Supervised Learning of Corrective Maneuvers for Vision-Based Autonomous Flight

DHRUV MAURIA SAXENA
dhruvsaxena@cmu.edu

AUGUST 2017

CMU-RI-TR-17-55

Robotics Institute
CARNEGIE MELLON UNIVERSITY

Thesis Committee:
Martial Hebert, Chair
Kris Kitani
Abhijeet Tallavajhula

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

Abstract

The ability of autonomous mobile robots to react to and recover from potential failures of on-board systems is an important area of ongoing robotics research. With increasing emphasis on robust systems and safe navigation, mobile robots must be able to respond safely and intelligently to dangerous situations. Recent developments in computer vision have made autonomous vision based navigation possible. However, vision systems are known to be imperfect and prone to failure due to variable lighting, terrain changes, and other environmental variables. The notion of introspection for mobile robots has been developed recently which provides autonomous agents with a self-evaluating capability. This allows them to assess the quality of decisions made by them in the future based on the present input given or available to them. This thesis focuses on the situation when an agent is in a situation where the input available is unreliable (for whatever reason), and therefore any action taken using that input will also likely be unreliable. In this paradigm, we propose two different solutions.

First, we describe a system for learning simple failure recovery maneuvers based on experience. A failure instance is one where the input data is unreliable. This involves both recognizing when the vision system is prone to failure, and associating failures with appropriate responses that will most likely help the robot recover. We implement this system on an autonomous quadrotor and demonstrate that behaviors learned with our system are effective in recovering from situational perception failure, thereby improving reliability in cluttered and uncertain forest environments.

While the first solution only looks at recovering after a failure has been detected, we also consider the case where we pre-emptively avoid failures by proactively executing a 'recovery' maneuver if our system believes that for the current input, reliable or not, it will improve performance. This is essentially a continuous case extension of the previous solution which looked at discrete changes between a non-failure and failure mode. Again, we evaluate our performance on an autonomous quadrotor in flight through a outdoor forest environment.

Acknowledgements

While I might be the primary author for this document, there is a small group of people that have contributed in one way or another to its contents. It was a pleasure and privelege to spend every day over the last two years working with some subset of this group of people. The primary recipient of my gratitude is my advisor for this thesis, Martial Hebert. During our meetings there were moments when he seemed almost omniscient, which was extremely helpful for someone relatively new to the rigour of academic research like me. Martial is as solicitous about his students' progress as one could hope for. He gave me freedom to dictate my research, and helped me get things back on track when I inevitably strayed or felt lost. If I can be excused a moment of appropriating the parlance of my generation, I will say that Martial is a cool dude.

I worked on BIRD with an amazing group of students. Shreyansh Daftry, who is probably sitting in some exotic corner of the world as I type this, laid the foundation that this thesis builds upon while he was a Master's student at Carnegie Mellon. Vince Kurtz, who spent a summer on the project on an internship, helped develop and refine the early part of this research. Arbaaz Khan worked on BIRD before and during my time at CMU, lost a number of FIFA games to me, and also put in place part of the data collection and training protocol I use. Sam Zeng is a licensed UAV pilot, so his role and its importance during in-flight experiments cannot be understated.

I would be remiss not to acknowledge the 23 (going on 24) years of unconditional love and support my parents have given me. I love them and everyday I strive to make them prouder of me than the last. My brother, who I have been taller than for well over 5 years now, is the other constant in my life. There is a constant flow of love, admiration, respect, support and humour between us - traits of a strong sibling relationship.

Finally, to all of my friends at CMU and back home in India - old, new and unknown alike - thank you!

Table of Contents

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

Contents

Micro Aerial Vehicles (MAVs) are one of the most popular robotics research platforms, along with being a successful commercial and hobby product. They span the gamut from platforms that can fit in the palm of a human hand, to larger platforms that can carry several kilograms of payload, to platforms capable of flying at over 150 $kmph$, and capable of recording production grade 4K video. Add on the fact that MAVs can be equipped with any sensor imaginable, the capabilities of the platform are endless.

Given the form factor, MAVs are easy to transport, deploy, operate and recover. They can prove to be useful in any environment imaginable, provided that it is safe to fly. In the hands of a well-trained remote pilot, the usefulness of MAVs cannot be under-estimated. However, as we use these platforms over longer distances, in harsher conditions, bad illumination settings, and generally unsafe environments for humans, making these systems autonomous, safe and robust will be of utmost precedence. Regardless of the skills of the available pilot, there will be a need for autonomous MAVs that can fly better

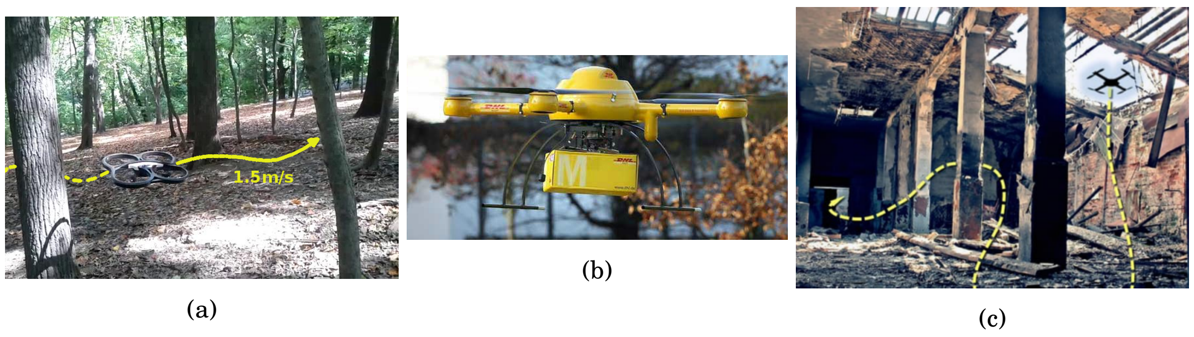


Figure 1.1: *Quadrotors can be used for a variety of applications including (a) flying through dense forests for surveillance or search-and-rescue missions, (b) package delivery in urban environemnts, (c) surveillance and reconnaissance missions as part of a disaster response effort.*

than the best human pilot. Figure 1.1 shows examples of the wide variety of applications in which quadrotors can prove to be useful.

1.1 Autonomous Flight

Autonomous MAV flight has been a topic of research for several years. The various components that go into this complicated task have been at the forefront of robotics research. Planning algorithms, perception systems, robust control, multi-agent systems etc. all form a part of the modern day understanding of autonomous MAV flight. There has been a lot of work in generating time-optimal trajectories for quadrotors by utilizing the differential flatness property of quadrotor dynamics [29]. It is possible to solve quadratic programs to generate polynomial trajectories in the time domain that satisfy the constraints placed on the dynamics [33]. Vision-based navigation of quadrotors is another active area of research [15, 5]. These methods use cameras as exteroceptive sensors to navigate through potentially unstructured and/or unknown environments. Monocular cameras, RGB-D cameras, and stereo cameras have all been used for this purpose. For a multi-agent team of quadrotors, the problem of optimally assigning goals to the individual agents while also avoiding all collisions [40] is a higher-level task that needs to be solved before the agents execute their individual trajectories. It is clear that there are several pieces that need to work together to achieve safe autonomous flight of quadrotor(s) in unknown and unstructured environments in the presence of obstacles.

There are two major environments that quadrotors might operate in - indoors and outdoors - that present their own unique challenges. Indoors, we might have the benefit of using motion capture systems or other external cameras to aid state estimation and localisation for autonomous flight, since there is no GPS signal available. Outdoors, quadrotors can use GPS and seemingly infinite airspace for safe flight. There is obviously research in both these areas that tries to use as little external information as possible, and relies on onboard computation to come up with viable plans to perform the intended task. This thesis is a small part of that effort, and uses a quadrotor flying outdoors in a dense, GPS-denied forest as its research platform.

In terms of the overall picture of autonomous flight, it is imperative to recognise the efforts that have been put in by the robotics community to tackle a vast variety of applications. For example, they have been used for mapping indoor spaces individually and collaboratively [16, 17, 13]. Helicopters are another popular platform capable of fast, agile flight among cluttered obstacles [35] and acrobatic maneuvers [7]. Formation flight of close to 50 nano-quadrotors has been achieved [32]. This could prove to be extremely useful for mapping and surveying tight spaces packed with obstacles.

For outdoor applications, the challenge lies in the fact that the use of external sensors is not viable. We cannot set up an outdoor motion capture system over an area even somewhat close to a meaningful size, nor is it feasible to put up 'beacons' of some sort to aid with state estimation and localisation. All sensing capabilities need to be on-board the MAV. Machine learning algorithms have also been used for autonomous quadrotor flight in outdoor environments [34, 18]. As we explore applications in previously unknown spaces, these learning algorithms could prove to be useful in terms of generalizing to a wide variety of environments. In this thesis we are limited by the unavailability of a GPS signal inside a dense forest. Thus, the goal is to fly through a forest while avoiding collisions with trees, bushes, and other foliage.

1.2 Safe and Robust Systems

As we approach higher and higher levels of autonomy in robotics systems, there is a desire and need for these systems to be safe during the entirety of their operation. This includes being robust to 'noise', whether that be in terms of environmental disturbances, sensor imperfections, or general uncertainty about the robot's belief over the world and itself. There has been work in generating motion plans for robots with implicit safety guarantees which ensure that at any given point along the plan, the robot will be able to generate and execute a safety maneuver if necessary [36, 42]. The drawback of these methods is that they rely on complete knowledge of the distribution of obstacles in the environment, and have not been validated in the presence of unknown or dynamic obstacles. There is also related work in using an offline computed library of safety maneuvers and selecting one of these when the need arises [3]. Neither of these approaches consider the case when the sensor information is unreliable, which precludes both generation and selection of emergency maneuvers since safety can no longer be guaranteed.

On a slightly different note, researchers have explored the theoretical limits of fast flight through cluttered obstacle distributions [23, 6]. Such analyses make assumptions about the underlying distribution of obstacles in the environment in order to calculate their bounds on robot speed and planning resolution.

1.3 BIRD

The majority of research presented in this thesis was conducted as part of the BIRD Multidisciplinary University Research Initiatives (MURI) project under the Office of Naval Research (ONR) grant on “Provably-Stable Vision-based Control of High-speed Flights through Forest and Urban Environments”. It is common to look to nature for inspiration during the development and deployment of physical systems. BIRD, appropriately named, in particular looks toward the natural ability of agile birds that can swiftly maneuver through a dense forest during very high-speed flights. Figure 1.2 is an image of a northern goshawk flying at speed through a forest. Thus, as far as outdoor flight in cluttered environments is concerned, this is the holy grail and if it can be achieved repeatedly, we can think about planning and executing complex missions in these environments. Our particular quadrotor platform, which will be introduced and described in detail in Chapter 2, has taken several steps in this direction over the last few years. This thesis is a continuation of these efforts and takes one more step towards achieving safe, autonomous outdoor flight.

1.4 Thesis Contributions

With regards to the problem of autonomous outdoor flight, in this thesis we look at the particular problem of dealing with unreliable information from the on-board monocular camera intelligently. We call the ability to identify unreliable visual inputs ahead of time ‘introspection’ [9].

Figure 1.2: *The northern goshawk is capable of flying through dense forests at high speeds. The level of maneuverability exhibited by these birds represents the peak performance quadrotors can hope to achieve.*

1.4.1 Recovery Maneuver Selection

Our first approach towards dealing with these unreliable visual inputs (which we call ‘failures’) looks at associating a failure image at test time with the best recovery maneuver. The goal is to execute a maneuver which provides the greatest chance of putting us in a position such that the visual input is once again reliable so that normal flight can resume. We present a framework of data collection and associated training of classifiers that help us establish these relations between a failure and the best recovery maneuver. The set of recovery maneuvers is created by leveraging domain knowledge available to us.

1.4.2 Continuous Case Extension - *Corrective* Maneuvers

Previously, we only execute recovery maneuvers if and when the visual input is unreliable. Since the set of these maneuvers is predetermined, there may not be any good maneuver for a failed image. Even though it might be mission-critical to execute a recovery maneuver at this moment, there is no good way for us to make a decision about which one to execute. We extend the previous approach by removing the threshold on images that have failed or not when making these decisions. By considering all incoming images in conjunction with a measure of their quality, we now want to execute the *corrective* maneuver with the greatest chance of improving quality significantly.

1.4.3 Outline

This thesis is organised as follows. Chapter 2 introduces our quadrotor and on-board monocular visual navigation system. We introduce our sub-systems of monocular depth estimation and introspection. We briefly refer to past work done as part of the BIRD project during this discussion. In Chapter 3, we talk about our data collection, training, and inference protocols for selection of recovery maneuvers. Chapter 4 presents details about the extension to corrective maneuvers and the use of cost-sensitive classifiers for this purpose. Finally, in Chapter 5 we conclude the work done in this thesis and discuss alternate formulations of the problem this thesis explores, along with future research directions.

CHAPTER 2

VISION-BASED AUTONOMOUS FLIGHT

Contents

The challenge of autonomous flight for MAVs in dense, cluttered environments lies in the fact that there are payload and power constraints which limit the number and type of sensors that can be put on the platform, and also the flight time and speed. As a result, the specific configuration of the MAV needs to be carefully designed, with the necessary sensors, batteries, and computational power on-board so that the desired missions have the greatest chance of success. In this chapter, we give details about the quadrotor platform that we used for this thesis, along with details about some of the sub-systems we use in our perception, planning and control pipelines.

2.1 System Overview

The research platform used for this thesis is a quadrotor customised for the purpose of flying through dense forests. We have assembled together a number of commercially available parts on the platform. As an overview, the quadrotor contains two monocular cameras and a single-beam lidar sensor. There is also a single-board computer and autopilot on-board.

2.1.1 Quadrotor Platform

The primary hardware platform is a modified 3DR ArduCopter with an on-board Odroid XU3 quad-core ARM processor which runs Ubuntu 12.04 and ROS Fuerte (the same as our groundstation). A Microstrain 3DM-GX3-25 IMU is used to help correct the drift and noise of the ArduPilot integrated IMU. There are two monocular cameras on the quadrotor. A downward facing PlayStation Eye camera (320×240 at 30 fps) is used for real-time pose estimation. The image stream from a front-facing high-dynamic range PointGrey Chameleon camera (640×480 at 15 fps) is relayed to the base station, where the perception and planning modules use it for monocular navigation. These modules use a semi-dense 3D reconstruction of the scene to select the optimal trajectory which is then sent back to and executed by the quadrotor. The lidar sensor is used to recover the true scale (altitude) of the environment. The quadrotor platform used for this work is shown in Figure 2.1.

2.2 Monocular Depth Estimation

For monocular depth estimation, we use a semi-dense visual odometry approach following the method of Engel *et al.* [14]. Pixels with large disparity between frames are selected along with suitable reference frames. Line searches are used to refine the disparity estimates which are converted to an inverse depth map (inverse depth is directly proportional to the disparity). This is continuously propogated to future image frames to calculate their relative poses. In order to recover the true scale of the images being observed, measurements of the distance travelled by the quadrotor according to visual odometry are corrected using the same measurements from a single beam metric lidar sensor.

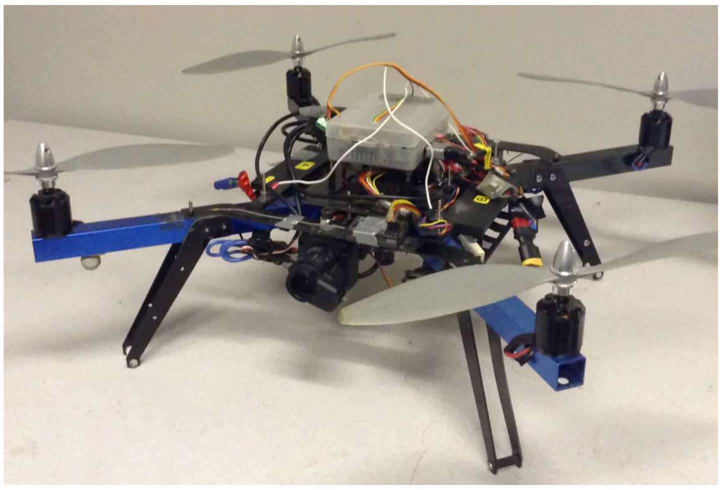


Figure 2.1: *Our customised quadrotor hardware platform. The front-facing monocular camera and on-board computer (in the plastic case on top) are easily visible.*

2.3 Introspection

The idea of introspection is central to the field of psychology [24]. The analogy in robotics is the model representation of a robot's current operational state[1]. This idea was first introduced by Morris *et al.* [30] who used it in an information-theoretic setting to improve a robot's decision making capability when it became uncertain of its operational state. Recently, the idea of introspection has been adopted for perception systems in terms of quantifying the predictive variance of classification and detection algorithms [21, 20, 39]. The idea there is to use algorithms cognizant of the fact that the assumption of independent and identically distributed (*iid*) data is usually not valid for real-world robotic systems. These algorithms can then be utilized in an active learning framework [39] to further improve predictor accuracy. The notion of introspection that we utilize in this work is best described by Daftry *et al.* [9] who obtain a confidence estimate by analyzing the input to the system. The key difference is that this approach makes it possible to quantify the reliability of input data which directly affects the quality of the prediction made.

A deep spatio-temporal convolutional network based on [37] (with individual architectures similar to AlexNet [25]) is used to generate a good feature vector representation

[1]**Operational state** is a high-level representation of the configuration of various sub-processes in a robotic system. It should not be confused with an instance from the *state space* model of a robot. More details can be found in [30].

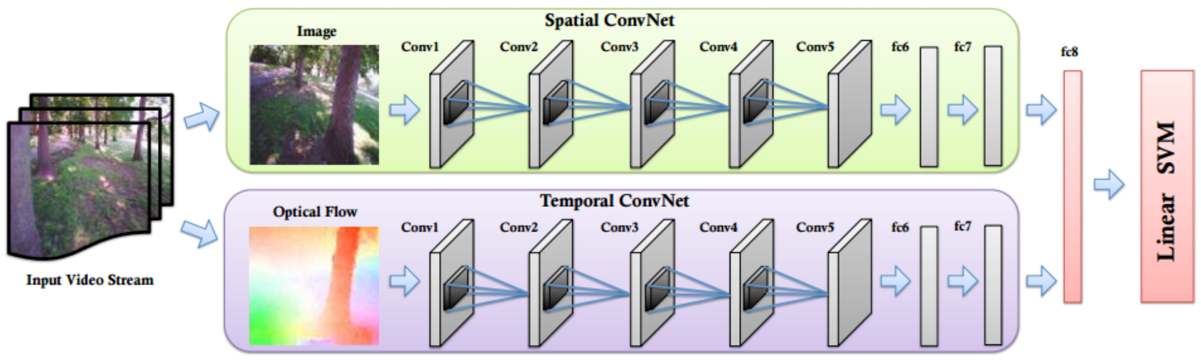
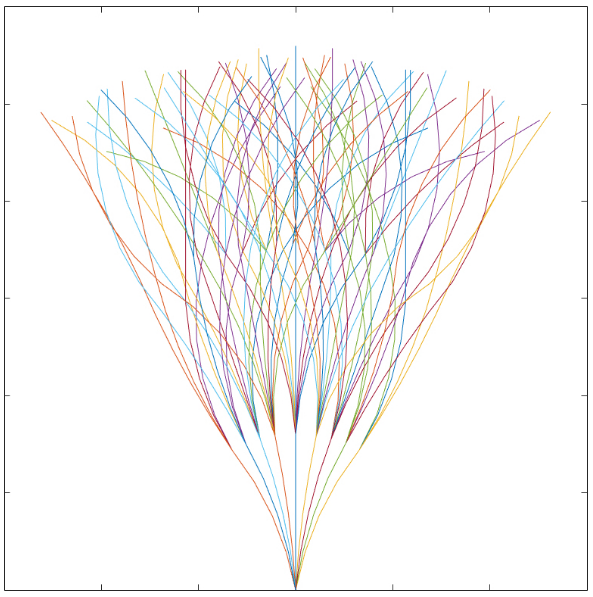


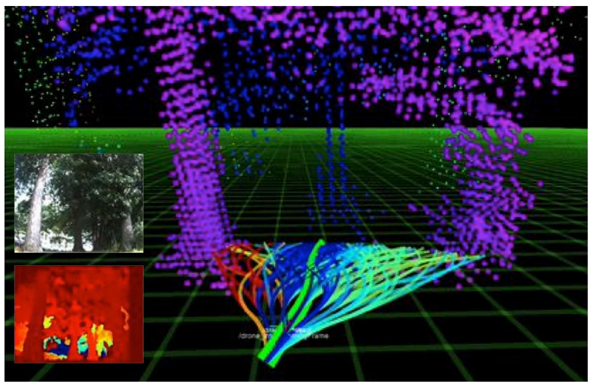
Figure 2.2: *Our customised quadrotor hardware platform. The front-facing monocular camera and on-board computer (in the plastic case on top) are easily visible.*

of the images from the front-facing camera stream. The CNN architecture is shown in Figure 2.2. These features are passed through a learned linear SVM which predicts a failure score $y \in [0,1]$ as output, where a higher score for an image indicates greater probability of failure of the perception system for that particular input image [9]. A failure of the perception system occurs when it is unable to reliably label the trajectories in our library as *collision-free* or *collision-prone*. The feature vectors $\mathbf{x}_i \in \mathbb{R}^d$ from the `fc7` layer of the spatial CNN for each image i are later used during training and selection of recovery/corrective maneuvers described in Chapters 3 and 4.

The two-stream CNN is trained against ground-truth from a stereo camera. The base task we are trying to accomplish is to first generate good (monocular) depth images, and then use this to calculate the probability of collision along each of the trajectories in our library. If this probability is above a certain threshold, the trajectory is deemed to be *collision-prone*, otherwise it is *collision-free*. We use a stereo-camera as ground truth since that is the best depth image we can hope to achieve in an outdoor environment, without a prior map. Given both a stereo depth image and our estimated monocular depth image, ideally we want the exact same labelling of trajectories. If the discrepancy between the two sets of labellings is too high, we deem that particular monocular image to be a failure. The two-stream CNN is then trained against a $0-1$ loss to predict the correct label - *failure* (predict 1) or *not* (predict 0). For more details on training the introspection CNN, please refer to [8].



(a)



(b)

Figure 2.3: *(a) Library of 78 trajectories optimally sub-sampled for maximal coverage area from a larger set of 2401 trajectories. (b) Receding horizon control using a ground-truth depth image from a stereo camera. Trajectories in red have the greatest chance of collision, while the thick light-green trajectory is chosen as the best in this case.*

2.4 Planning and Control

We use a receding horizon planner that selects the current best trajectory out of a library of 78 optimally sampled motion primitives [19]. The best trajectory is selected such that a weighted sum of certain parameters is minimized. These include probability of collision along the trajectory, deviation from current heading, and deviation from desired/goal heading. These costs are calculated using the depth map obtained from the perception module. Once a trajectory has been selected, it is sent over to the quadrotor that uses a pure-pursuit PD controller to track the trajectories. Our library of 78 trajectories and a snapshot of the planner running on ground-truth stereo images is shown in Figure 2.3.

2.5 Past Work in BIRD

The problem of monocular vision based autonomous flight in a dense forest has been approached in many different ways in this research project. The first approach was to learn a policy that would be able to mimic a human expert pilot to the best possible degree, given the data the learning procedure had access to. This method uses the DAgger algorithm [34], which is a no-regret imitation learning algorithm that helps us learn a purely reactive obstacle (tree) avoidance policy. By repeatedly querying the expert, we are able to regress to left-right velocity commands from image features. However, this method presented some interesting human-robot interaction challenges when querying the expert during training.

Next, we looked at two different methods of non-linear regression for depth prediction. The first relies on budgeted selection of image features for training the regressor and using the system uncertainty to combine multiple depth predictions for better performance [11]. It used a least-squares based non-linear regression [1]. We also experimented with getting rid of the budgeted feature selection and least-squares non-linear regression, and replacing it with a convolutional neural network (CNN) trained against ground-truth stereo depth images with a square-loss between the final `fc8` layer and raw stereo depth image [8].

Eventually, we switched to a semi-dense visual odometry based method for monocular depth estimation [10]. This was augmented with a CNN for introspection [9] which is discussed in detail in Section 2.3 above.

2.6 Summary

Our research platform for this thesis and some of the prior work done in this project is described in this chapter. Of note is the fact that we primarily rely on two monocular cameras - a front-facing camera for depth prediction, and a downward-facing camera for pose estimation. We also introduce the introspection pipeline that runs in parallel to the depth prediction and trajectory selection pipeline. Finally, we briefly go over the past research that has been done in this project to illustrate the systematic progress we have made towards achieving safe and robust flight in dense outdoor environments.

CHAPTER 3

Recovery Maneuver Selection

Contents

This chapter contains details about the entire pipeline involved in learning failure responses. This pipeline is shown schematically in Figure 3.1. The following sections describe the process of selecting candidate recovery maneuvers, training data collection, predictor training and inference, and finally experimental results.

3.1 Recovery Maneuvers

Maneuvers that will lead to recovery in a variety of failure cases must be chosen using domain knowledge before training the classifier. Maneuvers should be simple, interfere minimally with the robot's high-level task, not expose the robot to additional dangers while perception is unreliable, and provide a reasonable likelihood of ending the unfavorable conditions that led to perception failure. The approach outlined

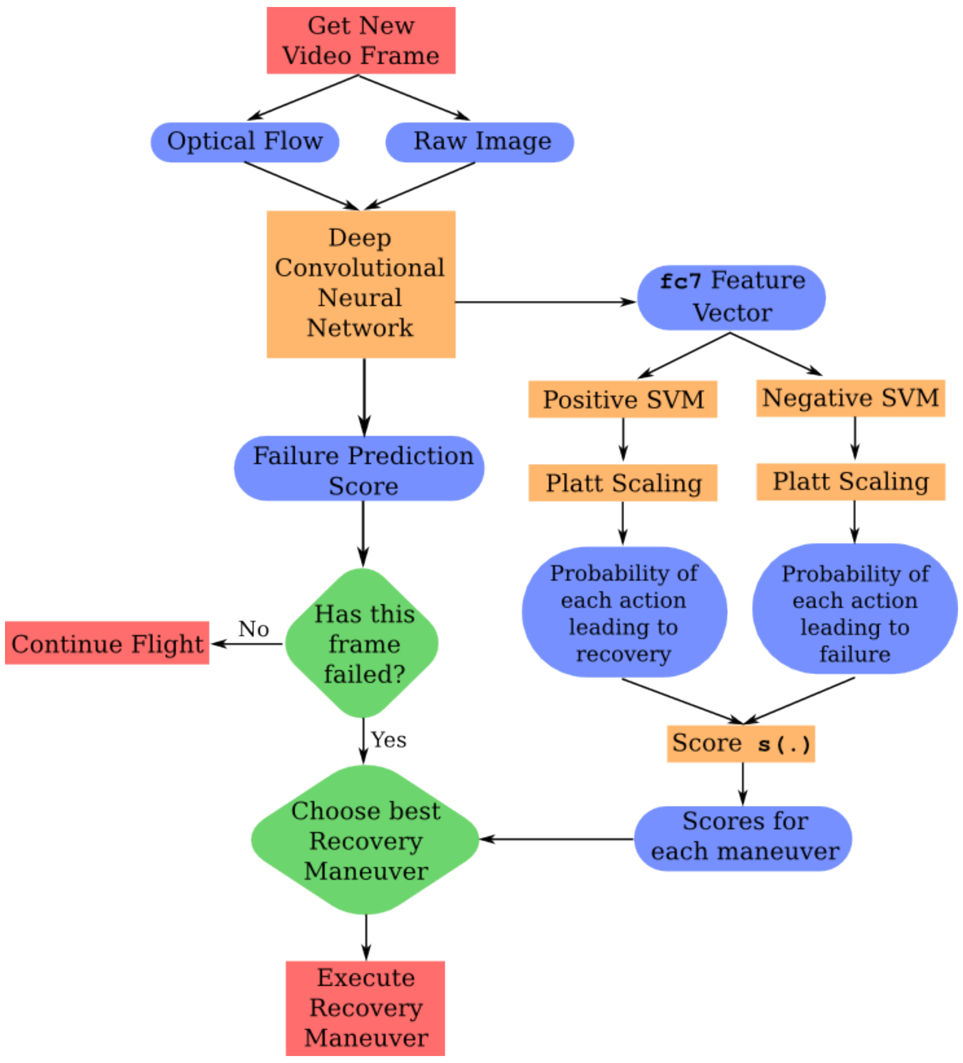


Figure 3.1: *Block diagram for learning failure responses.*

in this thesis requires at least two candidate maneuvers in order to make a decision about which one is better, but no other assumption is made pertaining to these maneuvers. The maneuvers themselves simply act as class labels for the predictors. For the work presented in this thesis, the four recovery maneuvers considered were $Y = \{$*translate right, translate left, rotate right, rotate left*$\}$ as shown in Figure 3.2. During execution, these maneuvers are sandwiched between hover commands sent to the quadrotor. Both translate trajectories cause the quadrotor to translate in the respective directions for $5s$ with a linear velocity of $\pm 0.1 m/s$ as desired. Similarly, the other two trajectories cause the quadrotor to yaw by $\pm 45°$ with a constant angular velocity over $5s$. These values for trajectory execution was hand-selected by us for smooth execution.

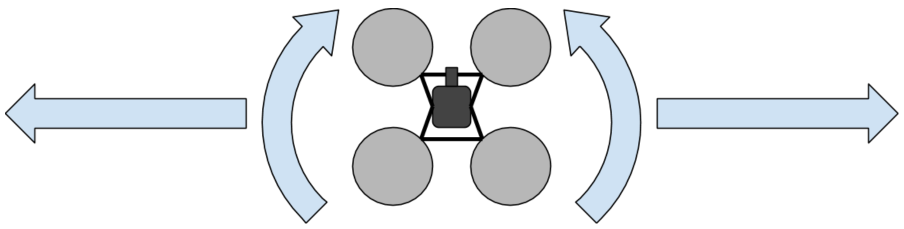


Figure 3.2: *We use four simple failure response maneuvers in this work: translate right, translate left, rotate right, and rotate left. These were selected by considering their ease of execution and resulting impact on the camera scene.*

Rotational trajectories are lower risk since the quadrotor does not move in a way that might cause it to collide with an obstacle, but they may hinder the high-level task of flying for as long as possible in a desired direction. Translational trajectories are comparatively higher risk since they may lead to collision if obstacles are present in the quadrotor's immediate vicinity (to the left or right). However, they are helpful in that they do not interfere much with the high-level task, and provide valuable information to help resolve camera parallax.

3.2 Data Collection

Data collection is a unique challenge when considering the case of failure classification. Ideally, once a perception failure is predicted, all candidate maneuvers would be executed and recovery status would be recorded for each maneuver. Not only would this approach to training data collection be tedious, it could be practically impossible. Often perception failures are specific to exact conditions such as lighting, position and motion and are thus not easily replicated. After executing one recovery maneuver and returning to the approximate pose of the failure, the perception system may no longer be failing. Even if the system still is failing, this new failure will not be quite the same as the one used to test the previous maneuver. For this reason, we propose an alternative data collection method.

To collect training data, when a failure is predicted, only one of the candidate maneuvers is executed and the recovery status at the end of the maneuver is recorded. If the perception system recovers from the failure at any time during the maneuver, that maneuver is considered to have *recovered*. The completed data set then contains the images ($\mathbf{x}_i$) along the maneuver (y_i), and an indicator of whether that maneuver resulted

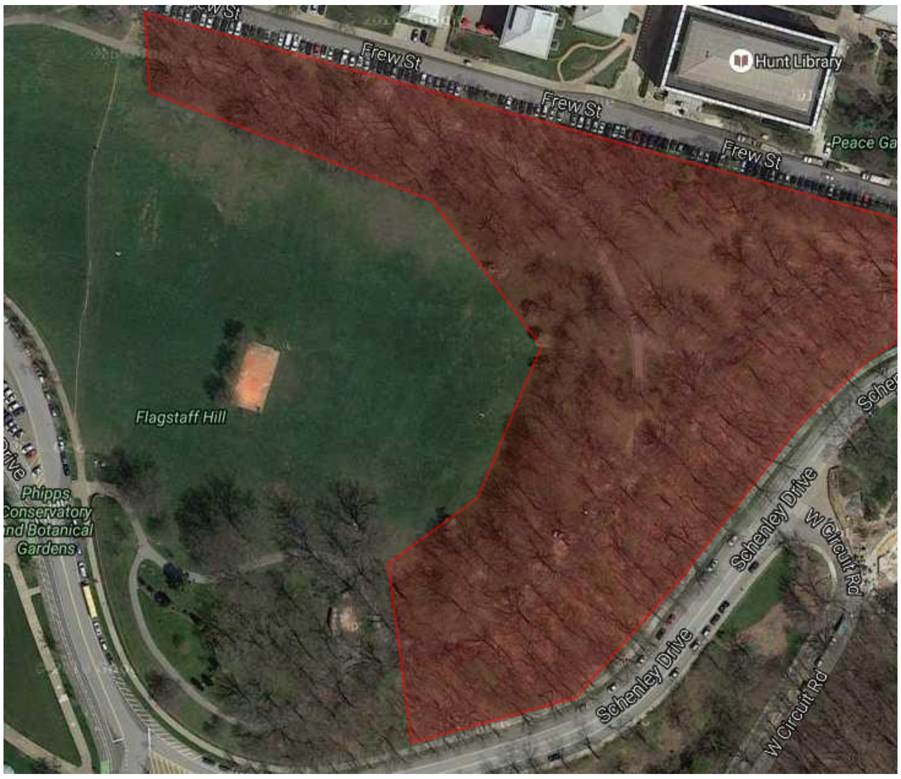


Figure 3.3: *Satellite view of training and testing area (highlited in red) in Schenley Park near Carnegie Mellon University, Pittsburgh, PA, USA.*

in recovery $\{0,1\}$. Training data was collected by holding the quadrotor as we walked through a forest environment, executing one of the four maneuvers every time a failure was predicted. This *handflying* approach drastically reduced the time required to collect data, and was overall logistically easier to carry out than actual flight. The data was collected by handflying for around $20km$ in this way in the area shown in Figure 3.3. Figure 3.4 shows example images from our actual flight tests that triggered perception failures, which were then resolved by one of the recovery maneuvers.

Even with $20km$ of handflying, only 825 failures were encountered: an insufficient number for training a robust predictor $y(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^d \rightarrow Y$ that maps from a high-dimensional feature space $\mathbb{R}^d$ to the set of candidate maneuvers Y [1]. To solve this issue, multiple images from each maneuver were used in the training set. As soon as the system was alerted of a perception failure, all image frames were recorded until either the system recovered or the maneuver ended. Our camera transmits images at 15

[1]For `fc7` feature vectors, $d = 4096$.

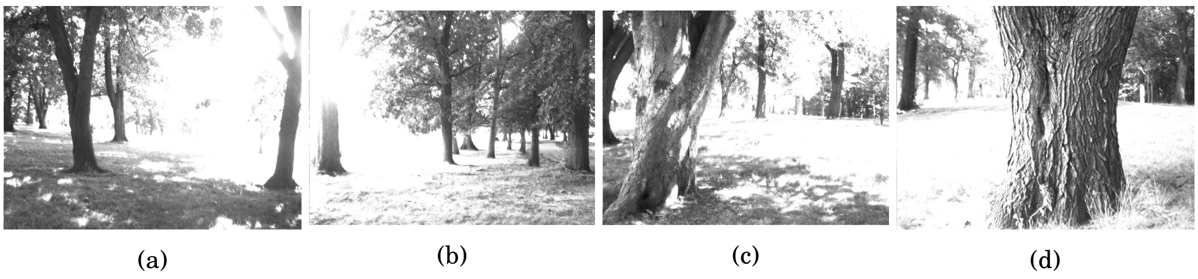


Figure 3.4: *Examples of images that failed during flight. Failures were resolved by (a) rotating left, (b) rotating right, (c) translating left, and (d) translating right. Intuitive causes of failure include direct glare from sunlight (c,d), strong shadows (b, d), open area (b,c), large obstacles preventing adequate parallax (a), and overexposure (a,c,b). Our algorithm used thousands of images like these to learn which trajectories are most likely to recover from perception failures.*

frames per second which results in around $50-100$ frames per maneuver executed. Using every recorded frame would result in unhelpful redundant data [12]. To avoid repeats, redundant images were removed greedily, using the L_1 distance norm between `fc7` feature vectors $\mathbf{x}$ from the deep introspection framework as a similarity metric. Consider the sequence of image feature vectors obtained during a maneuver, $X = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T\}$ where T is the end of the maneuver. Starting with $X' = \{\mathbf{x}_1 \in X\}$, we build $X' \subset X$ such that

$$X' = X' \cup \{\mathbf{x} : |\mathbf{x} - \mathbf{x}'| \geq \epsilon, \mathbf{x} \in X, \mathbf{x}' \in X'\}, \text{until} \frac{|X'|}{|X|} = 0.1 \tag{3.1}$$

where ϵ is a user-defined threshold set to satisfy our choice of $\frac{|X'|}{|X|} = 0.1$. Distance in feature space was used because similarity between these deep features matters to the predictor, not pixel-wise similarity between the images themselves. L_1 distance in particular was used because it provides more accurate results in high dimensional space than traditional Euclidean distance [2].

3.3 Predictor Training

$\mathbf{X} = \{X'_1, \cdots, X'_i, \cdots, X'_N\}$ is the entire data set of images collected. The subscript i refers to the i^{th} maneuver that was executed. For the data that we collected, $N = 825$. This data set is split into two sets $\mathbf{X}^+$ and $\mathbf{X}^-$. $\mathbf{X}^+$ contains all images from trajectories that

Maneuver	**Recovered**	**Failed**
Translate Right	745	631
Translate Left	738	530
Rotate Right	1280	863
Rotate Left	1234	1518
Total		**7539**

Table 3.1: *Number of images for each class in the training set.*

were successful in recovering from the perception failure, while $\mathbf{X}^{-}$ contains all images from trajectories that were unsuccessful in recovering from the perception failure. The exact numbers of images from each class $y \in Y$ in each of these two sets is shown in Table 3.1. Two SVMs are trained independently on these datasets $\mathbf{X}^{+}$ and $\mathbf{X}^{-}$, to predict the associated recovery maneuvers $y \in Y$ which are used as the class labels.

We use the `fc7` feature vectors from the CNN independently for predicting potential failures, and selecting a recovery maneuver. The combination of CNN features and SVMs is a popular architecture for supervised learning tasks [38]. Using SVMs in combination with CNN features for these two independent tasks is a simple and more comprehensible model architecture (than an end-to-end neural network approach) that is able to outperform the existing state-of-the-art for autonomous flight in dense forests.

To give a sense of how many recovery maneuvers were executed during data collection to obtain the dataset in Table 3.1, please refer to Table 3.2 below. During data collection, if at any point during the execution of a maneuver, the image was not in failure anymore (according to the CNN), we stopped execution, and marked that maneuver as 'recovered'. As a result, all frames up until the frame that recovered were added to the *Recovered* data corresponding to that maneuver. Thus, while the two columns in Table 3.1 have roughly equal numbers for each maneuver, during collection, more maneuvers *Recovered* than not (since only a subset of frames from these are considered for addition into the dataset, as opposed to all frames for maneuvers that did not recover). This is shown in Table 3.2.

3.4 Predictor Inference

At test time, when a query image that triggered a perception failure is obtained, Platt scaling [31] is used to convert the arbitrary confidence scores from the two SVMs to two

Maneuver	**Recoveries during Collection**	**Failures during Collection**
Translate Right	99	51
Translate Left	83	34
Rotate Right	78	40
Rotate Left	118	85

Table 3.2: *Each recovery maneuver was executed several times during data collection - all recovered more times than not during this process.*

different probabilities.

We represent $p^{+}(y|\mathbf{x})$ as the probability that the recovery maneuver leading to successful recovery from the perception failure is $y \in Y$. Similarly, $p^{-}(y|\mathbf{x})$ represents the probability that the recovery maneuver leading to failure is y. Note that $\sum_y p^{+}(y|\mathbf{x}) = 1$ and $\sum_y p^{-}(y|\mathbf{x}) = 1$, but in general $p^{+}(y_i|\mathbf{x}) \neq (1 - p^{-}(y_i|\mathbf{x}))\ \forall\ i$.

We use the ratio of the two probabilities obtained from the two SVMs as the scoring function $s(\mathbf{x}) \in \mathbb{R}$ for each such query feature vector $\mathbf{x} \in \mathbb{R}^d$. The query image is then classified as belonging to the class with the greatest score.

$$\hat{y} = \arg\max_{y \in Y} s(\mathbf{x}) = \arg\max_{y \in Y} \frac{p^{+}(y|\mathbf{x})}{p^{-}(y|\mathbf{x})} \tag{3.2}$$

The predicted recovery maneuver $\hat{y}$ is then executed. $\hat{y}$ represents the recovery maneuver maximally likely to end in recovery and minimally likely to stay in failure. Ties are broken by looking at the $p^{+}(y|\mathbf{x})$ values, with higher values getting preference. If there is still a tie between a translational and rotational maneuver after this, the rotational maneuver is selected due to the translational maneuvers being comparatively more dangerous to execute as discussed in Section 3.1.

3.5 Experiments & Results

3.5.1 Handflying

We first validated our approach by handflying the quadrotor for over $3km$ in similar fashion to that used in data collection. Instead of executing only one maneuver during a test run however, we ran the failure response predictor as we walked. The predicted

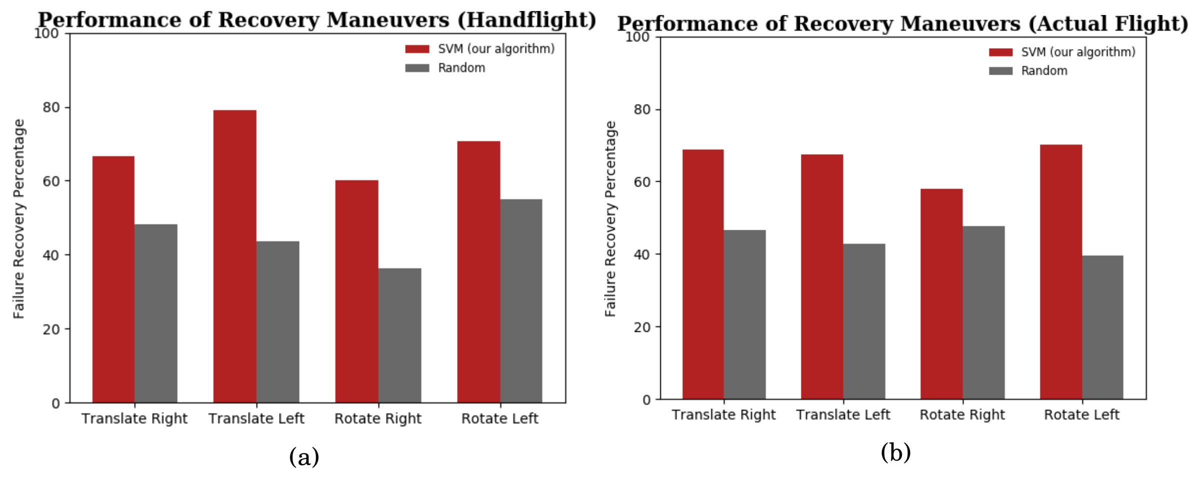


Figure 3.5: *Results obtained from the experiments. The graphs show the percentage of failures that ended up recovery for each maneuver during (a) Handflight, and (b) Actual flight.*

recovery maneuver was executed whenever a perception failure was encountered. Following this approach, 69% of failures ended in recovery, as compared to 45% when following a random maneuver as shown in Figure 3.5. The numerical breakdown of these failures and recoveries is given in Table 3.3.

3.5.2 Actual Flight

Finally, we carried out around 6km of tests in actual flight to test the robustness of this framework. Once a perception failure is predicted, the quadrotor stops following the previous trajectory it had received, and switches control over to the recovery maneuver to be executed. Upon completion of the maneuver, it resumes trajectory tracking. If however it did not recover, the maneuver is considered to have *failed*, and the quadrotor was commanded to land in-place. We do not consider this an intervention of autonomous flight. In our case, flight is intervened by a human if the quadrotor flew dangerously close to an obstacle (tree, branch, bush etc.).

When the recovery maneuver was predicted by our novel framework, the quadrotor recovered from failure 66% of the time. In contrast, it only recovered 43% of the time if the maneuver was chosen randomly. Figure 3.5 shows these results. In addition, with this framework in place, the quadrotor flew for over 1,200m on average through a dense, cluttered forest (roughly 2 trees per $4m \times 4m$ area) at 1.5 m/s before requiring human intervention. Again, the numerical breakdown of these failures and recoveries is given

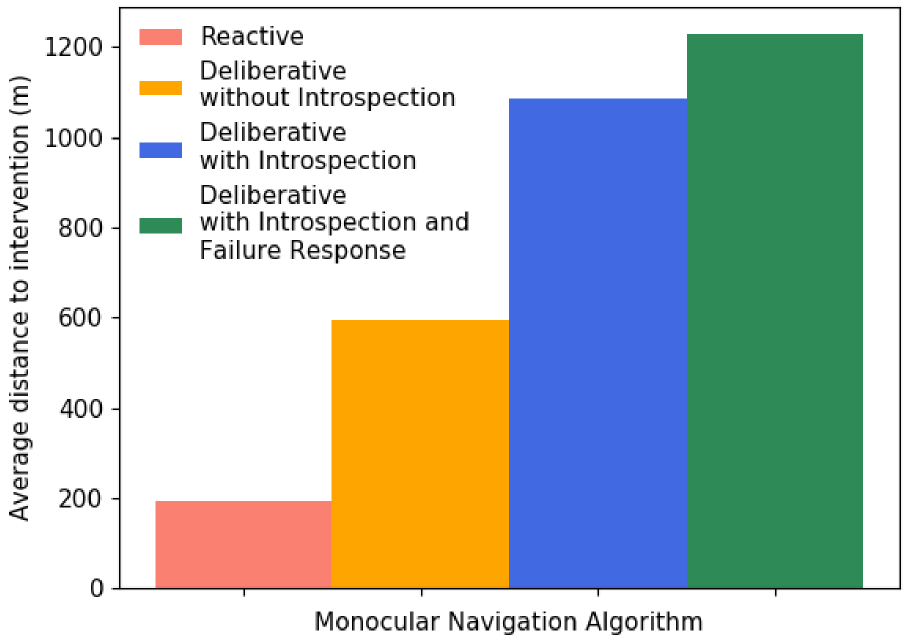


Figure 3.6: *Average flight distance without intervention in a dense, cluttered forest. The bars correspond to different algorithms from robotics literature - a purely reactive approach [34], a deliberative approach based on semi-dense monocular depth estimation [10], the same deliberative approach with introspection [9], and our framework which includes failure responses.*

in Table 3.3.

Figure 3.6 is a graph comparing various approaches that have been used for monocular flight through comparably dense forests. The average distance flown by the quadrotor is over 6x greater than a previous reactive controller based approach [34], and over 2x greater than a naive deliberative approach without introspection [10]. It is hard to compare our results with other existing research on autonomous outdoor flight [18, 4, 26]. The average distance flown without intervention, average flight speed, and density of obstacles in the environment are three metrics that are crucial for a fair comparison of attempts at autonomous outdoor flight. These works however either report only a subset of these metrics, or none of them.

3.6 Summary

This chapter presents both a comprehensive argument for learning failure responses, and a simple framework which achieves that goal. For collecting training data, we propose to

Experiment	Translate Right			Translate Left			Rotate Right			Rotate Left		
	Recovered	Not Recovered	% Recovery	Recovered	Not Recovered	% Recovery	Recovered	Not Recovered	% Recovery	Recovered	Not Recovered	% Recovery
Hand (Random)	14	15	48.28	10	13	43.48	8	14	36.36	11	9	55.00
Hand (SVM)	20	10	**66.67**	19	5	**79.17**	9	6	**60.00**	12	5	**70.59**
Flight (Random)	20	23	46.51	18	24	42.86	10	21	47.62	15	23	39.47
Flight (SVM)	31	14	**68.89**	29	14	**67.44**	22	16	**57.89**	28	12	**70.00**

Table 3.3: *Breakdown of failures encountered during experiments. The 'Hand' experiments refer to ones with handflight of the quadrotor, whereas the 'Flight' experiments refer to ones with actual flight of the quadrotor.*

handfly our quadrotor through our training and testing environment to save time and effort. Our experiments show that the data collected in this way does well at the intended task in-flight as well. We train two different SVMs to predict maneuvers most and least likely to lead to recovery, and use a combination of these scores to select a maneuver to execute at test time. This method achieves over $1,200m$ of uninterrupted flight on average, which is a $10-20\%$ improvement over the previous state-of-the-art developed as part of the same project. The older approach also utilized the deep introspection CNN, but did not have learned models of failure responses.

CHAPTER 4

Learning Corrective Maneuvers

Contents

In Chapter 3, we dealt with the problem of executing recovery maneuvers for images that are deemed to be causes of failure for our base monocular depth estimation algorithm. Some drawbacks of the approach we presented there are that,

a) it might be too late to execute a recovery maneuver after an image is deemed to have failed.

b) it is hard to collect data only for failure points because as the base perception algorithm improves, failure points are few and far between.

We address both these issues with the work in this chapter. We now seek to execute one of the same set of maneuvers as a *corrective* maneuver. The change in terminology is because of the fact that we no longer consider only failed images. We consider all images, and if there is a *corrective* maneuver that can help us improve overall performance, we will execute it. This way, we hope to avoid failures ahead of time, and not leave maneuver execution to the last second.

4.1 Data Collection

We follow the same data collection protocol as before, with details given in Section 3.2. Handflying the quadrotor lets us collect more data faster than we could if we collected data in flight. Moreover, since we are no longer restricted to executing maneuvers for images that have failed, we can execute many more maneuvers per handflight. During this process of handflying, we would walk a few steps with the quadrotor, stop, execute a maneuver, and continue until the next stop. The only consideration was that there should be a significant distance between two stops so that the images looked considerably different from each other.

The dataset for this work contains several frames along each maneuver executed, along with a measure of their quality (this will be discussed in Section 4.1.1). We do not categorise datapoints as being *good / beneficial* or *bad / detrimental* or *neutral* during this data collection process. As discussed previously in the introduction, eventually during test time, we want to execute maneuvers if and when there exists one that is *good / beneficial* in the sense that it will likely increase the quality of the image.

4.1.1 Quality Value (λ) of Images

For this work, we need a measure of the quality of monocular image frames. The output of introspection CNN described in Section 2.3 is $\hat{y} \in [0,1]$, where 1 corresponds to a *failure* image and 0 corresponds to a *good* image. For convenience, we use $1-\hat{y}$ as a quality measure for the frames in our dataset. We refer to this as the quality value λ. One thing to note here is that while the camera transmits frames at 15 fps, the CNN only outputs the λ at 2 Hz. Thus, the frames that go into our dataset are the ones that correspond to each output of the CNN. While this means we leave out a lot of frames (13 frames per second to be precise), it also means that we cannot go through the dataset augmentation step from Section 3.2.

During data collection, we handflew the quadrotor for about $20km$, the same amount as the work in Chapter 3. However, in contrast to the 825 failures we encountered, we were able to execute and collect data from over 2,400 corrective maneuvers. The maneuvers themselves are the same as before - rotate left/right and translate left/right. Our final dataset contains just over 17,000 images and their λ values. Table 4.1 shows the number of frames collected during the execution of each of the four different maneuvers.

While analyzing this data, we found as expected that the λ values for each maneuver nicely fit a normal distribution. These values are provided in the last column of the

Maneuver	**Images**	**λ Distribution**
Translate Right	4,539	0.681 ± 0.030
Translate Left	4,038	0.672 ± 0.031
Rotate Right	4,110	0.676 ± 0.033
Rotate Left	4,347	0.674 ± 0.030
Total	**17,034**	

Table 4.1: *Number of images for each maneuver in the training set, along with the mean and standard deviation values for λ.*

table above. This normal distribution of λ represents the fact that most images that the monocular camera might come across in flight are *good enough* in quality, with only a few that are better or worse. This was expected because we are well aware of the causes of failure of our base vision algorithm. Consequently, we are also aware of perfect images for our algorithm. From the numerous flights we have conducted, it was to be expected that our `(image, λ)` data would obey a normal distribution. Figure 4.1 shows histograms of λ and $\Delta\lambda$ for the datasets for all maneuvers, along with the fitted normal distributions (in red). We will discuss what we mean by (and how we use) $\Delta\lambda$ later in Section 4.2, but in short, $\Delta\lambda$ for a frame in the dataset is the difference between the quality values λ of the final frame in a maneuver and that particular frame.

4.1.2 Dataset Augmentation

While we cannot use the dataset augmentation trick from Section 3.2, we can still increase the size of our dataset in a much more sensible and logical way. Since each image in our dataset now has an associated λ, and because all our corrective maneuvers are mirrored by another (rotate/translate left $\leftrightarrow$ rotate/translate right), we can reverse the sequence of images and λ from a *left* dataset, and add it to the corresponding *right* dataset and vice versa. This way, each classifier is trained on 8,000+ images instead of 4,000+ according to Table 4.1. Figure 4.2 shows an example of this augmentation. For every right (left) maneuver we execute, we are able to add a maneuver to the corresponding left (right) dataset. In addition, the figure provides a sense of our normally distributed datasets. Most of the maneuvers we execute result in λ trajectories like the one in Figure 4.2(a). There are some that significantly increase or decrease the λ of the images, as shown in Figure 4.2(b), and we want to put more emphasis on these during classifier training. This is because at test time, we would like to avoid the ambiguity of a *neither* good or bad prediction from the classifiers.

4.2 Classifier Training

Since we treat our problem as a classification problem, we obviously need to divide our dataset into classes. As we alluded to before, we divide the dataset into three classes. Let ϕ represent a corrective maneuver. For every image in the ϕ dataset, ϕ could be

- *Good/beneficial*: this is the case if executing ϕ from that image increase λ significantly.
- *Bad/detrimental*: this is the case if executing ϕ from that image decreased λ significantly.
- *Neutral*: if ϕ does not cause a significant change in the λ after execution, we say its effect on the image is neutral.

The change in λ of an image is considered with respect to the last image frame and λ we receive during the execution of a corrective maneuver. Let x respresent the `fc7` feature vector of images in our dataset. Data collected from the execution of a single maneuver is $\phi = \{(x_1, \lambda_1), (x_2, \lambda_2), \cdots, (x_i, \lambda_i), \cdots, (x_N, \lambda_N)\}$. Then we define,

$$\Delta\lambda_i = \lambda_N - \lambda_i, \forall i \leq N \tag{4.1}$$

The normal distributions of $\Delta\lambda$ for all maneuvers can be seen in Figure 4.1(b). For the purpose of training naive SVM classifiers, an image x_i is assigned a class y_i on the following basis,

$$y_i = \begin{cases} 1, & \Delta\lambda_i^{\phi} > \mu^{\Delta\lambda,\phi} + \frac{\sigma^{\Delta\lambda,\phi}}{2} \\ 0, & \mu^{\Delta\lambda,\phi} - \frac{\sigma^{\Delta\lambda,\phi}}{2} \leq \Delta\lambda_i^{\phi} \leq \mu^{\Delta\lambda,\phi} + \frac{\sigma^{\Delta\lambda,\phi}}{2} \\ -1, & \Delta\lambda_i^{\phi} < \mu^{\Delta\lambda,\phi} - \frac{\sigma^{\Delta\lambda,\phi}}{2} \end{cases} \tag{4.2}$$

where $\mu^{\Delta\lambda,\phi}$ and $\sigma^{\Delta\lambda,\phi}$ are the mean and standard deviation respectively of $\Delta\lambda$ for maneuver ϕ. $y_i = 1$ means ϕ is *good* for x_i; $y_i = -1$ means ϕ is *bad* for x_i; and $y_i = 0$ means ϕ is *neutral* for x_i. Obviously, $\Delta\lambda_N = \lambda_N - \lambda_N = 0$, which is not helpful. This means that the final frame in a trajectory cannot be assigned a class on the basis of its $\Delta\lambda$ value. Instead, for the final frame, we follow the same logic as in Equation 4.2, except we use the λ of the frame and the λ statistics of the dataset for maneuver ϕ.

Maneuver	**Images**		
	Good	Neutral	Bad
Translate Right	2,299	3,961	2,317
Translate Left	2,325	3,945	2,307
Rotate Right	2,276	4,009	2,178
Rotate Left	2,243	4,028	2,186

Table 4.2: *Number of images of each class in the dataset for corrective maneuvers.*

A naive SVM classifier is then trained on the (x_i, y_i) datapoints thus created for each maneuver ϕ. In our case, we ultimately end up training 4 different classifiers - one for each of the 4 maneuvers. At test time, we can use these classifiers to predict if a particular corrective maneuver will be good or not for an image frame. We execute a corrective maneuver at random from the set of maneuvers predicted to be *good* (which could easily be the empty set as well).

4.2.1 Cost-Sensitive Classification

The downside of the naive classifiers stems from the fact that our data is normally distributed. This means that when we split it into the three different classes according to Equation 4.2, there is a severe class imbalance problem, seen in Table 4.2. Around 50% of the images for each maneuver are labelled as *neutral*. This results in the predictions at test time being dominated by this class, which is unhelpful to us when we want to make a decision to execute a maneuver. There end up being a number of instances of frames that would likely benefit from a corrective maneuver, but our classifiers miss these by classifying all maneuvers as *neutral*. In literature, two common ways to deal with the class imbalance problem are to oversample or undersample the minority or dominant class(es); or introduce some form of regularisation in the training procedure [27]. Cost-sensitive classification effectively acts as a regularisation scheme that decreases the number of false negatives at the expense of allowing more false positives. What this means in our scheme is that we might execute unnecessary maneuvers from time to time, but we will be less likely to miss the opportunity to execute a maneuver when necessary.

Consider a two-class classification problem. The extension to more than two classes is trivial. You can train multiple `one-vs-one` or `one-vs-rest` classifiers and make a prediction by aggregating their individual predictions [22]. We use the `one-vs-one` classification technique for this work. Given training datapoints $x_i \in \mathbb{R}^d, i = 1, \cdots, N$ and their corresponding class labels $y_i \in \{-1, 1\}^N$, the naive SVM solves the following optimisation problem:

$$
\begin{aligned}
&\min_{\mathbf{w},b,\zeta} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\zeta_i \\
&\qquad \text{subject to} \quad y_i\left(\mathbf{w}^T\kappa(x_i) + b\right) \geq 1 - \zeta_i, \\
&\qquad\qquad\qquad\qquad \zeta_i \geq 0, \quad i = 1, \cdots N
\end{aligned}
\tag{4.3}
$$

with sign $\left(\mathbf{w}^T\kappa(x_i) + b\right)$ as the decision function. $\kappa(x_i)$ represents some feature transformation of the input data. In our case, input data is an image, and the result of $\kappa(x_i)$ is the `fc7` feature vector we use. In Equation 4.3, C acts as a regularization parameter and ζ_i are slack variables. These help account for overlapping classes during the optimisation procedure. What the equation does not capture is the class imbalance problem we referred to above.

By incorporating datapoint-dependent costs in the optimisation procedure instead of a common C variable, we can account for the 'importance' of a particular datapoint, and solve a cost-sensitive optimisation problem to train our SVMs [43, 28]. In our case, importance is related to the $\Delta\lambda$ of a datapoint. The cost-sensitive optimisation problem we solve instead of Equation 4.3 is:

$$
\begin{aligned}
&\min_{\mathbf{w},b,\zeta} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\left(\sum_{i=1}^{N} s_i\zeta_i\right) \\
&\qquad \text{subject to} \qquad y_i\left(\mathbf{w}^T\kappa(x_i) + b\right) \geq 1 - \zeta_i, \\
&\qquad\qquad\qquad\qquad \zeta_i \geq 0, \quad i = 1, \cdots N \\
&\qquad \text{where} \qquad s_i = \max\left(4\left(\frac{\Delta\lambda_i - \mu^{\Delta\lambda,\phi}}{\sigma^{\Delta\lambda,\phi}}\right)^2, 1\right)
\end{aligned}
\tag{4.4}
$$

As before, since $\Delta\lambda_N = 0$ for the last frame along an executed maneuver, the datapoint-dependent costs for these frames is calculated using their λ and corresponding λ statistics.

The effect of such a cost-sensitive classification can be seen in Figure 4.3. We create a toy problem to imitate some of the characteristics of the actual problem we are trying to solve. We generate points in $\mathbb{R}^2$ uniformly at random, and assign a colour value to them on a continuous gradient from blue (0) to red (1). This is analogous to our feature vectors in $\mathbb{R}^d$ having a λ (or $\Delta\lambda$) associated with them. The task is to classify points as blue or red or neither (analogous to maneuvers being good or bad or neutral). We split the input data in the same way as Equation 4.2, using the colour values and its statistics. The plot on the left shows the input data. The plot in the middle shows a naive SVM classification and decision boundaries. Note that neutral points are coloured as almost white. The datapoint-dependent weights are assigned using Equation 4.4, and the plot on the right shows the cost-sensitive SVM classification and decision boundaries. Obviously, the effect of the datapoint-dependent weighting is to enlarge the boundaries surrounding more blue or more red datapoints. But the important takeaway is that cost-sensitive classification is able to find the 'more blue' or 'more red' regions in a dataset where the different classes overlap considerably. While cost-sensitive classification may not offer a big advantage for less data as in Figure 4.3(a), it does a much better job with more data which can be seen in Figure 4.3(b).

The SVMs in this toy example are trained using all the same parameters as in our actual experiments. The only difference is the dimensionality of the input data, the distribution it is drawn from, and the distribution of the colour values.

4.3 Experiments & Results

As before, we evaluated our approach by first handflying the quadrotor and then in actual flight. We conducted both sets of experiments over roughly $4km$. During one run of an experiment, we query all four SVMs corresponding to the four maneuvers for every image frame that we get from the front-facing monocular camera. For both naive and cost-sensitive cases, if any of the SVMs make a *good* prediction, we execute a maneuver. If there is only one SVM that makes a *good* prediction, we execute that maneuver. If there are multiple, we pick one at random and execute it. $\Delta\lambda_1$ indicates the effectiveness of the maneuver. The detailed results from these experiments are given in Table 4.3. We did not run any experiments where we executed a maneuver at random since that makes less sense in this paradigm. Should we execute a maneuver at random for every frame? Should we execute a maneuver at random every k frames/seconds/meters? Executing maneuvers at random is not a fair alternative in this case to using SVM predictions to make a decision.

The graphs below display the results from Table 4.3. Figure 4.4 shows the $\Delta\lambda$ error bar plots we observed after executing a corrective maneuver in our experiments. Note that only one 'Rotate Left' maneuver was executed while using the naive 3-class SVMs during handflight. It can be seen that the variance for cost-sensitive classifiers tends to decrease as more maneuvers are executed. In comparison, the naive SVM predictions

Experiment	Translate Right		Translate Left		Rotate Right		Rotate Left	
	# Executed	$\Delta\lambda$	# Executed	$\Delta\lambda$	# Executed	$\Delta\lambda$	# Executed	$\Delta\lambda$
Hand (Naive SVM)	7	-0.0023 ± 0.0319	8	0.0018 ± 0.0262	9	$\mathbf{0.0010 \pm 0.0185}$	1	-0.0007
Hand (C-S SVM)	20	$\mathbf{0.0116 \pm 0.0023}$	19	$\mathbf{0.0089 \pm 0.0097}$	15	-0.0002 ± 0.0277	12	$\mathbf{0.0250 \pm 0.0086}$
Flight (Naive SVM)	10	0.0015 ± 0.0271	8	0.0002 ± 0.0238	5	0.0010 ± 0.0304	14	0.0187 ± 0.0106
Flight (C-S SVM)	9	$\mathbf{0.0102 \pm 0.0113}$	11	$\mathbf{0.0158 \pm 0.0094}$	23	$\mathbf{0.0269 \pm 0.0145}$	16	$\mathbf{0.0191 \pm 0.0037}$

Experiment	Translate Right				Translate Left			
	# Failed	Recovered	Not Recovered	% Recovery	# Failed	Recovered	Not Recovered	% Recovery
Hand (Naive SVM)	2	1	1	50.00	5	3	2	60.00
Hand (C-S SVM)	6	5	1	83.33	2	2	0	100.00
Flight (Naive SVM)	9	2	7	22.22	3	1	2	33.33
Flight (C-S SVM)	1	0	1	0.00	4	3	1	75.00

Experiment	Rotate Right				Rotate Left			
	# Failed	Recovered	Not Recovered	% Recovery	# Failed	Recovered	Not Recovered	% Recovery
Hand (Naive SVM)	3	2	1	66.67	0	N/A	N/A	N/A
Hand (C-S SVM)	8	5	3	62.50	4	3	1	75.00
Flight (Naive SVM)	4	0	4	0.00	7	4	3	57.14
Flight (C-S SVM)	7	6	1	85.71	13	10	3	76.92

Table 4.3: *The ‘Hand’ experiments refer to ones with handflight of the quadrotor, whereas the ‘Flight’ experiments refer to ones with actual flight of the quadrotor. ‘C-S SVM’ is the cost-sensitive SVM. The table at the top shows the effect on image λ due to corrective maneuver execution. The bottom two tables look at the performance of these continuous case classifiers in the discrete case paradigm.*

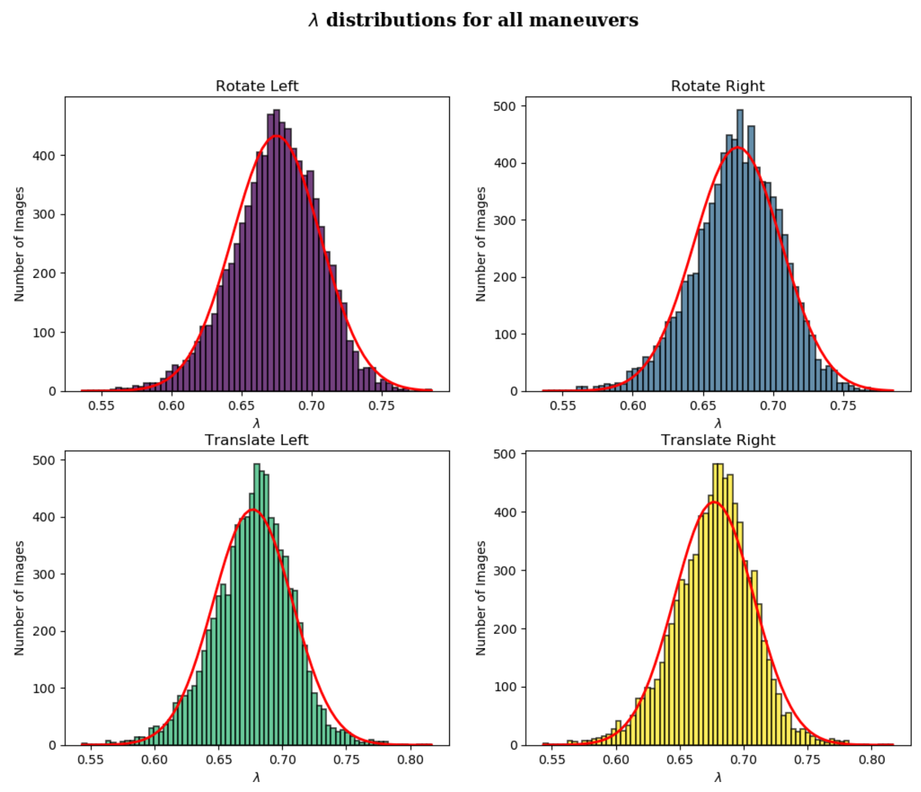
vary much more in terms of the effectiveness of the maneuver.

There are a few conclusions we can draw from these experiments. The first is that using a cost-sensitive SVM is much more effective in terms of improving image quality on average, even though the variance between the naive and cost-sensitive SVMs is comparable. Secondly, naive SVMs predict far fewer *good* maneuvers during handflight or actual flight. This was to be expected since these classifiers are dominated by the *neutral* class from our training data. In terms of maneuvers executed for images that would have been flagged as failures under the paradigm from Chapter 3, cost-sensitive classifiers experience significantly less failure images than the SVMs from the previous approach. We believe that this is because these classifiers achieve the intended goal of avoiding failures pre-emptively. Even for the failure images that they do predict maneuvers for, the recovery percentages are comparable to those from Table 3.3. All things considered, cost-sensitive SVM classifiers retain the positive aspect of the approach from the previous chapter in that they are able to recover from failures, but in addition, they improve the image quality after executing a corrective maneuver, and also experience far fewer failure cases.

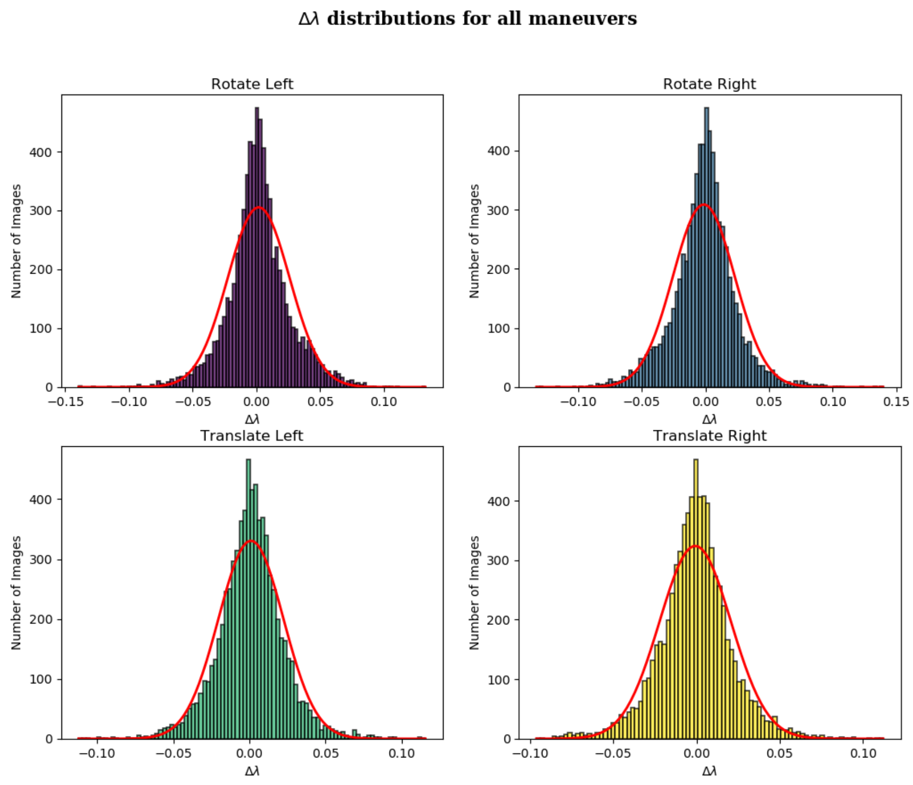
In terms of average distance flown without human intervention, these classifiers outperform the ones from Chapter 3 by 7%. Over roughly $4km$ (this distance is measured using a combination of the quadrotor's on-board visual odometry algorithm and lidar sensor), the previous approach flew for $1,232m$ on average without intervention. By using cost-sensitive classifiers, we flew for $1,318m$ on average without intervention. It is interesting to note that we only flew for $948m$ on average without intervention when we relied on the naive SVM classifiers from this chapter. Again, the reason for such comparatively poor performance is the fact that the naive SVMs predict *neutral* maneuvers most of the time, which is akin to flying without introspection. Figure 4.5 shows these numbers in a bar graph. It is an extension of Figure 3.6 with additional bars for the two approaches discussed in this chapter for comparison.

4.4 Summary

This chapter presents a continuous case extension of Chapter 3. The concept of failed images is done away with in favour of a quality measure of input images. By doing this, we can reason in terms of trying to improve image quality by executing corrective maneuvers. We collect data by handflying the quadrotor and executing a corrective maneuver uniformly at random. We observe that this data is normally distributed in terms of the maneuver's effect on the image quality. Some make it worse, some improve it, but most do not affect it much. This leads us to a cost-sensitive classification framework where we use datapoint-dependent costs to put more emphasis on images for which a particular maneuver was significantly good or bad. We evaluate these classifiers against naive ones that use equal costs for all datapoints in handflight and actual flight.

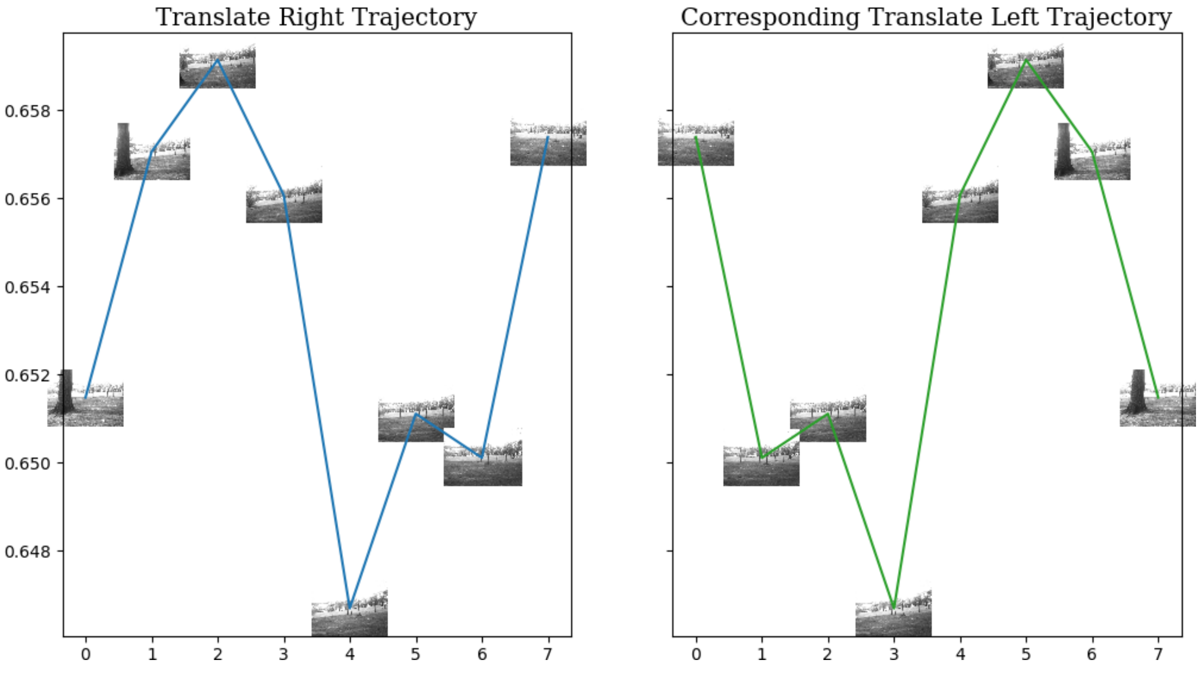


(a)

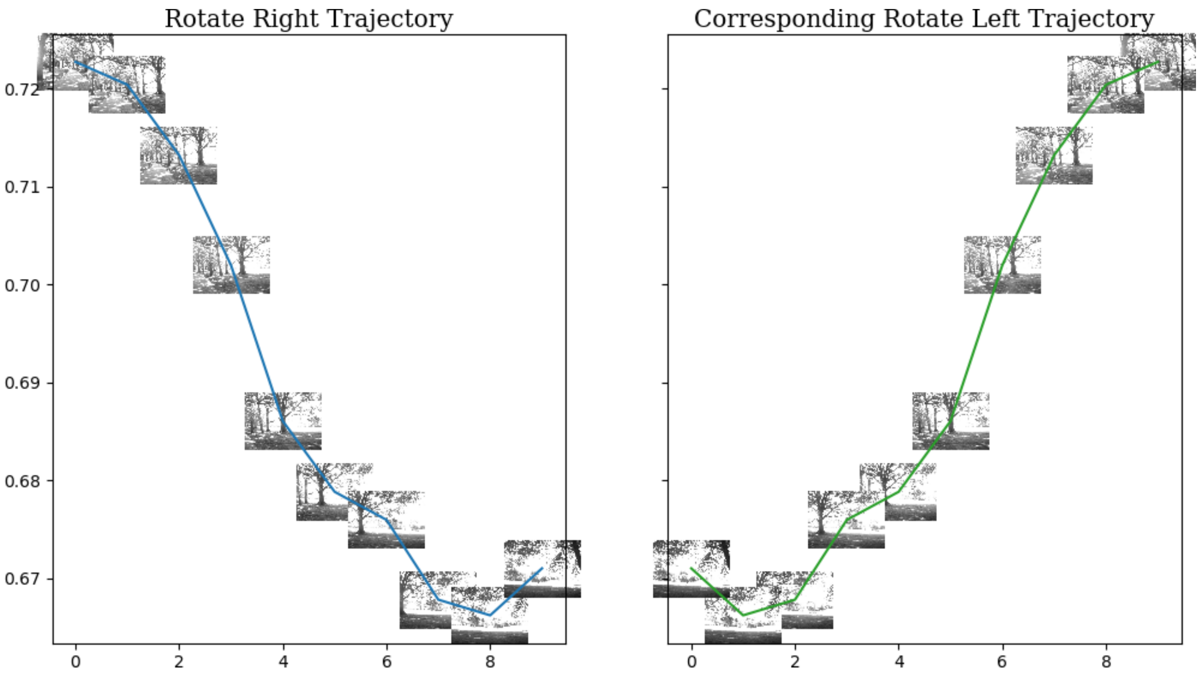


(b)

Figure 4.1: *(a) λ histograms for all maneuvers. The fitted normal distribution curve is in red. (b) $\Delta\lambda$ histograms for all maneuvers. The fitted normal distribution curve is in red.*

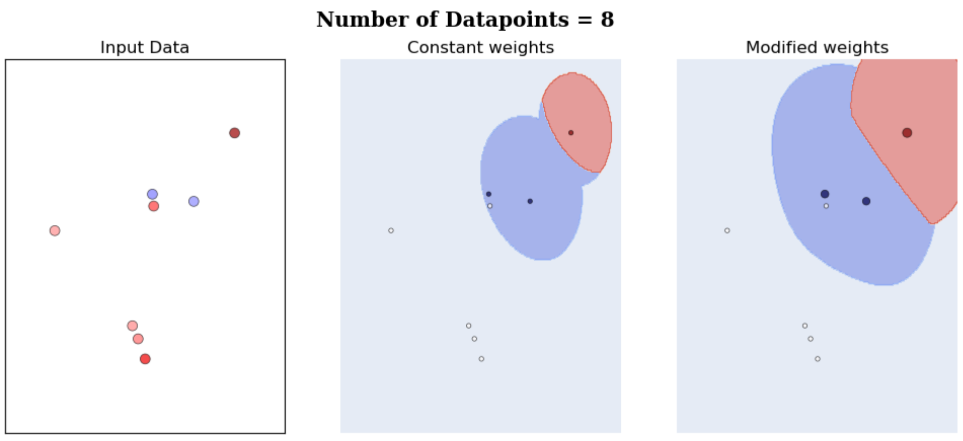


(a)

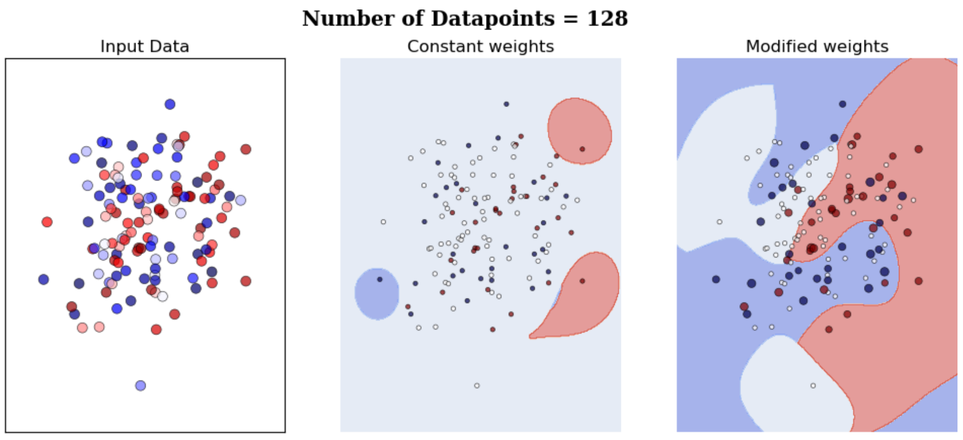


(b)

Figure 4.2: *We are able to increase the size of our dataset by flipping a right (left) maneuver, and adding it to the corresponding left (right) dataset. While most trajectories (a) do not change the λ much, we put more emphasis on (b) the ones that do.*

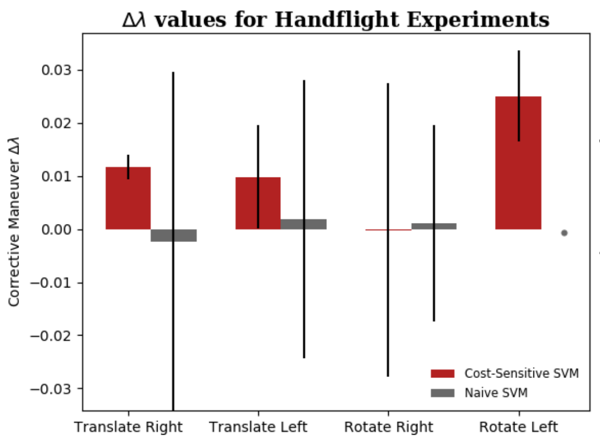


(a)

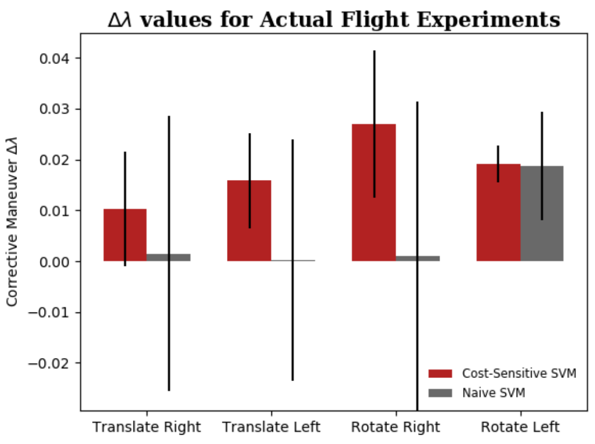


(b)

Figure 4.3: *While trying to classify points in $\mathbb{R}^2$ as blue or red, where the classes overlap, cost-sensitive classification (a) does not offer a big advantage with less data. However, with more data available, (b) it can grow out regions of both classes by using the importance weights of each datapoint.*



(a)



(b)

Figure 4.4: *The graphs show the mean ± one standard deviation of $\Delta\lambda$ of executed corrective maneuvers that was observed during (a) Handflight, and (b) Actual flight.*

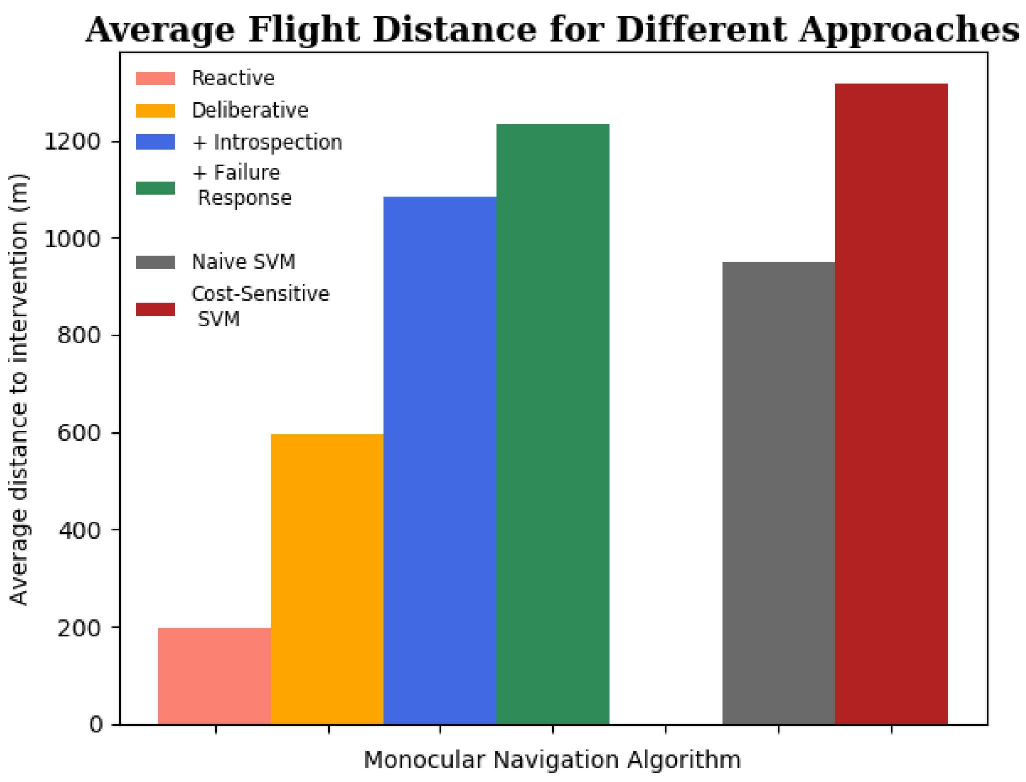


Figure 4.5: *The majority of this graph is the same as Figure 3.6. In addition, we plot the bars for average distance flown without intervention using the naive 3-class and cost-sensitive SVM approaches discussed in this chapter.*

CHAPTER 5

Conclusions & Future Work

Contents

In this thesis, we explore the problem of learning to execute corrective maneuvers to improve the quality of the input given to a base vision algorithm. In our case, the base algorithm is monocular depth estimation using semi-dense visual odometry. We use an introspective measure to get a quality measure for images that this algorithm uses. Given these two pieces, to try to learn to associate a small set of corrective maneuvers with images for which they are likely to increase quality.

5.1 Conclusions

There is no doubt that introspection is an important tool for mobile robots especially. There is a lot of value in being able to quantify the quality of inputs supplied to other pieces of a mobile robot's pipeline downstream. It can help deal with compounding errors at an early stage. However, it is important to come up with a good performance metric while training such a system since this is what determines the efficacy of a trained introspection system. The performance metric is an engineering decision that needs to be made depending on the robot and task at hand.

In a similar vein, the set of recovery or corrective maneuvers is also an engineering decision where the mobile robot and domain of operation play an important role. The approaches discussed in this thesis are able to increase flight performance by $10-20\%$. It might be possible to obtain greater performance boosts by improving the base algorithms. But because these algorithms can never be perfect, it is extremely important to give robots the capabilities explored in this thesis. We present a comprehensive argument for learning failure responses and also avoiding failures altogether. In addition, we provide a framework which achieves that goal starting from data collection to training to implementation on a real platform.

Cost-sensitive classifiers are a popular way to deal with problems with an inherent class imbalance. By tuning the datapoint-dependent costs to suit our dataset, we can find regions in `fc7` feature space corresponding to beneficial maneuvers. Without using cost-sensitive classifiers, it is impossible to do so since the data that we observe is heavily dominated by maneuvers that have a negligible effect on image quality. This approach should be applicable to any mobile robot platform equipped with an introspection pipeline. Using these two, we can collect data that increases or decreases a quality measure over the inputs used by the robot. From this dataset, we can weight samples according to their effect on this quality measure and train cost-sensitive classifiers to learn the desired behaviour. For example, a mobile robot running a monocular SLAM algorithm could use its localisation uncertainty as a quality measure. Feature-rich images are likely to decrease this uncertainty, and we can thus enable the robot to execute maneuvers that make it move towards regions it expects to be feature-rich.

5.2 Future Work

The work that we explore in this thesis makes no use of past information, i.e. the history of images received by the camera prior to executing a maneuver. By incorporating temporal information, or using a local map of the surrounding region, a robot could make better decisions. For instance, in the current framework, if a robot just moves past a tree to its left and faces a need to execute a maneuver, there is nothing stopping it from executing the translate left maneuver and crashing right back into the tree. This problem has been a part of our monocular navigation research for a long time [34].

Similarly, during training of the classifiers in this thesis, we treate the separate frames along one executed maneuver as separate datapoints. If we can learn a time-series dynamical model [41] of the system in λ trajectory space, we could try and predict the evolution of the system state for each candidate maneuver and use these multi-step predictions to make a more informed decision. This is related to the idea of training regressors instead of classifiers. The goal here would be to predict future λ trajectories given the current image, similar to those shown in Figure 4.2.

Following the discussion in Section 3.2, we can only execute one maneuver per image and observe the result of executing that maneuver only. This means that in our dataset, we really only have information about a single maneuver for an image. Such a situation is captured better as a contextual bandit problem. The context in our case would be the image, with the different maneuvers being the bandits. We then seek to learn reward function estimates for each maneuver online. There are two major problems with such an approach. Online learning algorithms are under-explored in high-dimensional spaces (our images are 640×480 and the `fc7` feature vectors have $4,096$ elements). In addition, online learning algorithms can be slow to converge. This would mean we would essentially be executing maneuvers at random in the beginning which can be hazardous since we operate specifically with unreliable inputs. However, using such an online learning framework along might allow us to dynamically edit the set of recovery maneuvers (add/delete) based on the estimates of the mean rewards of maneuvers and perhaps even the uncertainty around these estimates.

Bibliography

[1] A. Agarwal, S. M. Kakade, N. Karampatziakis, L. Song, and G. Valiant, *Least squares revisited: Scalable approaches for multi-class prediction*, in Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32, ICML'14, JMLR.org, 2014, pp. II–541–II–549.
Section 2.5.

[2] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, *On the surprising behavior of distance metrics in high dimensional space*, in International Conference on Database Theory, Springer, 2001, pp. 420–434.
Section 3.2.

[3] S. Arora, S. Choudhury, D. Althoff, and S. Scherer, *Emergency maneuver library-ensuring safe navigation in partially known environments*, in Robotics and Automation (ICRA), 2015 IEEE International Conference on, IEEE, 2015, pp. 6431–6438.
Section 1.2.

[4] A. J. Barry and R. Tedrake, *Pushbroom stereo for high-speed navigation in cluttered environments*, in Robotics and Automation (ICRA), 2015 IEEE International Conference on, IEEE, 2015, pp. 3046–3052.
Section 3.5.2.

[5] M. Blösch, S. Weiss, D. Scaramuzza, and R. Siegwart, *Vision based mav navigation in unknown and unstructured environments*, in Robotics and automation (ICRA), 2010 IEEE international conference on, IEEE, 2010, pp. 21–28.
Section 1.1.

[6] S. Choudhury, S. Scherer, and J. A. Bagnell, *Theoretical limits of speed and resolution for kinodynamic planning in a poisson forest*, in Robotics: Science and Systems XI, Sapienza University of Rome, Rome, Italy, July 13-17, 2015, 2015.
Section 1.2.

[7] A. Coates, P. Abbeel, and A. Y. Ng, *Learning for control from multiple demonstrations*, in Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 144–151.
Section 1.1.

[8] S. Daftry, *Towards scalable visual navigation of micro aerial vehicles*, Master's thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, April 2016.
Sections 2.3 and 2.5.

[9] S. Daftry, S. Zeng, J. A. Bagnell, and M. Hebert, *Introspective perception: Learning to predict failures in vision systems*, in Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on, IEEE, 2016, pp. 1743–1750.
Sections 1.4, 2.3, 2.5, and 3.6.

[10] S. Daftry, S. Zeng, A. Khan, D. Dey, N. Melik-Barkhudarov, J. A. Bagnell, and M. Hebert, *Robust monocular flight in cluttered outdoor environments*, arXiv preprint arXiv:1604.04779, (2016).
Sections 2.5, 3.6, and 3.5.2.

[11] D. Dey, K. S. Shankar, S. Zeng, R. Mehta, M. T. Agcayazi, C. Eriksen, S. Daftry, M. Hebert, and J. A. Bagnell, *Vision and learning for deliberative monocular cluttered flight*, in Field and Service Robotics, Springer, 2016, pp. 391–409.
Section 2.5.

[12] P. Domingos, *A few useful things to know about machine learning*, Commun. ACM, 55 (2012), pp. 78–87.
Section 3.2.

[13] J. Dong, E. Nelson, V. Indelman, N. Michael, and F. Dellaert, *Distributed real-time cooperative localization and mapping using an uncertainty-aware expectation maximization approach*, in Robotics and Automation (ICRA), 2015 IEEE International Conference on, IEEE, 2015, pp. 5807–5814.
Section 1.1.

[14] J. Engel, T. Schöps, and D. Cremers, *Lsd-slam: Large-scale direct monocular slam*, in European Conference on Computer Vision, Springer, 2014, pp. 834–849.
Section 2.2.

[15] J. Engel, J. Sturm, and D. Cremers, *Camera-based navigation of a low-cost quadrocopter*, in Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, IEEE, 2012, pp. 2815–2821.
Section 1.1.

[16] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, *Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle*, Journal of Field Robotics, 33 (2016), pp. 431–450.
Section 1.1.

[17] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys, *Vision-based autonomous mapping and exploration using a quadrotor mav*, in Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, IEEE, 2012, pp. 4557–4564.
Section 1.1.

[18] A. Giusti, J. Guzzi, D. C. Cireşan, F. L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella, *A machine learning approach to visual perception of forest trails for mobile robots*, IEEE Robotics and Automation Letters, 1 (2016), pp. 661–667.
Sections 1.1 and 3.5.2.

[19] C. Green and A. Kelly , *Optimal sampling in the space of paths: Preliminary results*, Tech. Rep. CMU-RI-TR-06-51, Robotics Institute, Pittsburgh, PA, November 2006.
Section 2.4.

[20] H. Grimmett, R. Paul, R. Triebel, and I. Posner, *Knowing when we don't know: Introspective classification for mission-critical decision making*, in 2013 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2013, pp. 4531–4538.
Section 2.3.

[21] H. Grimmett, R. Triebel, R. Paul, and I. Posner, *Introspective classification for robot perception*, The International Journal of Robotics Research, (2015), p. 0278364915587924.
Section 2.3.

[22] C.-W. Hsu and C.-J. Lin, *A comparison of methods for multiclass support vector machines*, IEEE transactions on Neural Networks, 13 (2002), pp. 415–425.
Section 4.2.1.

[23] S. Karaman and E. Frazzoli, *High-speed flight in an ergodic forest*, CoRR, abs/1202.0253 (2012).
Section 1.2.

[24] H. Kohut, *Introspection, empathy, and psychoanalysis: An examination of the relationship between mode of observation and theory.*, Journal of the American Psychoanalytic Association, (1959).
Section 2.3.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, in Advances in neural information processing systems, 2012, pp. 1097–1105.
Section 2.3.

[26] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, *Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments*, IEEE Robotics and Automation Letters, PP (2017), pp. 1–1.
Section 3.5.2.

[27] R. Longadge and S. Dongre, *Class imbalance problem in data mining review*, arXiv preprint arXiv:1305.1707, (2013).
Section 4.2.1.

[28] H. Masnadi-Shirazi, N. Vasconcelos, and A. Iranmehr, *Cost-sensitive support vector machines*, CoRR, abs/1212.0975 (2012).
Section 4.2.1.

[29] D. Mellinger and V. Kumar, *Minimum snap trajectory generation and control for quadrotors*, in Robotics and Automation (ICRA), 2011 IEEE International Conference on, IEEE, 2011, pp. 2520–2525.
Section 1.1.

[30] A. C. Morris, *Robotic introspection for exploration and mapping of subterranean environments*, ProQuest, 2007.
Sections 2.3 and 1.

[31] J. C. PLATT, *Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods*, in Advances in Large Margin Classifiers, MIT Press, 1999, pp. 61–74.
Section 3.4.

[32] J. A. PREISS, W. HÖNIG, G. S. SUKHATME, AND N. AYANIAN, *Crazyswarm: A large nano-quadcopter swarm*, in Proc. IEEE International Conference on Robotics and Automation, 2017.
Section 1.1.

[33] C. RICHTER, A. BRY, AND N. ROY, *Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments*, Springer International Publishing, Cham, 2016, pp. 649–666.
Section 1.1.

[34] S. ROSS, N. MELIK-BARKHUDAROV, K. S. SHANKAR, A. WENDEL, D. DEY, J. A. BAGNELL, AND M. HEBERT, *Learning monocular reactive uav control in cluttered natural environments*, in Robotics and Automation (ICRA), 2013 IEEE International Conference on, IEEE, 2013, pp. 1765–1772.
Sections 1.1, 2.5, 3.6, 3.5.2, and 5.2.

[35] S. SCHERER, S. SINGH, L. CHAMBERLAIN, AND S. SARIPALLI, *Flying fast and low among obstacles*, in Robotics and Automation, 2007 IEEE International Conference on, IEEE, 2007, pp. 2023–2029.
Section 1.1.

[36] T. SCHOUWENAARS, J. HOW, AND E. FERON, *Receding horizon path planning with implicit safety guarantees*, in American Control Conference, 2004. Proceedings of the 2004, vol. 6, IEEE, 2004, pp. 5576–5581.
Section 1.2.

[37] K. SIMONYAN AND A. ZISSERMAN, *Two-stream convolutional networks for action recognition in videos*, in Advances in neural information processing systems, 2014, pp. 568–576.
Section 2.3.

[38] Y. TANG, *Deep learning using support vector machines*, CoRR, abs/1306.0239 (2013).
Section 3.3.

[39] R. Triebel, H. Grimmett, R. Paul, and I. Posner, *Driven learning for driving: How introspection improves semantic mapping*, in The 16th International Symposium of Robotics Research (ISRR), Springer, 2016, pp. 449–465.
Section 2.3.

[40] M. Turpin, K. Mohta, N. Michael, and V. Kumar, *Goal assignment and trajectory planning for large teams of interchangeable robots*, Autonomous Robots, 37 (2014), pp. 401–415.
Section 1.1.

[41] A. Venkatraman, M. Hebert, and J. A. Bagnell, *Improving multi-step prediction of learned time series models.*, in AAAI, 2015, pp. 3024–3030.
Section 5.2.

[42] M. Watterson and V. Kumar, *Safe receding horizon control for aggressive mav flight with limited range sensing*, in Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on, IEEE, 2015, pp. 3235–3240.
Section 1.2.

[43] B. Zadrozny and C. Elkan, *Learning and making decisions when costs and probabilities are both unknown*, in Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2001, pp. 204–213.
Section 4.2.1.