to fly by crashing. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 3948–3955. IEEE, 2017. 2.1, 2.3

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. 5.2.1

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 7(1), 2015. 3.2

Christoforos Kanellakis and George Nikolakopoulos. Evaluation of visual localization systems in underground mining. In *Control and Automation (MED), 2016 24th Mediterranean Conference on*, pages 539–544. IEEE, 2016. 1.1

Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza. Deep drone racing: Learning agile flight in dynamic environments. *arXiv preprint arXiv:1806.08548*, 2018. 3.3.1, 5.2.3

Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007. 2.2

S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE, November 2011. 2.2, 4.3.1

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 3.2, 3.3.4

Andress Nuchter, Hartmut Surmann, Kai Lingemann, Joachim Hertzberg, and Sebastian Thrun. 6d slam with an application in autonomous mine mapping. In *Robotics and Automation, 2004.*

Proceedings. ICRA'04. 2004 IEEE International Conference on, volume 2, pages 1998–2003. IEEE, 2004. 1.1

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, 2011. 3.3.4

Stéphane Ross, Narek Melik-Barkhudarov, Kumar Shaurya Shankar, Andreas Wendel, Debadeepta Dey, J Andrew Bagnell, and Martial Hebert. Learning monocular reactive uav control in cluttered natural environments. *arXiv preprint arXiv:1211.1690*, 2012. 1.3, 2.3, 3.3.1

Inkyu Sa, Mina Kamel, Raghav Khanna, Marija Popovic, Juan Nieto, and Roland Siegwart. Dynamic system identification, and control for a cost effective open-source vtol mav. *arXiv preprint arXiv:1701.08623*, 2017. 3.6.3

Steven Scheding, Gamini Dissanayake, Eduardo Mario Nebot, and Hugh Durrant-Whyte. An experiment in autonomous navigation of an underground mining vehicle. *IEEE Transactions on robotics and Automation*, 15(1):85–95, 1999. 1.1

Sebastian Scherer, Sanjiv Singh, Lyle Chamberlain, and Srikanth Saripalli. Flying fast and low among obstacles. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2023–2029. IEEE, 2007. 1.3

Gary Shaffer and Anthony Stentz. A robotic system for underground coal mining. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 633–638. IEEE, 1992. 1.1

Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. *arXiv preprint arXiv:1703.01030*, 2017. 3.3.3, 3.3.5

Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. *arXiv preprint arXiv:1509.06791*, 2015. 2.4

Weikun Zhen, Sam Zeng, and Sebastian Soberer. Robust localization and localizability estimation with a rotating laser scanner. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 6240–6245. IEEE, 2017. 1.2, 2.2