

Learning Visual Odometry Primitives for Computationally Constrained Platforms

by

Tori Wuthrich

Submitted to the Department of Aeronautical and Astronautical
Engineering

in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2019

© Massachusetts Institute of Technology 2019. All rights reserved.

Author

Department of Aeronautical and Astronautical Engineering

May 23, 2019

Certified by

Gian Luca Mariottini

Draper Supervisor

Thesis Supervisor

Certified by

Sertac Karaman

Associate Professor of Aeronautics and Astronautics

Thesis Supervisor

Accepted by

Sertac Karaman

Associate Professor of Aeronautics and Astronautics

Chairman, Department Committee on Graduate Theses

Learning Visual Odometry Primitives for Computationally Constrained Platforms

by

Tori Wuthrich

Submitted to the Department of Aeronautical and Astronautical Engineering
on May 23, 2019, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Autonomous navigation for robotic platforms, particularly techniques that leverage an onboard camera, are of currently of significant interest to the robotics community. Designing methods to localize small, resource-constrained robots is a particular challenge due to limited availability of computing power and physical space for sensors. A computer vision, machine learning-based localization method was proposed by researchers investigating the automation of medical procedures. However, we believed the method to also be promising for low size, weight, and power (SWAP) budget robots. Unlike for traditional odometry methods, in this case, a machine learning model can be trained offline, and can then generate odometry measurements quickly and efficiently. This thesis describes the implementation of the learning-based, visual odometry method in the context of autonomous drones. We refer to the method as RetiNav due to its similarities with the way the human eye processes light signals from its surroundings. We make several modifications to the method relative to the initial design based on a detailed parameter study, and we test the method on a variety of challenging flight datasets. We show that over the course of a trajectory, RetiNav achieves as low as 1.4% error in predicting the distance traveled. We conclude that such a method is a viable component of a localization system, and propose the next steps for work in this area.

Thesis Supervisor: Gian Luca Mariottini

Title: Draper Supervisor

Thesis Supervisor: Sertac Karaman

Title: Associate Professor of Aeronautics and Astronautics

Acknowledgments

I would first like to acknowledge my advisor at Draper, Gian Luca Mariottini. His guidance throughout the process was invaluable. From him, I learned how to think about research projects in a broader context, and how to break down large problems into simpler and more manageable ones. I would also like to thank my academic advisor, Sertac Karman at MIT, for his advice on research direction and course selection. Several others at Draper, including Greg Busillo, Tim Dyer, and Alex Helderman went out of their way to help, and provided significant technical assistance with the simulation and drone flights, which were essential to this thesis. I am also grateful to those at the education office at Draper for allowing me the opportunity to do my master's research in conjunction with the Perception and Localization Systems group.

There are many people outside of my academic mentors and colleagues who I would like to acknowledge as well. The MIT cycling team provided me the opportunity to pursue a sport I love, but my teammates themselves are by far the most meaningful part of my cycling experience. I have had an absolute blast riding and racing with all of you, and have a wealth of fond memories from my time on the team. To Robbie Raymond, you've managed to make me laugh no matter how much work I was juggling, and you've motivated me to work harder at school and cycling both. Finally, to my parents, who are incredibly supportive and encouraging, you have been a positive influence on my life and education for as long as I can remember, and I am extremely grateful. Thank you for all the sacrifices you have made on my behalf. I remain amazed by, and appreciative of, your willingness to make countless road trips to Boston to support me in my various endeavors.

Contents

1	Introduction	13
1.1	Related Work	14
1.1.1	SLAM	14
1.1.2	Visual-Inertial Odometry	14
1.1.3	Visual Odometry	15
1.2	Original Contribution	16
2	Problem Statement	17
2.1	The Pinhole Camera Model	17
2.2	Analytical Optical Flow	19
2.3	The RetiNav Problem	20
3	Technical Approach	23
3.1	Overview	23
3.2	Feature Detection	24
3.3	Feature Tracking	24
3.4	Optical Flow Estimation	24
3.5	Optical Flow Descriptor Calculation	25
3.5.1	Overview	25
3.5.2	Descriptor Computation	25
3.6	Machine Learning Algorithms	29
3.6.1	Datasets	29
3.6.2	Machine Learning Algorithm Choice	30

3.6.3	Artificial Neural Networks	31
3.6.4	Random Forests	31
4	Experimental Results	33
4.1	Overview	33
4.2	Evaluation Metrics	33
4.2.1	Evaluation Metric Selection	33
4.2.2	Total Traveled Distance Error	34
4.3	Data Acquisition and Pre-Processing	35
4.3.1	Data Sources	35
4.3.2	Data Acquisition	35
4.4	Parameter Study	37
4.4.1	Overview	37
4.4.2	Test Data	38
4.4.3	Results & Analysis	38
4.5	Trajectory Tests	41
4.5.1	Overview	41
4.6	Image Resolution Testing	43
4.7	Environmental Changes	45
4.7.1	Overview	45
4.7.2	Process	45
5	Conclusion & Future Work	49

List of Figures

3-1	The RetiNav Algorithm	23
3-2	Partitioned Image with Optical Flow Overlaid	26
4-1	Creation of Training and Test Sets	37
4-2	Model Performance on Vertical Test Flights	39
4-3	Model Performance on Lateral Test Flights	39
4-4	Model Performance on Forward Test Flights	40
4-5	Estimated vs. Actual Simulation Trajectories	42
4-6	Estimated vs. Actual Flight Trajectories	43
4-7	Effect of Image Resolution on Model Performance	44
4-8	Model Performance with Changing Test Environments	46

List of Tables

2.1	Optical Flow Field Examples	20
3.1	Optical Flow Descriptors	29

Chapter 1

Introduction

In recent years, the demand for autonomous vehicles, drones in particular, has increased significantly for both commercial and military applications. In order to navigate autonomously, these vehicles must be able to perform localization, or determine their location in their environment, and respond with the appropriate control action. Relative to localization, the control systems of autonomous drones are fairly well established. The localization problem, however, is still of great interest in the robotics community. A significant amount of work has been done to develop localization systems for platforms which are well-equipped with a variety of sensors, and large amounts of computational resources. For instance, Simultaneous Localization and Mapping (SLAM) systems can discern both a map of the environment and the location of the vehicle within it [5]. Running SLAM, however, is very computationally expensive.

Some platforms with low SWAP (size, weight, and power) budgets, are not physically large enough to support the necessary sensors, or provide sufficient computing power for SLAM. It is challenging, though necessary, to design a localization method for autonomous low-SWAP vehicles. Relative to larger ones, they offer distinct advantages, such as the ability to operate in physically constrained environments. Furthermore, they are more discrete, typically low-cost, and can be fairly easily maneuvered. Many of these vehicles already have a small camera on-board, and so vision-based localization methods are particularly promising since they do not require any addi-

tional sensors. Thus, we are motivated to work towards a computationally efficient, vision-based localization method for low-SWAP vehicles.

1.1 Related Work

1.1.1 SLAM

Visual-inertial odometry, or VIO, is the use of the camera and the inertial measurement unit (IMU) to localize a vehicle. VIO is often considered a "special case" of SLAM, in which the goal is to localize the vehicle relative to its initial location [20], while not necessarily mapping the entire surroundings as is done in SLAM. There are a variety of different implementations of SLAM in use. ORB-SLAM, for instance, is based upon the detection and tracking of ORB features from frame to frame [14]. The ORB feature detection algorithm is similar to SIFT (Scale-Invariant Feature Transform) and SURF (Speeded-Up Robust Features), but is more robust to noise, and more computationally efficient [16]. Another implementation of SLAM is LDSO, or Direct Sparse Optimization with Loop Closure [7]. LDSO, unlike ORB-SLAM, does not rely on the detection or tracking of features. Instead, it is a "direct method", meaning that it uses image brightness gradient information. The lack of dependence on distinct features makes LDSO more robust in areas that are lacking in trackable features.

1.1.2 Visual-Inertial Odometry

Relative to SLAM, VIO is significantly less costly from a computational standpoint, and requires fewer sensors. As a result, it is better-suited to low-SWAP budget platforms. A number of VIO systems have been developed for use on such platforms. One such example is Navion [20], which is specifically for use on nano and pico drones, or drones with power budgets on the order of 100mW. Navion is a VIO accelerator in the form of a small chip, which fuses IMU and camera data to estimate the vehicle's 3-D position, as well as a map of its surrounding environment. Navion is an example

of a localization method that works well with low-SWAP vehicles.

1.1.3 Visual Odometry

Visual odometry (VO) is similar to VIO, except that IMU data is not used. Another system, Semi-Dense Visual Odometry, or SVO, is a direct method that estimates a semi-dense inverse depth map from the image, propagates the depth map forward from frame to frame, and uses the depth map to localize the camera [6]. SVO is computationally lightweight, and does not require a second camera, or a depth sensor.

Navion and SVO are two examples of state-of-the-art systems that are suitable for low-SWAP platforms, and have built upon significant previous efforts to develop localization methods. One example of another approach to the localization problem involved the use of RGB-D cameras, which provide depth, in addition to brightness measurements [12]. Another approach, as described in [11], is a stereo method which involves the use of two, non-overlapping cameras, and leverages the generalized camera model. With RetiNav, however, we take a slightly different approach from traditional SLAM, SVO, Navion, and the other methods described here.

In 2014, a group of researchers used visual odometry to localize instruments being used for medical procedures [1]. Their approach leveraged optical flow, or the movement of features in the image between consecutive frames, and was machine-learning based. A neural network was trained to estimate the velocity of the instrument based on the optical flow field, and the results were encouraging. They achieved as little as 3.61mm of position error over a 174mm trajectory. The promising results demonstrated here motivated us to apply similar principles to develop a localization method for drones. Machine learning models are time-consuming to train, but once trained, can make predictions quickly. Hence, they are a particularly attractive option for low-SWAP vehicles, since the model can be trained offline and subsequently loaded onto the drone.

1.2 Original Contribution

To the best of our knowledge, this is the first use of learning-based visual odometry for drones. Our method is particularly targeted towards small vehicles that cannot support any of the existing methods which have been developed for larger vehicles. The concepts used here are similar to those first presented in [1], and further tested, again for medical applications, in [2]. Here, we provide a number of additional contributions. We investigate the feasibility of the method for use on a vehicle with more complex motion patterns than those seen with the medical instruments, and we test in a variety of environments which are typically challenging for VO algorithms. We conduct several experiments to determine the optimal algorithm parameters, and the most effective training methods for machine learning models.

Chapter 2

Problem Statement

The goal of RetiNav is to use optical flow, or the "apparent motion of the brightness pattern" between consecutive images [9], in order to estimate the motion of the camera, and hence the platform itself. We begin by introducing the concepts and equations that are relevant to the problem.

2.1 The Pinhole Camera Model

A set of points in 3-dimensional space, in the reference frame of the camera, is denoted as shown in Equation 2.1, where the entries of \mathbf{X}_c are measured in meters, and n features have been detected.

$$\mathbf{X}_c = \begin{bmatrix} X_{c,1} & \cdots & X_{c,n} \\ Y_{c,1} & \cdots & Y_{c,n} \\ Z_{c,1} & \cdots & Z_{c,n} \end{bmatrix} \quad (2.1)$$

The pinhole camera model [9] allows us to calculate the projection of \mathbf{X}_c onto the image frame. We denote this projection of \mathbf{X}_c onto the image frame as \mathbf{u}_c . The x and y components of \mathbf{u}_c are u and v respectively, which are measured in pixels.

$$\mathbf{u}_c = \begin{bmatrix} u_{c,1} & \cdots & u_{c,n} \\ v_{c,1} & \cdots & v_{c,n} \end{bmatrix} \quad (2.2)$$

To compute the j th entry of \mathbf{u}_c , we use Equations 2.3 and 2.4, which come from the pinhole camera model. Knowledge of camera calibration parameters u_o , v_o , f , K_u , and K_v is required. (u_o, v_o) is the center, or principal point, of the image in pixels, f denotes the focal length of the camera in meters, and K_u and K_v represent the number of pixels per meter in the x and y directions, respectively. We use $(u_{c,j}, v_{c,j})$ to represent the j th entry of \mathbf{u}_c , where j ranges from 1 to n .

$$u_{c,j} = u_o + fK_u \frac{X_{c,j}}{Z_{c,j}} \quad (2.3)$$

$$v_{c,j} = v_o + fK_v \frac{Y_{c,j}}{Z_{c,j}} \quad (2.4)$$

We use $x_{c,j}$ and $y_{c,j}$ respectively to denote the ratio the ratio of the x and y coordinate of a point to the depth.

$$x_{c,j} = X_{c,j}/Z_{c,j} \quad (2.5)$$

$$y_{c,j} = Y_{c,j}/Z_{c,j} \quad (2.6)$$

Therefore, we can re-write Equations 2.3 and 2.4 as follows:

$$x_{c,j} = \frac{u_{c,j} - u_o}{fK_u} \quad (2.7)$$

$$y_{c,j} = \frac{v_{c,j} - v_o}{fK_v} \quad (2.8)$$

Thus, $x_{c,j}$ and $y_{c,j}$ are the coordinates of a point projected into the image frame, where the principal point (u_o, v_o) is the center of the coordinate system. Similarly to Equation 2.9, we can write

$$\mathbf{x}_c = \begin{bmatrix} x_{c,1} & \cdots & x_{c,n} \\ y_{c,1} & \cdots & y_{c,n} \end{bmatrix} \quad (2.9)$$

2.2 Analytical Optical Flow

RetiNav uses optical flow as a means to infer the motion of the camera itself. The optical flow (\dot{x}, \dot{y}) , at a given pixel location $\mathbf{x}_{c,j}$, can be computed directly using Equations 2.12 and 2.13 when the translational and rotational camera motion, \mathbf{v}_c and ω_c is known. The units of optical flow are pixels per second.

$$\mathbf{v}_c = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \quad (2.10)$$

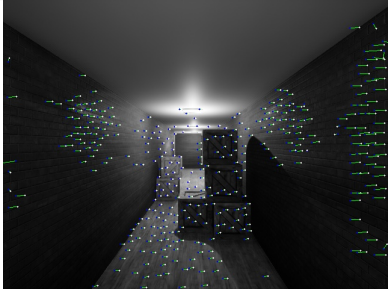
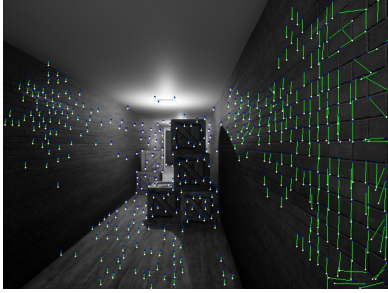
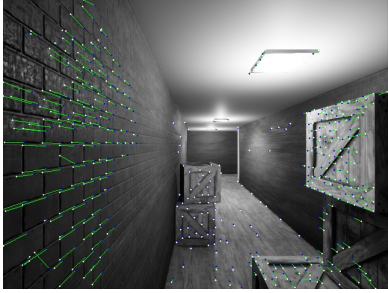
$$\omega_c = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \quad (2.11)$$

$$\dot{x}_j = -v_x/Z_{c,j} + x_{c,j}v_z/Z_{c,j} + x_{c,j}y_{c,j}\omega_x - (1 + x_{c,j}^2)\omega_y + y_{c,j}\omega_z \quad (2.12)$$

$$\dot{y}_j = -v_y/Z_{c,j} + y_{c,j}v_z/Z_{c,j} - x_{c,j}y_{c,j}\omega_y + (1 + y_{c,j}^2)\omega_x - x_{c,j}\omega_z \quad (2.13)$$

Equations 2.12 and 2.13 can be applied to each point in the scene, i.e. $j = 1, \dots, n$, resulting in an optical flow field. These equations show that the profile of the optical flow field changes based on the direction of motion of the camera. Several examples of the optical flow fields which result from different types of camera motion are shown in Table 2.1. Horizontal optical flow vectors indicate lateral motion, and vertical ones indicate vertical motion. Diagonal lines emanating outwards from the center of the image indicate forward motion. The ability to infer information about camera motion from the optical flow field is the premise on which RetiNav was developed.

Table 2.1: Optical Flow Field Examples

Lateral Motion	Vertical Motion	Forward Motion
		

2.3 The RetiNav Problem

We can re-write Equations 2.12 and 2.13 in matrix form, where \mathbf{J} is known as the interaction matrix.

$$\mathbf{J} = \begin{bmatrix} -\frac{1}{Z_{c,j}} & 0 & \frac{x_{c,j}}{Z_{c,j}} & x_{c,j}y_{c,j} & -(1+x_{c,j})^2 & y_{c,j} \\ 0 & -\frac{1}{Z_{c,j}} & \frac{y_{c,j}}{Z_{c,j}} & (1+y_{c,j})^2 & -x_{c,j}y_{c,j} & -x_{c,j} \end{bmatrix} \quad (2.14)$$

$$\dot{\mathbf{x}}_c = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \mathbf{J} \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \quad (2.15)$$

\mathbf{J} depends on the camera frame location of a point, $(x_{c,j}, y_{c,j})$ as well as $Z_{c,j}$, which is the depth of the point in the 3-D reference frame of the camera. When given an image, the depth to the point $Z_{c,j}$ is unknown, leaving us with incomplete knowledge of \mathbf{J} .

In order to estimate camera velocity, we are left with two primary tasks - firstly, to estimate the optical flow, as we cannot compute it analytically. We do this through the use of existing and well-established computer vision algorithms which will be described in subsequent sections. The second task is to train an algorithm to make

the association between an optical flow field, and a camera velocity, and herein lies the contribution of this work. Section 3 will describe the process of estimating optical flow, creating a training data set, and training a machine learning algorithm.

Chapter 3

Technical Approach

3.1 Overview

The process of calculating the optical flow from a pair of images, processing it into a descriptor, and forming training and test sets is summarized by Figure 3-1, and detailed in the remainder of this section.

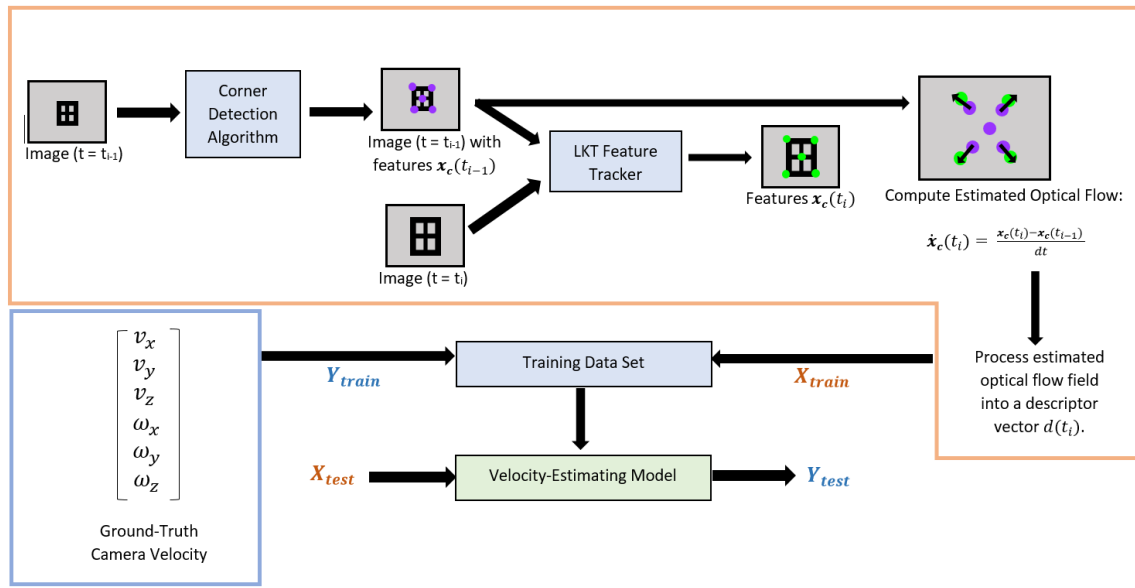


Figure 3-1: The RetiNav Algorithm

3.2 Feature Detection

The first step in computing optical flow between a pair of images is to detect features in the first image. In other words, if a pair of images comes from times t_{i-1} and t_i , we want to obtain $\mathbf{x}_c(t_{i-1})$. These features are distinctive elements of the image, such as corners. Features can be identified in a subsequent frame, in which they will appear visually similar, but provided that the camera is moving, they will have different image coordinates. For feature detection, we use the Shi-Tomasi [19] corner detection algorithm, which has been implemented in the OpenCV library [3].

3.3 Feature Tracking

Now that we have obtained $\mathbf{x}_c(t_{i-1})$, we need to know the location of these features in the subsequent image frame, $\mathbf{x}_c(t_i)$. In order to accomplish this, we use the Lucas-Kanade feature tracking algorithm [13]. This algorithm takes as input $\mathbf{x}_c(t_{i-1})$, as well as the image from t_i , and outputs $\mathbf{x}_c(t_i)$. We once again use the OpenCV implementation of this algorithm.

3.4 Optical Flow Estimation

When given $\mathbf{x}_c(t_{i-1})$ and $\mathbf{x}_c(t_i)$, we have only to compute the displacement and divide by the elapsed time between images, dt , as in Equation 3.2, in order to compute the estimated optical flow.

$$\dot{\mathbf{x}}_c = \begin{bmatrix} \dot{x}_1 & \cdots & \dot{x}_n \\ \dot{y}_1 & \cdots & \dot{y}_n \end{bmatrix} \quad (3.1)$$

$$\dot{\mathbf{x}}_c(t_i) = \frac{\mathbf{x}_c(t_i) - \mathbf{x}_c(t_{i-1})}{dt} \quad (3.2)$$

Since the features detected in the initial image are likely to leave the field of view as the platform moves, we re-detect features as each new image frame becomes available, use the tracker to compute their location in the next image, and repeat

this process continually. A dataset of N images will result in $N - 1$ $\dot{\mathbf{x}}_c$ vectors, i.e. $\dot{\mathbf{x}}_c(t_2), \dots, \dot{\mathbf{x}}_c(t_N)$, since each $\dot{\mathbf{x}}_c$ requires a pair of consecutive images.

3.5 Optical Flow Descriptor Calculation

3.5.1 Overview

The optical flow as described in the previous section will become the input to a machine learning algorithm, which will use the magnitude and direction of the optical flow vectors to infer the velocity of the moving platform. This machine learning algorithm will expect a consistently-sized input. However, the number of features that are detected will change from frame to frame based on the visual properties of the scene, such as the lighting and the composition of the scene, as well as the image resolution. Thus, we must process the optical flow field into a descriptor, which will serve as the input to the machine learning algorithm. The descriptor is computed first by partitioning the image in a certain predetermined pattern, such as a grid. Within each section of the partitioned space, we extract the median optical flow. The medians from each section are then concatenated into a single column vector, which serves as the descriptor for the corresponding set of images. This process will be described in detail in the subsequent sections.

3.5.2 Descriptor Computation

This section describes the process of taking the estimated optical flow field at a certain time t_i , which is $\dot{\mathbf{x}}_c(t_i)$, and processing it into the optical flow descriptor corresponding to images from times t_i and t_{i-1} . The diagram in Figure 3-2 represents the image at time t_i , with the optical flow vectors $\dot{\mathbf{x}}_c(t_i)$ overlaid. Here, the image is partitioned into a grid of size $n \times n$. We will later vary the method of partitioning the image for purposes of experimentation, but here we focus on the grid pattern for purposes of illustration.

The tracker identifies features $1, \dots, n$, which may be distributed over the image in

any order. We use $\dot{\mathbf{x}}_{c,j}$ to represent the j th column of $\dot{\mathbf{x}}_c$, where j ranges from 1 to n .

$$\dot{\mathbf{x}}_{c,j} = \begin{bmatrix} \dot{x}_j \\ \dot{y}_j \end{bmatrix} \quad (3.3)$$

Once $\dot{\mathbf{x}}_c$ is obtained from the tracker, the individual optical flow vectors $\dot{\mathbf{x}}_{c,j}$ must be assigned to their corresponding sections of the partitioned image. The diagram in Figure 3-2 depicts an image that has been partitioned into a grid of size $n \times n$, and 12 features have been detected. The features are distributed throughout the image in the order in which they are output by the tracker, which for our purposes, is arbitrary. The optical flow vectors corresponding to the features $\mathbf{x}_{c,1}, \dots, \mathbf{x}_{c,12}$ are $\dot{\mathbf{x}}_{c,1}, \dots, \dot{\mathbf{x}}_{c,12}$.

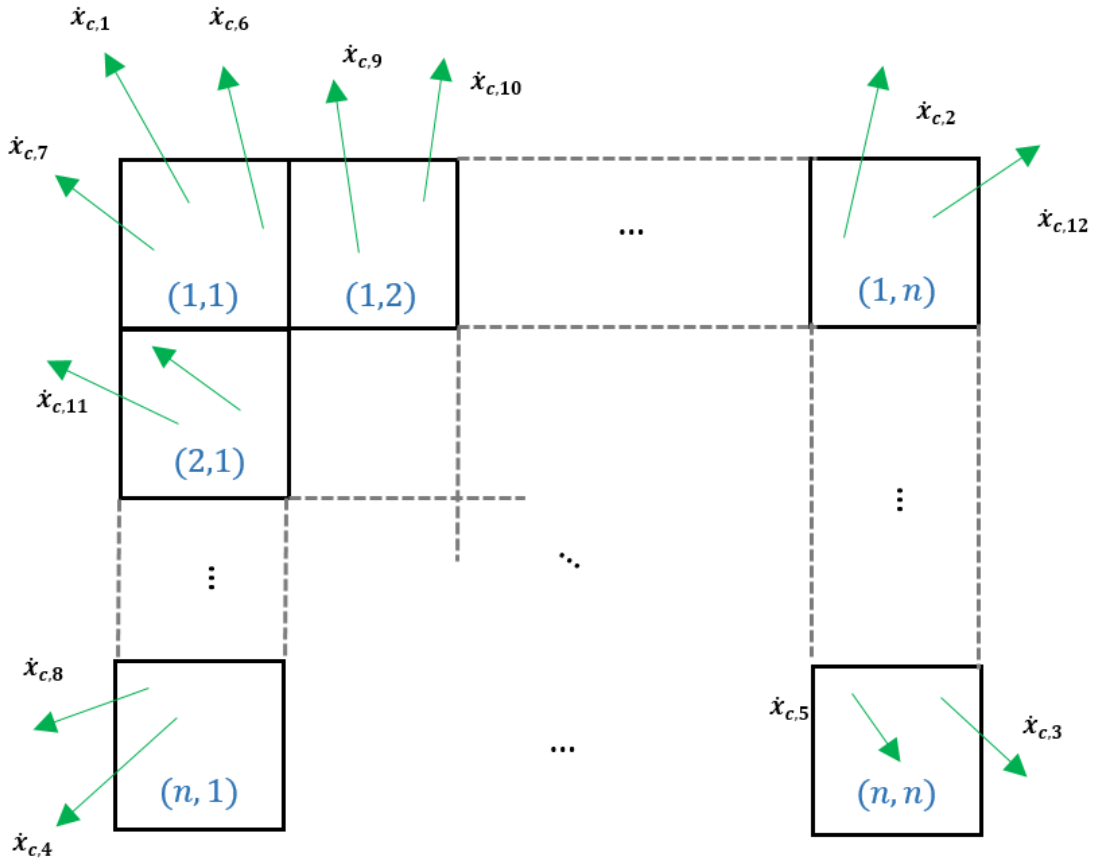


Figure 3-2: Partitioned Image with Optical Flow Overlaid

Based on the example in the above diagram, optical flow vectors $\dot{\mathbf{x}}_{c,1}$, $\dot{\mathbf{x}}_{c,6}$, and

$\vec{x}_{c,7}$ have been assigned to the upper left grid section. We break down the optical flow vectors in each section into their x and y components as follows:

$$\dot{\mathbf{x}}_{c,(1,1),x}(t_i) = \begin{bmatrix} \dot{x}_1 & \dot{x}_6 & \dot{x}_7 \end{bmatrix} \quad (3.4)$$

$$\dot{\mathbf{x}}_{c,(1,1),y}(t_i) = \begin{bmatrix} \dot{y}_1 & \dot{y}_6 & \dot{y}_7 \end{bmatrix} \quad (3.5)$$

We then repeat a similar process to obtain the vectors corresponding to the remainder of the partitioned spaces, i.e. through $\dot{\mathbf{x}}_{c,(n,n),x}(t_i)$ and $\dot{\mathbf{x}}_{c,(n,n),y}(t_i)$.

Once these vectors have been constructed, we have one final step before we are ready to assemble the descriptor: we extract the median of each one. The matrices $\hat{\mathbf{d}}_x(t_i)$ and $\hat{\mathbf{d}}_y(t_i)$ denote the $n \times n$ matrices composed of the medians of each vector, broken down into components.

$$\hat{\mathbf{d}}_x(t_i) = \begin{bmatrix} med(\dot{\mathbf{x}}_{c,(1,1),x}(t_i)) & \cdots & med(\dot{\mathbf{x}}_{c,(1,n),x}(t_i)) \\ \vdots & \ddots & \vdots \\ med(\dot{\mathbf{x}}_{c,(n,1),x}(t_i)) & \cdots & med(\dot{\mathbf{x}}_{c,(n,n),x}(t_i)) \end{bmatrix} \quad (3.6)$$

$$\hat{\mathbf{d}}_y(t_i) = \begin{bmatrix} med(\dot{\mathbf{x}}_{c,(1,1),y}(t_i)) & \cdots & med(\dot{\mathbf{x}}_{c,(1,n),y}(t_i)) \\ \vdots & \ddots & \vdots \\ med(\dot{\mathbf{x}}_{c,(n,1),y}(t_i)) & \cdots & med(\dot{\mathbf{x}}_{c,(n,n),y}(t_i)) \end{bmatrix} \quad (3.7)$$

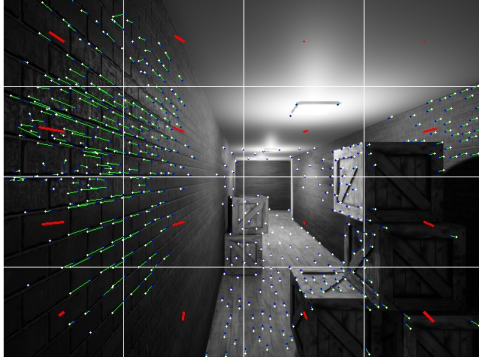
For an $n \times n$ grid, the final, column-vector descriptor will be of length $2n^2 \times 1$. When using the polar schemes, the size of the descriptor be twice the number of image sections, either 16 or 56. The entries of the descriptor will correspond to the medians of $\dot{\mathbf{x}}_{c,(1,1),x}$ and $\dot{\mathbf{x}}_{c,(1,1),y}$ through $\dot{\mathbf{x}}_{c,(n,n),x}$ and $\dot{\mathbf{x}}_{c,(n,n),y}$, i.e. the entries of $\hat{\mathbf{d}}_x(t_i)$ and $\hat{\mathbf{d}}_y(t_i)$. We denote the descriptor corresponding to the images at time t_{i-1} and t_i as \mathbf{d}_i , where \mathbf{d}_i is composed according to Equation 3.8. We have now shown how to construct the descriptor corresponding to an arbitrary pair of consecutive images collected from a flight. This process is repeated similarly over the entire dataset.

$$\mathbf{d}_i = \begin{bmatrix} \hat{\mathbf{d}}_x(t_i)(1, 1) \\ \hat{\mathbf{d}}_y(t_i)(1, 1) \\ \hat{\mathbf{d}}_x(t_i)(1, 2) \\ \hat{\mathbf{d}}_y(t_i)(1, 2) \\ \vdots \\ \hat{\mathbf{d}}_x(t_i)(n, n) \\ \hat{\mathbf{d}}_y(t_i)(n, n) \end{bmatrix} \quad (3.8)$$

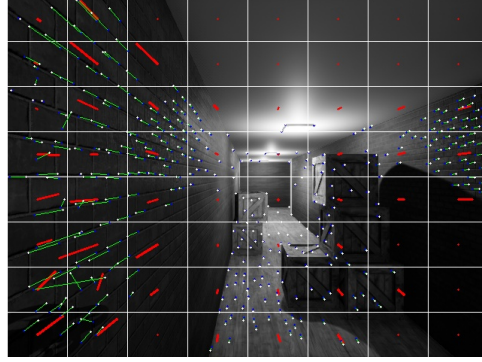
To implement the assignment of vectors $\dot{\mathbf{x}}_c(t_i)$ to sections of the partitioned image, we make use of the SciPy 2-D Binned Statistic Function [10]. In addition to solving the assignment problem, this function computes the statistic of choice over the data in each section. For RetiNav, we have chosen to extract the median of each section. This is a result of the observation that the median improved the robustness of RetiNav to errors in feature detection or tracking.

In this work, we experiment with a variety of methods for computing the descriptor in order to determine its effect on the accuracy of the estimates. We vary the partitioning schemes - implementing both a Cartesian grid as well as a polar scheme. The polar scheme uses circles and lines emanating outwards from the center of the image. In addition, we vary the resolution of the descriptor, or how finely the image is partitioned. Table 3.1 provides examples of the four different types of descriptors used in this work, in which the bold lines in each section represent the values used in the descriptor.

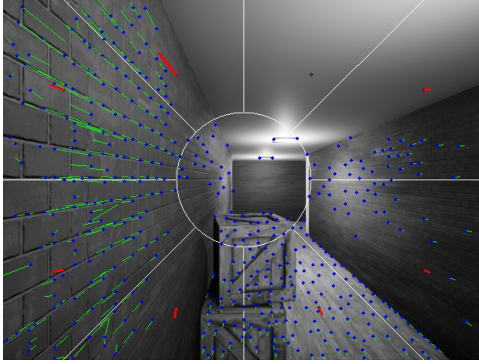
4x4 Grid Descriptor



8x8 Grid Descriptor



Basic Polar Descriptor



Complex Polar Descriptor

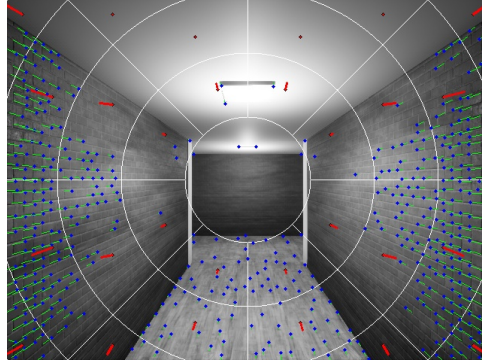


Table 3.1: Optical Flow Descriptors

3.6 Machine Learning Algorithms

3.6.1 Datasets

We can form training and test sets using the optical flow descriptors, along with the ground truth camera velocities. If a flight contains N images, we will have $N - 1$ descriptors and velocity measurements, since each descriptor requires a pair of images. We represent the vector of combined translational and rotational velocities as in Equation 3.9.

$$\mathbf{v}_i = \begin{bmatrix} v_x(t_i) \\ v_y(t_i) \\ v_z(t_i) \\ \omega_x(t_i) \\ \omega_y(t_i) \\ \omega_z(t_i) \end{bmatrix} \quad (3.9)$$

We can now represent the training set, $(\mathbf{X}_{train}, \mathbf{Y}_{train})$, and the test set, $(\mathbf{X}_{test}, \mathbf{Y}_{test})$ with Equations 3.10 through 3.13. N represents the number of images in the flight used for training data, while M represents the number of images in a separate flight used for test data.

$$\mathbf{X}_{train} = \begin{bmatrix} \mathbf{d}_2 & \mathbf{d}_3 & \dots & \mathbf{d}_N \end{bmatrix} \quad (3.10)$$

$$\mathbf{Y}_{train} = \begin{bmatrix} \mathbf{v}_2 & \mathbf{v}_3 & \dots & \mathbf{v}_N \end{bmatrix} \quad (3.11)$$

$$\mathbf{X}_{test} = \begin{bmatrix} \mathbf{d}_2 & \mathbf{d}_3 & \dots & \mathbf{d}_M \end{bmatrix} \quad (3.12)$$

$$\mathbf{Y}_{test} = \begin{bmatrix} \mathbf{v}_2 & \mathbf{v}_3 & \dots & \mathbf{v}_M \end{bmatrix} \quad (3.13)$$

3.6.2 Machine Learning Algorithm Choice

We experiment with two different inference algorithms- a random forest regressor, as well as an artificial neural network (ANN). Convolutional neural networks, or CNNs, were considered as another potential machine learning algorithm for use with RetiNav since they are commonly used for image-processing applications. Ultimately, however, CNNs were not used for RetiNav since inputs to CNNs are typically much larger than a 4x4 or 8x8 grid descriptor, as we use here. Using the optical flow at every pixel would increase the size of the input to a more traditional size. However, this would result

in many zero-valued entries, and would make the input more subject to tracking and detection errors, a problem that is mitigated by partitioning the image and extracting the median optical flow values. Thus, we stick to Random Forests and ANN's, using the scikit-learn [15] Python implementations for both.

3.6.3 Artificial Neural Networks

An artificial neural network is a network of weights that maps the inputs, which are optical flow descriptors in our case, into outputs, or velocity estimates. The weights are calculated through back-propagation with the mean-squared error between the estimate and the ground truth value from the training data. Further detail regarding ANN's as well as Random Forests, can be found in [8], which was referenced for this work. In order to ensure consistency in the magnitude of the input, we use min-max input scaling on the descriptors before training or testing a model. Some change in performance was observed as the number of layers in the ANN was changed. Thus, when training ANN's, several different sizes were tried, and the size that yielded the best results was selected. Generally a network size of 300-500 layers resulted in the best performance.

3.6.4 Random Forests

A random forest is an inference algorithm that is based on a collection of decision trees. A decision tree consists of a series of tests for a certain, automatically-detected, feature in the dataset. These tests become increasingly specific towards the outer edges of the tree. The leaves of the tree represent the output of the random forest, in this case, the velocity estimate. Random forests may be used for either regression or classification problems. Since our task is to estimate a velocity, we use random forests for regression. We use the default parameters assigned by scikit-learn, which result in a random forest consisting of 10 decision trees.

As will be described in detail in the following section, we experiment with a variety of descriptors, machine learning algorithms, flight environments, and test trajectories

to evaluate the performance of RetiNav.

Chapter 4

Experimental Results

4.1 Overview

Here we present the results of testing RetiNav under a variety of experimental conditions. We begin by explaining our choice of metrics for evaluation, as well as the method used to obtain and pre-process our data. We then analyze RetiNav’s performance on basic, one-dimensional trajectories. We break down these results by direction of motion, descriptor type, and machine learning algorithm. Informed by the results of this analysis, we select a descriptor type, and a machine learning algorithm for use in subsequent tests. We proceed to test RetiNav’s performance on longer and more complicated flight patterns, both in simulation and real flight. We then present results when the resolution of the test imagery is not the same as the training resolution. Lastly, we analyze the performance of RetiNav when the testing environment is different from the training environment.

4.2 Evaluation Metrics

4.2.1 Evaluation Metric Selection

Before presenting results, we first explain our choice of a metric by which to evaluate our localization method. RetiNav outputs an estimated velocity, and we are given

ground truth position either from simulation, or from a motion capture system. Thus we have two options. We may propagate our estimated velocities forward in time, obtain an estimated position as a function of time, and compare ground truth position to estimated position. Alternatively, we may convert our ground truth position data into a velocity, and compare estimated velocity against actual velocity. We have chosen to do the former - compare a trajectory generated from estimated velocities against the ground truth position of the vehicle. We feel that being able to compare the estimated location of the vehicle against the true location is a much clearer indicator of the performance of the localization method than directly analyzing the error of the velocity estimate.

However, there is one subtlety of this comparison. In order to generate datasets for training and testing, we take position data from simulation, or a motion capture system, and convert it into a velocity. In order to obtain a smooth ground truth velocity, we must do some post processing on the ground truth position data. Thus, if we were to compare our estimated trajectory directly to the ground truth position data, a slight amount of additional error introduced by the post-processing would be unaccounted for. To avoid this issue and obtain the fairest possible comparison, we take our post-processed, ground-truth velocities, and propagate them forward. We now have a trajectory which may differ very slightly from the position data output by the simulation or motion capture system. We compare this trajectory to our estimated one, and thus eliminate any additional error that may be introduced by during post-processing.

4.2.2 Total Traveled Distance Error

In order to evaluate the method, we have elected to use the percent total traveled distance error, $Error_{TTD}$, expressed in Equation 4.2. This metric reflects the discrepancy between the estimated and actual trajectories, expressed as a percentage of the total distance traveled. Using Equation 4.1, we obtain the ground truth total traveled distance, where $\Delta t_i = t_i - t_{i-1}$. Thus, for a set of n ground-truth velocity measurements $\mathbf{v}_c(t_i)$,

$$d_{traveled} = \sum_{i=1}^n |\mathbf{v}_c(t_i) \Delta t_i| \quad (4.1)$$

We now define \mathbf{X}_{drone}^w as the 3-dimensional vector from the drone’s starting point, to its ground-truth endpoint. Similarly, $\hat{\mathbf{X}}_{drone}^w$ is the vector from the starting point to the estimated endpoint. The %TTD error can now be expressed as follows:

$$Error_{TTD} = \frac{||\mathbf{X}_{drone}^w - \hat{\mathbf{X}}_{drone}^w||}{d_{traveled}} \quad (4.2)$$

4.3 Data Acquisition and Pre-Processing

4.3.1 Data Sources

Some results presented here were obtained by training and testing on simulation-generated data, others from real flight data. Our simulation is built upon the AirSim [18] framework, which is an open-source simulator for autonomous vehicles that leverages the Unreal graphics engine. Our flight data was collected by flying a Parrot Bebop drone in a motion capture room equipped with OptiTrack cameras.

We train the models to estimate both translational and rotational velocity. However, we have chosen to test on trajectories which involve minimal rotation. This choice allows us to focus on determining the best parameters and design choices for RetiNav while using simple trajectories where it is easier to test the effect of changes. Furthermore, the problem of estimating rotation is essentially already solved by IMUs so priority was placed on the estimation of translational velocity.

4.3.2 Data Acquisition

In order to generate data on which to train and test a model, we follow the same process whether flying in simulation or the laboratory. We initially position the vehicle facing a wall, and fly it back and forth in one single plane of motion at a time. We fly the drone back and forth horizontally, vertically, and forward and backward for several minutes each. During the flights, we record ROS bag files, which contain

the imagery as well as the position data. Following the flights, some post-processing on the data is required. The bag file is split so as to remove occasional dropped frames, and unintended motion of the drone. The remaining bag files are then split into segments in which the drone starts from zero velocity, flies in its desired plane of motion, and comes to a stop. We have chosen to do this so as to minimize the amount of training data during which the drone is at a standstill, as this data was often particularly noisy.

Following the steps described in the Technical Approach, we process all of the data from these short segments into optical flow descriptors and velocity measurements. In order to build a training set, we then concatenate the majority of these smaller datasets into a single large \mathbf{X}_{train} , the descriptors, and \mathbf{Y}_{train} , the corresponding velocities. If we have P segments allocated for training, then our training sets are formed as follows, where each individual \mathbf{X}_{train} and \mathbf{Y}_{train} is formed according to Equations 3.10 and 3.11. If we are using a grid descriptor of size $n \times n$, then \mathbf{X}_{train} is of size $2n^2 \times P$, and \mathbf{Y}_{train} is of size $6 \times P$.

$$\mathbf{X}_{train} = \begin{bmatrix} \mathbf{X}_{train,1} & \mathbf{X}_{train,2} & \dots & \mathbf{X}_{train,P} \end{bmatrix} \quad (4.3)$$

$$\mathbf{Y}_{train} = \begin{bmatrix} \mathbf{Y}_{train,1} & \mathbf{Y}_{train,2} & \dots & \mathbf{Y}_{train,P} \end{bmatrix} \quad (4.4)$$

The process of breaking down a flight into short segments is shown visually in Figure 4-1. We prepare the testing data differently for different experiments, and will explain the processes in their corresponding sections.

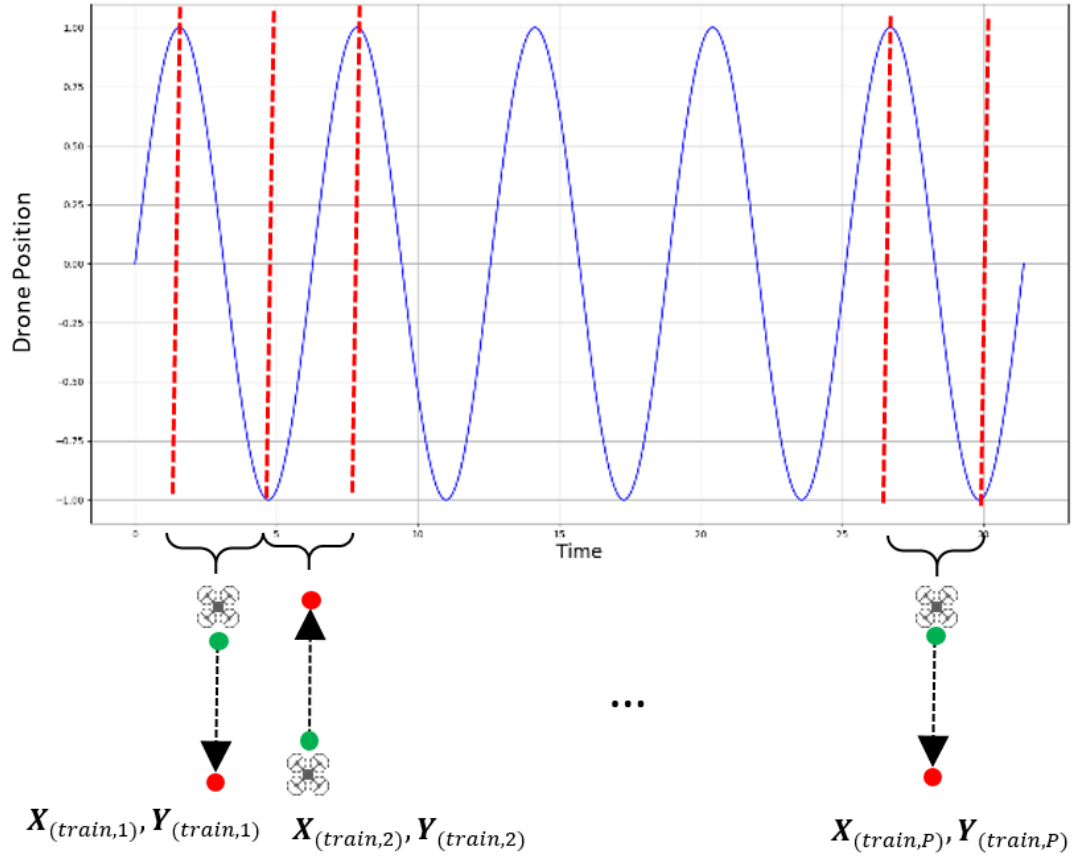


Figure 4-1: Creation of Training and Test Sets

4.4 Parameter Study

4.4.1 Overview

We have presented the various types of descriptors and machine learning algorithms that RetiNav can use, and here, we use a controlled procedure to compare RetiNav's performance under different options. We break down the results by direction of motion in order to further control the experiment.

4.4.2 Test Data

As described previously, each \mathbf{X}_{test} and \mathbf{Y}_{test} corresponds to a simple trajectory in which the drone starts from a standstill, flies in a given plane of motion, and comes to a stop. When creating a training dataset, as we described in the previous section, we concatenated these short segments together. When creating our test set, for this parameter study, we take a slightly different approach. Instead of concatenating flight segments together into a single large dataset, we leave each one in isolation. Thus, if there are Q segments allocated for testing, we will have Q individual test sets. Thus, we are able to evaluate $Error_{TTD}$ on a set of short flights. Here we set aside $Q = 30$ trajectories for testing, and group the remaining $P = 240$ together into the training set. Data for this parameter study was collected from flights in the motion capture laboratory.

4.4.3 Results & Analysis

With training and test sets prepared, we train a series of models - both random forests and artificial neural networks. In order to analyze motion in each plane separately, we train three individual models using data generated in a single plane of motion. For each plane of motion, we vary the type and resolution of the descriptor, as well as the machine learning algorithm. Figures 4-2, 4-3, and 4-4 show the results on the thirty test trajectories in %TTD error for the different types of descriptors and machine learning algorithms for vertical, horizontal, and forward motion. The whiskers of the boxplot extend to 1.5 times the interquartile range beyond the first and third quartiles.

For the better performing models, such as the random forest with the 4x4 grid or basic polar descriptors, we obtain about 3.5 %TTD error on the test trajectories. Notably, the random forest models consistently outperform the neural networks. Though it is difficult to identify the cause of this trend, one possibility is that the architecture of an ANN was too complex for the nature of these datasets, and overfit to the training data, resulting in poor performance on test data. The random forest,

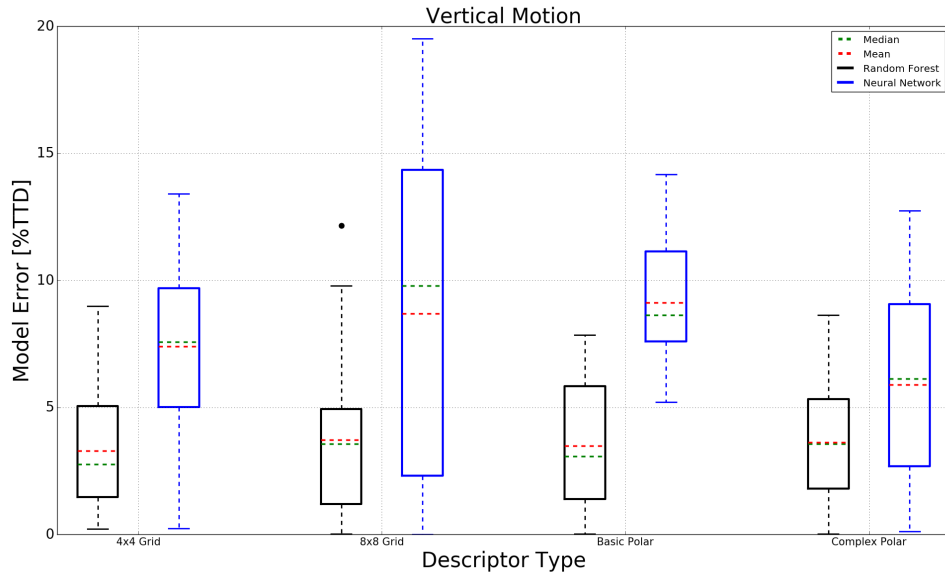


Figure 4-2: Model Performance on Vertical Test Flights

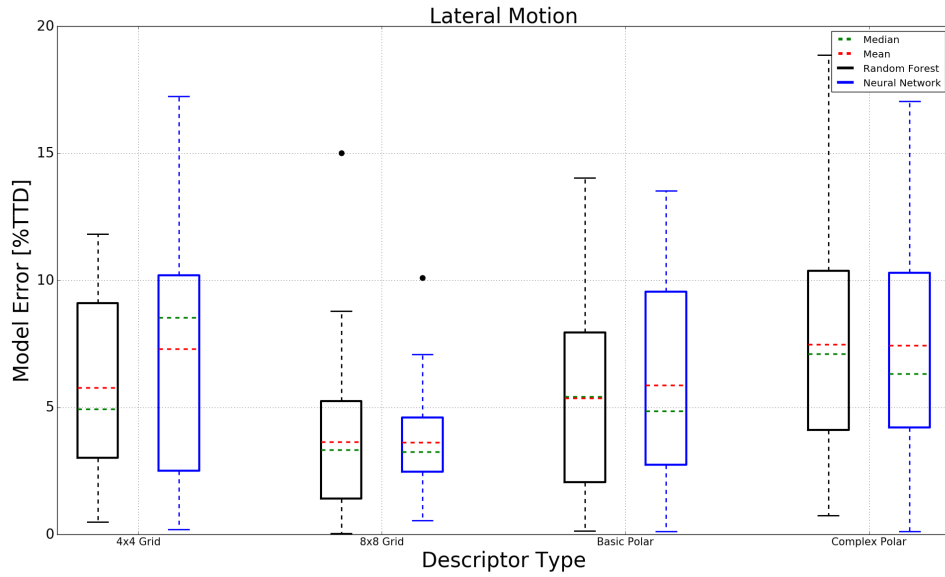


Figure 4-3: Model Performance on Lateral Test Flights

with its simpler tree-based structure, proved to be a better fit for this particular application. The results are broken down for each plane of motion, and are described below.

As we would expect, there is no meaningful performance difference between dif-

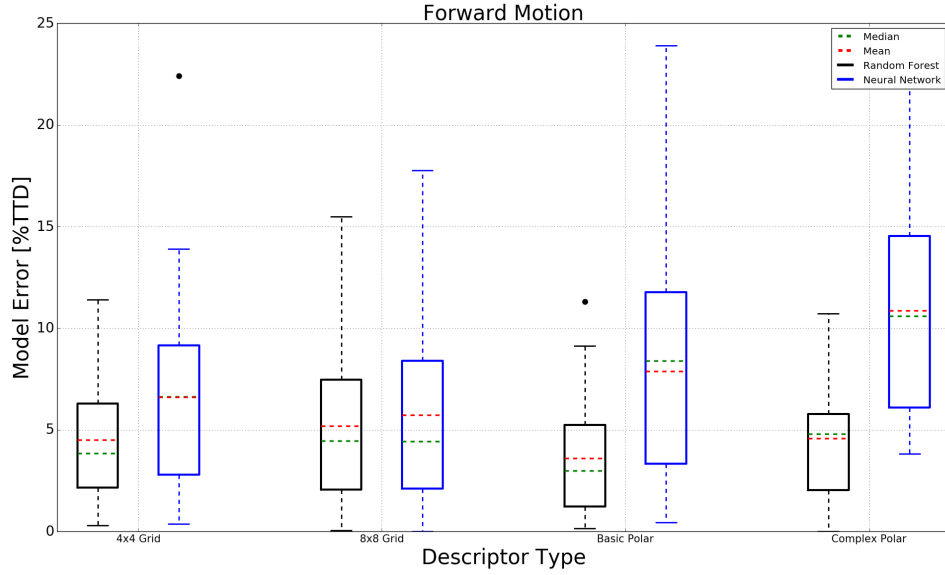


Figure 4-4: Model Performance on Forward Test Flights

ferent descriptors for vertical and horizontal motion. This is to be expected, since for vertical or horizontal motion in front of a flat wall, as we have here, the optical flow will be nearly the same everywhere in the image. Even for forward motion, where the optical flow radiates outward from the center of the image, the choice of descriptor does not make an appreciable difference. From this parameter study, we take away two primary conclusions. First, random forests are highly preferable to ANNs for RetiNav. Second, the pattern and resolution of the optical flow descriptor does not have a significant impact on algorithm performance. In subsequent tests, we therefore opt to use a random forest model with a 4x4 grid descriptor, since it is the simplest and least computationally expensive descriptor to form.

4.5 Trajectory Tests

4.5.1 Overview

We have shown that RetiNav can achieve reasonable %TTD errors on simple trajectories, in which the drone flies in a single plane of motion from point A to point B. We now seek to test the algorithm on a more complicated trajectory which would more closely reflect an actual use case. We generate two training sets according to the process described previously, one with flight data, and one with simulation data. We train two separate random forest models. We then fly a relatively complex trajectory in both environments, and these serve as the test sets for their corresponding models.

The test trajectory was generated by flying the drone, either in simulation or flight, in such a way so as to test all types of motion included in the training set. We process the imagery from this test flight into 4x4 grid descriptors, so as to obtain \mathbf{X}_{test} for simulation and flight. We then evaluate the models on their corresponding test sets, and compute the %TTD error. The estimated trajectories are plotted against the true trajectories in Figures 4-5 and 4-6 for simulation and flight, respectively. In simulation, we obtain a %TTD error of 1.4%, and on the flight data, the error is 4.1%. In the parameter study, we observed errors near 3%, so it is encouraging to see that the error shows no significant increase when evaluating on a lengthier and more complicated trajectory.

It is important to note that relative to the flight environment, the simulation environment presents some additional challenges. First, the lighting conditions are dim and shadowy, as shown in Figure 3.1. Lighting conditions such as this make feature detection and tracking more difficult, which reduces the quality of the optical flow estimates. Secondly, there are objects in the foreground. Optical flow, as demonstrated by Equations 2.12 and 2.13, is dependent on the depth of the objects in the image. For a given camera motion, features closer to the camera will appear to move more than features that are further from the camera. When flying laterally and vertically in the motion capture room, all optical flow vectors were of similar length since the depth of all features was constant over time, and uniform over the image. In the

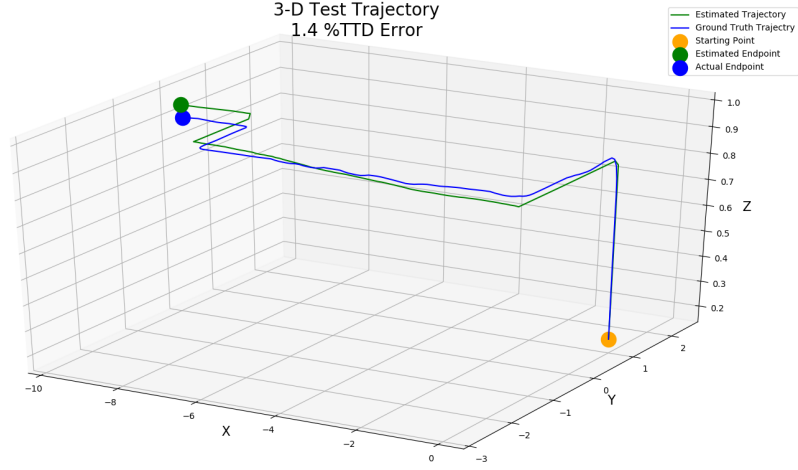


Figure 4-5: Estimated vs. Actual Simulation Trajectories

simulation environment, there are features in the foreground and background, which will result in optical flow vectors of varying lengths.

This experiment demonstrated that RetiNav can perform well on longer and more complicated trajectories, as well as in visually challenging environments. It also provides some assurance that the past good performance exhibited on simple test trajectories was not due to over-training. Previously, the test trajectories were similar to, albeit distinct from, the training trajectories. However, this particular test trajectory is different from the previous, simple, test ones since it includes changes of direction, and is longer in duration and distance traveled.

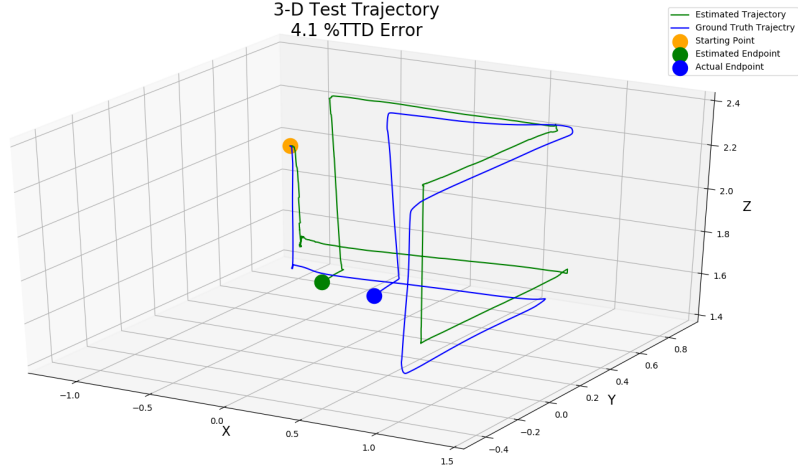


Figure 4-6: Estimated vs. Actual Flight Trajectories

4.6 Image Resolution Testing

Up to this point, the tests we have done with RetiNav have used data of the same quality for training and testing. However, in practice, it is not necessarily reasonable to expect that the data encountered during normal operation will match the quality of the training data. One important way in which the data quality may change is the camera’s resolution. Training data is often obtained from simulations, which can easily provide large quantities of high-resolution imagery. However, many small platforms with low size, weight, and power budgets may operate with a camera resolution that is significantly less than that of the training data. As camera resolution decreases, the performance of the feature detection and tracking algorithms degrades. Since optical flow is used to build the descriptors, this would likely result in lower quality descriptors. Additionally, changing the camera resolution may result in a different set of features being detected, compared to what is seen in training. Thus, testing on data that was generated from imagery of a different resolution doubles as a way to ensure that our method is not overtrained as a result of constantly seeing the same features in training and testing. Thus, we conduct an experiment to determine

the sensitivity of RetiNav to changes in camera resolution.

For this test, we use a single model - the same model described in the previous section which was trained on simulation data. In order to train this model, images of resolution 480x640 were processed into grid descriptors of size 4x4. In this experiment, images are still processed into 4x4 grid descriptors, but we begin with images of different and varying resolution. Using the Unreal simulation environment, we fly the same test trajectory 10 more times. The flights themselves are identical, but each time, we vary the resolution of the image that is collected from the flight. We vary the resolution from 50% to 150% of the original. In terms of pixels, the image size varies from 240x320 to 720x960. After following the usual process of converting optical flow into descriptors, we test the model on this set of ten test trajectories, once again evaluating with the %TTD metric. The results of these tests are shown in Figure 4-7.

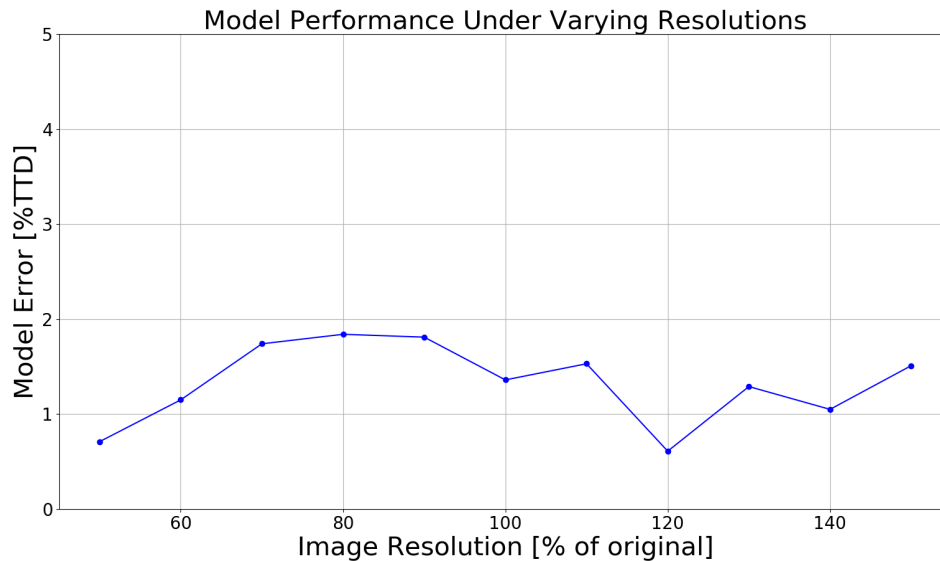


Figure 4-7: Effect of Image Resolution on Model Performance

In the the baseline case, as before, we see a %TTD error of approximately 1.4%. Despite large changes in the image resolution, the resulting changes in %TTD error are fairly minimal - we see a difference of only 1.2% TTD error between the best and worst performances. The aforementioned changes in which features are being

detected likely account for the performance differences between the different cases. However, there is no significant improvement or degradation in performance. From this experiment, we conclude that RetiNav is robust to changes in the image resolution between training and testing data. This is encouraging, as it suggests that RetiNav can be applied in cases where imagery encountered in the field is of lower quality than the images used for training.

4.7 Environmental Changes

4.7.1 Overview

In the tests described up to this point, the models have been tested on data from an environment that is exactly the same as the one which provided the training data. We now aim to ensure that the past performances of RetiNav were not a result of over-training in a specific environment. In this experiment, we present the results of RetiNav when the testing environment is changed relative to the training environment.

4.7.2 Process

For this experiment, we will use the same AirSim simulation environment described previously. However, we introduce physical change in the environment by modifying the width of the corridor. We will conduct 5 flights, and during each one the hallway will have a fixed width that is some percentage of original width used in training. The width will be varied from 80 to 120 percent of the original size.

In order to properly control for the changing environment, we fly the same trajectory in each test flight. As a result, we cannot use the same simulation trajectory presented previously. We instead fly a slightly different trajectory that allows the drone to avoid collision when the hallway is reduced to 80% of its original width. We collect imagery and velocity data from each test flight, and process the optical flow into \mathbf{X}_{test} , a set of 4x4 grid descriptors. We use the ground truth and estimated

velocities to calculate %TTD errors. The results of this experiment are shown in Figure 4-8.

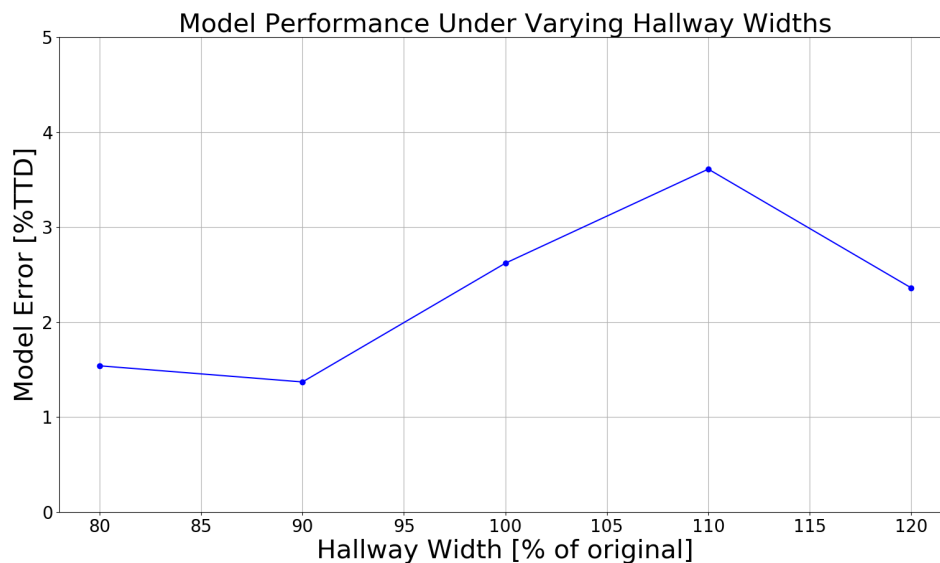


Figure 4-8: Model Performance with Changing Test Environments

On this particular trajectory, under the normal hallway width, RetiNav achieves a 2.6% TTD error. We see the performance degrade slightly when the width of the testing environment increases, and improve slightly when the width decreases. In the baseline case, for which the hallway is 100% of its normal width, the estimated endpoint is farther away from the starting point than the actual endpoint. That is, the estimate overshoots. A possible explanation for the improved performance with a narrower hallway, and worse performance with a wider hallway is as follows. In this particular trajectory, the drone moves laterally, but the majority of the time is spent moving forwards down the corridor. That is, v_z is non-zero, while the rest of the velocities, rotational and translational, are zero. If this is the case, we can re-write the x-component optical flow equation, Equation 2.12, as Equation 4.5:

$$\dot{x}_j = x_{c,j}v_z/Z_{c,j} \quad (4.5)$$

For a given feature in the scene, reducing the width of the hallway has the effect of decreasing the magnitude of $x_{c,j}$, since points to the right of the x-component of

the principal point, u_0 , are positive, and those to the left are negative. The depth of the point, $Z_{c,j}$, remains the same. The y-component of optical flow is unaffected. Thus, for a given camera position, and a given feature, narrowing the hallway will result in the optical flow vectors having a slightly smaller x-component. Generally, longer optical flow vectors are associated with faster camera motion, and shorter ones with slower motion. It is possible that the shortening of optical flow vectors in the narrower hallways compensates for the overshoot observed in the baseline trajectory, and thus accounts for the improved performance at 80 and 90% of the initial hallway widths. The same reasoning would explain why performance becomes slightly worse when the hallway is widened.

Chapter 5

Conclusion & Future Work

In this thesis, we have built upon previous efforts to design a machine learning-based visual odometry method. While previous works were primarily targeted towards medical applications, here we implement and test similar concepts in the context of autonomous drones. We leverage the connection between optical flow, and the motion of the camera which is capturing the imagery. We train machine learning models to make this association, and we test them on data gathered from a drone. We begin by doing a parameter study in which we experiment with different methods of processing optical flow, different types of machine learning models, etc. We move on to test the model on longer and more complicated trajectories which involve motion in multiple directions. We conclude by testing the robustness of the method to variables such as the image resolution, and discrepancies between the training and testing environments. In these tests, RetiNav demonstrates as low as 1.4% total traveled distance error on simulation data, and 4.1% on data from a real flight. This is particularly encouraging, since the only sensor involved in this localization method is the camera.

One potential avenue for future work is to leverage the IMU in conjunction with the camera. The models presented in this work were trained and tested on data which involves minimal amounts of rotation. Using the two together could allow RetiNav to be effective on trajectories that involve both rotation and translation.

Additionally, event-based cameras are of interest for future iterations of RetiNav.

Traditional cameras record a brightness measurement at every pixel, at a specific, pre-determined time interval. Event cameras, however, give a stream of binary outputs that correspond to only those pixels whose brightness has changed significantly [4]. Thus, the output of an event-based camera is very compact in comparison to a traditional camera. Despite a different output format, it is still possible to calculate the optical flow from an event-based camera, as demonstrated by [17]. The high-frequency of the event-based camera could facilitate similarly high frequency localization, which would be especially advantageous for fast-moving vehicles. In addition, the compact output of the camera would reduce the amount of data to be processed, which is helpful for low-SWAP vehicles.

Bibliography

- [1] Obstein-K. Valdastrì P. Bell, C. Image partitioning and illumination in image-based pose detection for teleoperated flexible endoscopes. *Artificial intelligence in medicine*, 2013.
- [2] Puerto-G. Mariottini G.L. Valdastrì P. Bell, C. 6 DOF pose estimation for teleoperated flexible endoscopes using optical flow: A comparative study. *International Conference on Robotics and Automation*, 2014.
- [3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [4] Linares-Barranco B. Culurciello E. Posch C. Delbrück, T. Activity-driven, event-based vision sensors. *International Symposium on Circuits and Systems*, 2010.
- [5] Bailey T. Durrant-Whyte, H. Simultaneous localisation and mapping (SLAM): Part I the essential algorithms. 2006.
- [6] Sturm J. Cremers D. Engel, J. Semi-dense visual odometry for a monocular camera. *IEEE International Conference on Computer Vision*, 2013.
- [7] X. Gao, R. Wang, N. Demmel, and D. Cremers. Ldso: Direct sparse odometry with loop closure. In *iros*, October 2018.
- [8] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly, 2017.
- [9] Berthold Klaus Paul Horn. *Robot Vision*. The MIT Press: McGraw-Hill Book Company, 1986.
- [10] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2019-04-17].
- [11] Kneip L. Nikolic J. Pollefeys M. Siegwart M. Kazik, T. Real-time 6d stereo visual odometry with non-overlapping fields of view.
- [12] Sturm J. Cremers D. Kerl, C. Robust odometry estimation for rgb-d cameras. *International Conference on Robotics and Automation*, 2013.
- [13] Kanade T. Lucas, B. An iterative image registration technique with an application to stereo vision. *International Joint Conference on Artificial Intelligence*, 1981.

- [14] Montiel J. M. M. Mur-Artal, Raúl and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.
- [15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [16] Rabaud V. Konolige K. Bradski G. Rublee, E. Orb: an efficient alternative to sift or surf. *International Conference on Computer Vision*, 2011.
- [17] Delbruck T. Rueckauer, B. Evaluation of event-based algorithms for optical flow with ground-truth from inertial measurement sensor. *Frontiers in Neuroscience*, 2016.
- [18] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.
- [19] Tomasi C. Shi, J. Good features to track. *IEEE Conference on Compute Vision and Pattern Recognition*, 1994.
- [20] Zhang Z. Carlone L. Karaman S. Sze V. Suleiman, A. Navion: A 2mw fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones. *Journal of Solid State Circuits*, 2018.