Autonomous Visual Navigation of a Quadrotor VTOL in complex and dense environments

Jean-Luc Stevens

A thesis submitted for the degree of
Master of Philosophy of
The Australian National University

February 2021

© Jean-Luc Stevens 2021

Except where otherwise indicated, this thesis is my own original work.

Jean-Luc Stevens
19 February 2021

Acknowledgments

I would love to acknowledge my colleagues, whom many are my friends, who provided support to me during the writing of this thesis. I would love to acknowledge my parents, who supported me in all things during me thesis in a loving manner, making sure I am always on path. I would also like to thank my church and God, in whom I would have gone insane without.

Foremost thanks to my supervisor, Robert Mahony, who guided me during the creation of this thesis, who provided valuable insight into the science and mathematics of my work, who spent large amounts of time in helping me write my conference paper, thesis proposal and this main thesis, and who persevered through my lack of understanding and preparedness in all things. He was invaluable in all of my research, and I would have just been completely lost without him, so I say thanks to him.

I would like to acknowledge the contribution of my supervisory panel - Robert Mahony, Viorella Ila, and Jochen Trumpf - who helped guide me down the right path during this thesis. Similarly, I would like to acknowledge Alex Martin who helped me design the quadrotor platform and provided valuable help during this thesis.

Abstract

This thesis presents a system design of a micro aerial vehicle platform, specifically a quadrotor, that is aimed at autonomous vision-based reactive obstacle avoidance in dense and complex environments. Most modern aerial system are incapable of autonomously navigating in environments with a high density of trees and bushes. The presented quadrotor design uses leading-edge technologies and inexpensive off-the-shelf components to build a system that presents a leap forward in technologies aimed at overcoming the issues with dense and complex environments.

Several major system requirements were met to make the design effective and safe. It had to be completely autonomous in standard operations and have a manual override function. It had to have its computational capability completely on-board along with vision processing ability. As such, all state estimation and visual guidance had to be performed on-board the vehicle, removing the need for remote connection which can easily fail in forest-like environments. The quadrotor had to be made from mostly off-the-shelf components to reduce cost and make it replicable. It also had to remain under 2kg to meet Australian commercial aerial vehicle regulations regarding licencing.

In order to meet the system requirements, many design decisions were developed and altered as needed. The main body of the quadrotor platform was based on off-the-shelf hobby assemblies. A Pixhawk 2.1 was the flight controller used due to its open-source code and design which included all sensors needed for state estimation, has manual override for control, and control the motors. A leading-edge computational device called the NVIDIA Tegra TX2 was used for vision processing on the quadrotor. The NVIDIA Tegra TX2's embedded NVIDIA Graphics Processing Unit (GPU), is compact and consumes low amounts of power. It also is capable of estimating dense optical flow on the GPU at rates of 120Hz when using a camera that outputs grey-scale images at a resolution of 376x240. The vision processor is responsible for providing directional guidance to the on-board flight controller. A design decision during the project was to include a 3-axis gimbal to stabilise the camera.

The quadrotor was shown to be able to hover and locally move both indoors and outdoors using the optical flow measurements. Optical flow measurements give a sense of velocity which can be integrated to get a position estimate, though it was susceptible to drift. The drift was compensated using a combination of recognisable targets and positioning systems such as GPS.

The experimental data obtained during the project showed that the algorithms presented in this thesis are capable of performing reactive obstacle avoidance. The reactive obstacle avoidance experiments were performed in both simulation and in real world environments, including the dense forest-like environments. By fusing vehicle speed estimates with optical flow measurements, visible points in 3D space can have their distance estimated relative to the quadrotor. By projecting a 3D cylinder in the direction of travel onto the camera plane, the system can perform reactive obstacle avoidance by steering the cylinder (direction of travel)

to a point with minimal interference. This system is intended to augment a point to point navigation system such that the quadrotor responds to fine obstacle that may have otherwise not been detected.

Contents

List of Figures

Chapter 1

Introduction

As unmanned and autonomous aerial vehicles (UAV) are becoming common place in the world they must be capable of manoeuvring around potential obstacles. UAVs as designed do not require a human pilot. UAVs are used for both commercial and recreational purposes.

1.1 What is a Quadrotor UAV?

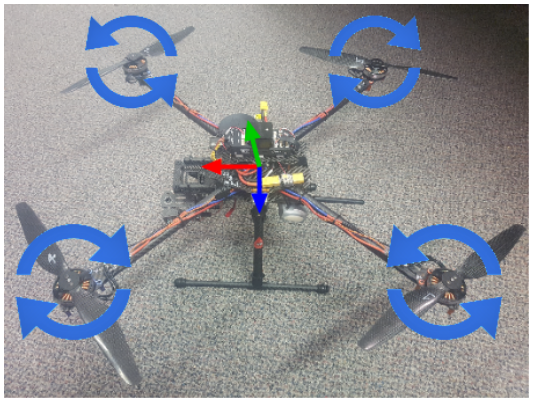

Figure 1.1: A quadrotor used at the Australian National University. Rotor speed differences affect the attitude and speed of the quadrotor.

A quadrotor is a form of UAV that is designed for accurate 3D motion. Unlike helicopters, a quadrotor consist of 4 individual fixed-pitch rotors. This simplifies the complexity involved with helicopters that must have a rotor pitch collective mechanism to control attitude. The removal of rotor collective systems makes quadrotors cheaper than the equivalent helicopter as it reduces the need for specialised and expensive mechanical parts, while also reducing maintenance costs as the only moving parts are the 4 motors.

The basic design of a quadrotor consists of a diagonal pair of rotors that spin in the opposite direction of the other diagonal pair (see Figure 1.1). Controlling the speed on each rotor individually will create torque which is used adjust the attitude of the vehicle, while a uniform increase/decrease of speed of all the motors will modify the thrust.

Designs of multi-rotor are not limited to 4 rotors. The most commonly used platforms are the quadrotor, the hexarotor (6 rotors), and the octarotor (8 rotors). Quadrotors are generally aimed at carrying low weight payloads such as a single camera. However, hexarotors and octarotors are generally aimed at higher weight payloads such as needed by delivery drones. Generally a largely number of rotors implies a heavier, larger platform with a higher load capacity.

Figure 1.2: DJI S1000 octarotor carrying objects[2]

1.2 What applications are currently feasible for UAVs?

UAVs have been widely used to perform tasks such as photography, inspection, search and rescue, military applications, and more recently delivery. They are currently capable of flying from their origin to a target position, and can navigate some low-density forest and city environments.

Figure 1.3: Australian Bushland[3]

Despite all of these applications, there are many scenarios in which flying robots cannot perform well. Autonomous aerial vehicles had been found to be incapable of flying well in dense and complex environments, such as forests with trees that have many low-lying branches (eg. Australian bushland) or areas with wire-based fences.

An application that is currently infeasible which would provide great benefits, would be to release a swarm of fast-moving quadrotors that can navigate dense forest environments. This would greatly increase the efficiency of Search and Rescue operations due to the vast area being covered quickly, while flying under the canopy of the forest would make finding the lost person easier. This could mean the person missing has a significantly higher chance of survival, than using helicopters which fly above the visually restrictive canopy, or humans walking on the ground which is incapable of covering vast areas quickly. Most current aerial vehicle technologies either cannot or have limited ability to do this.

[2] https://deskgram.net/p/1862707655912817336_2202005788
[3] https://commons.wikimedia.org/wiki/File:Blue_Gum_Forest_Blue_Sky.jpg

1.3 What are the current Obstacle Avoidance Methods of UAVs?

Most methods for navigation of autonomous aerial vehicles are split into 2 different categories: reactive control and path planning.

Reactive control is based on using instantaneous clues about the environment to perform obstacle avoidance. The most common reactive control method uses optical flow on a monocular camera, which is the measure of movement as perceived by the 2D camera frame. Optical flow reactive control typically uses the net magnitude and direction of optical flow to steer the vehicle Eresen et al. [2012]; Conroy et al. [2009]; Peszor et al. [2017]. Optical flow has also been used many times to perform corridor centring Conroy et al. [2009]; Eresen et al. [2012]; Zingg et al. [2010], estimate time to contact McCarthy et al. [2008], stabilisation and landing over a textured surface Herisse et al. [2012]; Serra et al. [2016]; Ho et al. [2018], and motion estimation Grabe et al. [2012]; Herisse et al. [2012]; Serra et al. [2016]; Honegger et al. [2013].

Path planning is a predictive control method that requires the estimation of the structure of the environment to plot a course to a target. The most common method for estimating the structure of the environment is using Simultaneous Localization and Mapping (SLAM). This is typically done using a suite of sensors such as RGB-D depth cameras Valenti et al. [2014]; Huang et al. [2017], stereo depth camera Fraundorfer et al. [2012]; Schauwecker and Zell [2013]; Schaub et al. [2017], LIDAR Achtelik et al. [2009]; Alpen et al. [2010], or monocular cameras Weiss et al. [2011]; Fu et al. [2014]; Forster et al. [2013]; Blösch et al. [2010]; Fan et al. [2018]. Estimation time of the structure of the 3D environment however changes in speed with SLAM, dependant on the methods and scene density the SLAM method estimates.

1.4 Contributions of this Thesis

It was found out that reactive control algorithms that use optical flow, steer the vehicle directly using a combination of magnitude and direction of the optical flow. However with modern computational systems such as the NVIDIA Tegra TX2, on-board control using optical flow can be expanded in capability, particular with access to both a CPU and GPU. A few years ago this was not possible on-board micro multi-rotor platform.

Limited research has been done into determining whether the magnitude and direction of optical flow on a point on the 2D image plane is actually representing a threat to the motion of the vehicle. The research gap has been explored in this thesis. This thesis describes the design of a quadrotor system that is capable of implementing a modern optical flow algorithm Adarve and Mahony [2016] that is dense and runs on the latest mobilised computational hardware to perform visual guidance.

The contributions of this thesis can be summarised as follows:

1. Design of a quadrotor system for utility aerial robotic experiments using off-the-shelf components.
2. Software architecture and on-board computational hardware for utility aerial robotics experimental work.
3. Sensor fusion for quadrotor state estimation.

4. Implementation of image based hover and flight control.
5. Propose a novel new obstacle avoidance algorithm for forward looking reactive control using optical flow.

The research for this thesis produced a conference paper published to IROS 2018, called Vision Based Forward Sensitive Reactive Control for a Quadrotor VTOL Stevens and Mahony [2018].

Chapter 2

Literature Review

Autonomous control of micro-aerial vehicles (MAV), specifically the quadrotor platform, has had significant research and is being improved upon over time. Depending on the sensors and computational equipment available, many state estimators and control methods have been developed. Many algorithms that combine multiple sensors have been used to estimate the state of the quadrotor, which includes attitude R, angular rate Ω, position ξ, velocity v, and acceleration a.

Due to the level of sophistication of quadrotors, they are designed to be partly autonomous as all 4 motors must be controlled to achieve the desired input, where it would be too difficult for most, if not all, humans to achieve. This means manual control is done by using human-set attitude or attitude rates, rather than controlling motor states directly.

2.1 History of Quadrotors

The first known manned quadrotor built was the 'Breguet-Richet Gyroplane No. 1', built in 1907 [Leishman, 2002]. The quadrotor has 4 rotors with 8 different lifting surfaces per rotor, leading to a total of 32 lifting surfaces. The Gyroplane No.1 was reported to have lifted a pilot up to 1.5m high for a short period of time, using a single 40hp (29.8kW) engine to power the rotors with an on-board weight about 578kg. However, there was no mechanism to control the vehicle, so it was tethered to the ground.

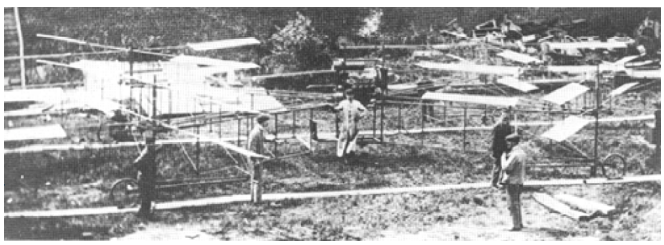

Figure 2.1: Breguet-Richet Gyroplane No. 1, the first manned quadrotor [Leishman, 2002]

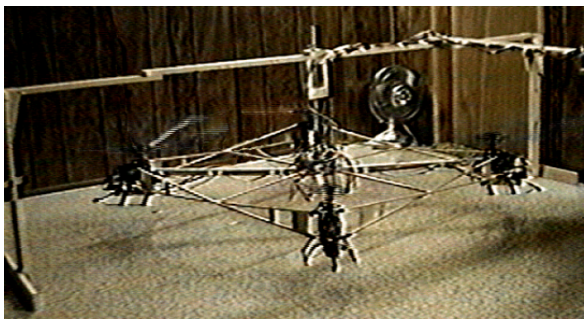

Figure 2.2: Borenstein Hoverbot [Borenstein, 1992]

Figure 2.3: Stanford Mesicopter [Kroo et al., 2000]

The first micro quadrotor was the 'Borenstein Hoverbot' [Borenstein, 1992; Pounds et al., 2007]. Micro quadrotors, while having no official definition, are typically less than 2m in diameter and under 10kg. The 'Borenstein Hoverbot' consisted of 4 hobby helicopters connected by the tail, and controlled via changing rotor thrust and rotor pitch angles. Following this was the 'Roswell Flyer' and 'HMX-4' which later to become the 'Draganflyer', and then the Stanford 'Mesicopter' by the late 1990s which weighed in the order of a gram coming under the unofficial definition of nano-quadrotor [Kroo et al., 2000; Pounds et al., 2007].

The concept for an autonomous quadrotor, also known as a 'X4-Flyer', was first conceived in a papers submitted in 2002 [Hamel et al., 2002; Altug et al., 2002]. The quadrotor platform allows for the detachment of the motor dynamics from the body dynamics, unlike other Vertical Take-off and Landing (VTOL) vehicles such as helicopters [Hamel et al., 2002]. The 4-rotor design allows for omni-directional travel, however they are mostly aimed at low speed or stationary flight [Hamel et al., 2002; Altug et al., 2002].

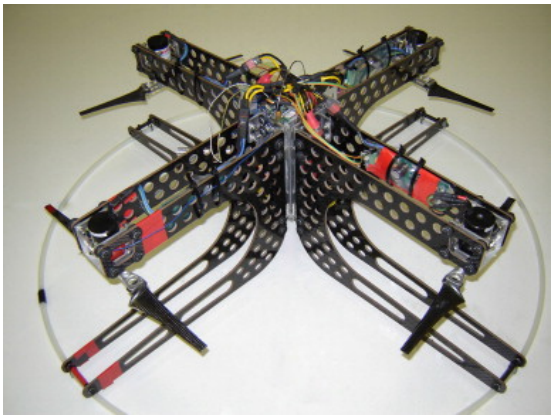

Figure 2.4: Australian National University X-4 Flyer [Pounds et al., 2010, 2007]

Figure 2.5: DJI Spark, general public quadrotor for taking photos or having a fun time

Figure 2.6: DJI Inspire 2, quadrotor for professional photography and film-making

Many commercial drones are available for purchase today. The 2 most well known commercial drone companies are DJI[1] and Parrot[2], which are aimed at the general public and professional photographers.

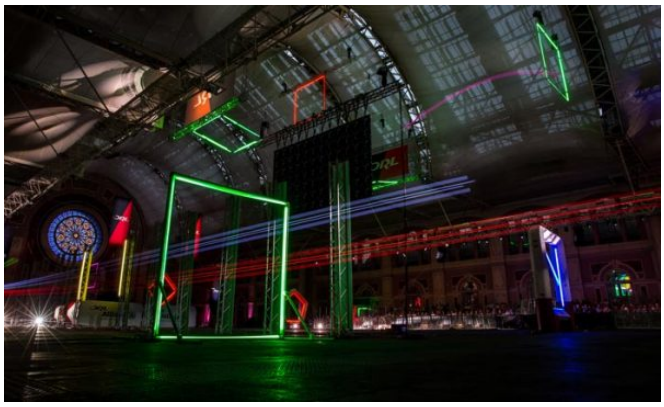

Figure 2.7: The common drone racing environment consists of illuminated rings that the pilots must follow through in a given sequence[4]

Figure 2.8: Parrot Bebop 2 FPV drone

One of the latest trends in quadrotors is first-person view (FPV) drone racing. Racing drones are outfitted with powerful motors and a camera. Pilots can see the environment from the output of the camera on the racing drone using virtual reality (VR) goggles connected by a radio link. The pilot then races against other pilots either via time-trials or having a few pilots fly at once.

[1] DJI home website: https://www.dji.com/
[2] Parrot home website: https://www.parrot.com/global/
[4] https://www.factor-tech.com/feature/drone-racing-zips-glides-and-zooms-into-the-mainstream/

Figure 2.9: Skydio R1 autonomous quadrotor

One of the latest innovations in commercial drone technology is the Skydio R1[5]. The quadrotor released for commerical buyers in 2018 uses Simulataneous Localisation and Mapping (SLAM) to build a local map of the environment. This is done using 13 cameras strategically placed to see the whole environment. Then using Deep Learning, it detects the person who it is set to follow and builds a path through the environment in order to follow the person while avoiding obstacles around it. It is primarily aimed at video photography of people doing activities in the environment.

2.2 Control and State Estimation of Quadrotors

Control of the quadrotor platform can be achieved by varying the rotors' speeds. This control is a function of both rotor speed and pitch [Hamel et al., 2002; Altug et al., 2002]. By varying different rotors' speeds, the thrust T and torque τ of the quadrotor will vary. The thrust and torque combined from each rotor will cause the vehicle to rotate and accelerate [Hamel et al., 2002; Altug et al., 2002]. To ensure that the net torque of the quadrotor can be controlled, each diagonal pair of rotors should spin in the opposite direction [Hamel et al., 2002; Altug et al., 2002]. Roll ϕ can be controlled by increasing or decreasing the rotor speeds of the left and/or right rotors. Pitch θ can be controlled by increasing or decreasing the rotor speeds of the forward and/or rear rotors. Yaw ψ can be controlled by varying the net torque of the rotors [Hamel et al., 2002; Altug et al., 2002].

Estimation of the current state of the quadrotor is undertaken in few different ways. Most modern quadrotors and flight controller have a sensor suite on board that consists of an accelerometer, gyroscope, magnetometer and barometer which constitutes the Inertial Measurement Unit (IMU). Using the IMU and software filters (eg. Extended Kalman Filter), the rotation, acceleration (translational and rotational) and altitude of the quadrotor can be estimated [Bloesch et al., 2015, 2017; Zhang et al., 2016a,b; Lim et al., 2012; Bangura et al., 2014].

Most quadrotors that are commonly used for commercial or hobby purposes also are equipped with a global position estimation unit (eg. GPS, GLONASS, Galileo), which estimates the latitude and longitude (in spherical coordinates) of the quadrotor in the world via a complex satellite network. This can be used to estimate position and velocity (via the derivative of position or doppler effect) in local coordinates. This allows for position and velocity control of the quadrotor to be achieved in outdoor environments. Local positioning systems, such as VICON [6], can also estimate the position of the quadrotor inside a predefined domain. These local positioning systems are commonly used in indoor environment, for the purpose of

[5] Skydio R1 information at: https://www.skydio.com/
[6] https://www.vicon.com/

testing algorithms that require position or velocity estimation. These position and velocity estimates are often incorporated into the filters used by the IMU, to estimate the 6-DOF (degrees of freedom) state of the quadrotor [Bangura et al., 2015; Brescianini et al., 2013; Weiss et al., 2011].

However, other methods are also used to estimate the 6-DOF state of the quadrotor. This can also be done using visual methods such as Simultaneous Localisation and Mapping (SLAM) or Visual-Inertial Odometry (VIO). Optical flow can also be used to estimate velocity and position, which is described later.

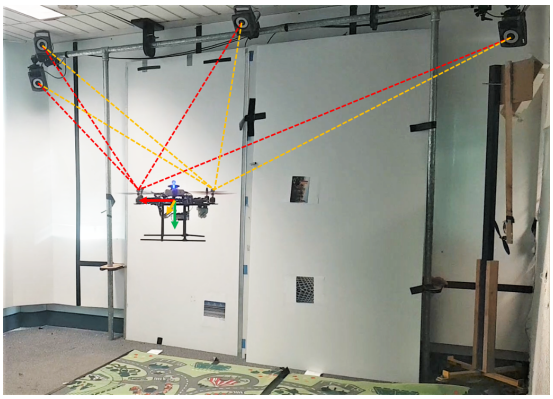

Figure 2.10: VICON local positioning system. The system is finding reflective markers on the quadrotor using infra-red radiation. It is then using the distances measured to get position estimates of the markers. From the combination of markers with known positions on the quadrotor, the position and attitude of the quadrotor can be determined.

One of the most common visual methods for the autonomous control of vehicles, including quadrotors, is using Simultaneous Localisation and Mapping (SLAM). SLAM is the process of building a map of the environment, usually by tracking features in the environment, while estimating its 6-DOF pose inside the map being built. This has been extensively done in quadrotors. The common sensors used to perform SLAM on quadrotors are monocular cameras [Weiss et al., 2011; Fu et al., 2014; Forster et al., 2013; Blösch et al., 2010], rgb-d cameras [Valenti et al., 2014; Huang et al., 2017], LIDAR (laser-based distance sensor) [Achtelik et al., 2009; Alpen et al., 2010], and stereo cameras [Fraundorfer et al., 2012; Schauwecker and Zell, 2013; Schaub et al., 2017].

Visual-Inertial Odemetry (VIO) is the fusion of Inertial Measurement Unit (IMU) data (accelerometer, gyroscope, magnetometer) with visual estimates, to estimate the rotation and position of the vehicle [Bloesch et al., 2015, 2017; Weiss et al., 2012; Mohta et al., 2018]. VIO is similar to SLAM that it creates a map to track its position and velocity, however it removes parts of the map it can no longer see or track. An example of a recent project is the Robust Visual-Inertial Odemetry (ROVIO) open-source project by a team at ETH Zurich [7]. This project uses an Extended Kalman Filter (EKF) with visual feature tracking on a camera to estimate the rotation, position and velocity of a UAV in real-time [Bloesch et al., 2015, 2017].

[7] https://github.com/ethz-asl/rovio

2.3 Optical Flow and its Characteristics

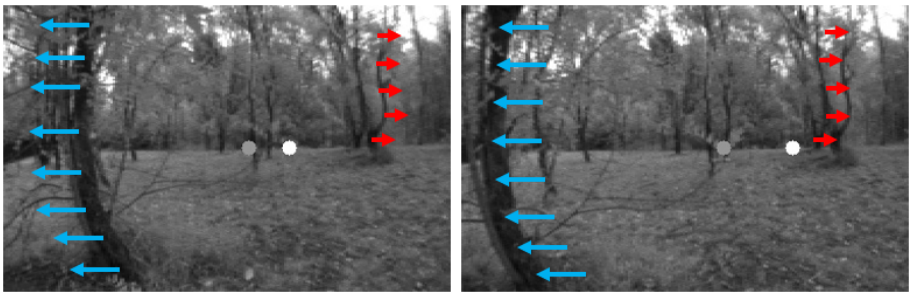

Figure 2.11: Optical flow estimation example. The difference in position of points between the left and right image is optical flow.

Optical flow is the measure of the movement of image gradients and intensities tangentially across the image plane [Lucas et al., 1981; Adarve and Mahony, 2016; Horn and Schunck, 1981]. The motion viewed from the camera frame is a representation of motion in the world that the camera sees, and can be used to estimate the motion of the camera itself.

One of the most commonly used methods for calculating optical flow is called the Lucas-Kanarde method [Lucas et al., 1981]. The Lucas-Kanarde method is open-source and available easily via most computer vision software packages such as OpenCV and MATLAB. The Lucas-Kanarde method was originally designed for the matching of stereo camera images for calculating depth, by calculating the relative disparity of image intensity gradients (via pixels) on the image plane [Lucas et al., 1981]. The same algorithm was adapted to match between 2 images in a sequence, to compute optical flow. This method is designed to compute optical flow over a very small (eg. 3x3 pixels) local area, however a large area can be computed by smoothing the image (low-pass filtering) and skipping pixels (called the 'binning' of pixels) effectively allowing distant pixels to act like they are closer [Lucas et al., 1981; Adarve and Mahony, 2016; Bouguet, 2001]. This is known as the 'Pyramidal' structure [Adarve and Mahony, 2016; Bouguet, 2001; Zingg et al., 2010].

The Horn Schunck method is another method for calculating optical flow, which was developed around the same time as the Lucas-Kanarde method [Horn and Schunck, 1981]. This method is designed around satisfying brightness constancy criteria, which is based on the shifting of image intensities E across the image plane.

Recent developments for optical flow algorithms are either focused on speed or accuracy of the computation. Current optical flow algorithms that are aimed for speed utilise computing hardware such as GPU and FPGA systems, which have allowed for optical flow algorithms to run in the hundreds of hertz range [Adarve and Mahony, 2016]. For example, [Adarve and Mahony, 2016], which is used during our experiments, is an example of a temporal-based optical flow algorithm that can preform optical flow at VGA resolution at around 800Hz. This was performed on an Intel i7-4790K processor (Quad core, 4.4GHz) and NVIDIA GTX 780 GPU (2304 CUDA cores, 288 GB/s of memory bandwidth) [Adarve and Mahony, 2016]. Current optical flow algorithms that aimed for accuracy, typically attempt to use a deep learning method to calculate optical flow, or combine parts of both standard and deep learning opti-

cal flow methods [Xu et al., 2017; Ren et al., 2017; Ilg et al., 2017; Dosovitskiy et al., 2015; Ranjan and Black, 2017].

Since optical flow is the measure of motion of the projection of the environment onto the camera plane, it is possible to estimate the motion of the camera itself. While most optical flow methods uses the magnitude and direction to estimate motion, it is also possible to do this using only the direction [Briod et al., 2016].

The motion of the camera, and position by integration of velocity, can be estimated from using the optical flow perceived by the camera and can be mapped onto a unit sphere or via using a spherical camera [Grabe et al., 2012; Herisse et al., 2012; Serra et al., 2016; Honegger et al., 2013]. Mapping the optical flow onto the unit sphere perceived allows for a measure of motion in all of $\mathbb{R}^3$, where the the motion in the $z = \mathbf{e}_3$ direction of the camera is measured from the divergence/convergence of the optical flow. The motion estimates scale with distance to the surface of the object, which requires rectification to get velocity [Grabe et al., 2012; Herisse et al., 2012; Serra et al., 2016; Honegger et al., 2013].

An open-sourced project, called the PX4Flow, is a device that computes Lucas-Kanarde optical flow on a camera with an embedded micro controller, which has an attached gyroscope and sonar [Honegger et al., 2013]. This sensor is shows to be able to calculate velocity estimates, which by integration position estimates. It also uses the gyroscope to estimate and remove rotational optical flow, as velocity estimates require only the translational component of optical flow to be accurate. It also takes into account the scaling of optical flow from the distance from the surface, using the sonar for distance measurement [Honegger et al., 2013].

2.4 Visual Control of Autonomous Aerial Vehicles

Reactive and Path Planning are the 2 primary control methods used on autonomous vehicles. Reactive control relates to the instantaneous reaction to visual cue, while path planning is based on using the known scene to estimate a path to follow.

Reactive control is typically achieved via using a visual input such as one of the few types of cameras (monocular, RGB-D, stereo, etc). The most common method for reactive control is based on using optical flow on a monocular camera, due to its lightweight and relative ease for computation [Zingg et al., 2010; Coombs et al., 1998; McGuire et al., 2017; Peng et al., 2016; Srinivasan et al., 1999].

Path planning requires that the structure of the environment can be determined, and a target destination is known. A path is often built that fits the vehicle's constraints, so that the vehicle navigates to the target. If a map is available, the path is built to navigate around the objects it can see. SLAM is used for path planning, as it naturally builds a map of the environment in which the quadrotor is moving [Weiss et al., 2011].

The focus of this thesis is for using optical flow for reactive control as opposed to full navigation and path planning of a quadrotor.

The optical flow method by Adarve and Mahony [2016] is used to perform the obstacle avoidance and visual servoing experiments shown in this thesis. Optical flow based control of robotic vehicles is shown to be a classical subject in ground-based and flying robotics [Coombs et al., 1998; McGuire et al., 2017; Peng et al., 2016; Srinivasan et al., 1999].

Bio-mimetic literature demonstrates that many animals, insects, and even humans rely on optical flow for low level control of motion [Srinivasan et al., 1996; Bhagavatula et al., 2011; Warren et al., 2001; Srinivasan et al., 1999]. Honey bees and budgerigars were shown to navigate through narrow passages by averaging the optic flow it experiences between both eyes [Srinivasan et al., 1996; Bhagavatula et al., 2011; Srinivasan et al., 1999].

Corridor centring is one of the most prominent uses for optical flow on quadrotors for indoor and outdoor environments [Conroy et al., 2009; Eresen et al., 2012; Zingg et al., 2010; Lai et al., 2018]. An early example of optical flow control for corridor centring is where a ground robot calculated optical flow from a camera which views both walls in a corridor, and steers the robot to make the optical flow velocities equal [Srinivasan et al., 1999]. They controlled the position of the robot by balancing the flow experienced. This was expanded to corridor centring by estimating the distance from the robot to wall using optical flow from its known velocity [Zingg et al., 2010]. The current limitations of corridor centring using optical flow is when the environment is complex, such as in forests.

Implementation of optical flow based control of quadrotors has been undertaken for hovering over a target [Herisse et al., 2012; Serra et al., 2016]. In the experiments by [Herisse et al., 2012], they calculated the inertial average optical flow w of the projection of a camera onto a unit sphere to achieve motion estimates in $\mathbb{R}^3$ (includes field divergence and convergence) [Herisse et al., 2012]. From this, they were capable of hovering and landing over a moving textured target. This was done using a face-down camera transmitting the camera feed to a ground-station which is computing Lucas-Kanarde optical flow, then transmitting commands back to the quadrotor [Herisse et al., 2012]. Hovering experiments have also been used to estimate velocity from a face-down camera, when you know the distance from the ground [Grabe et al., 2012; Herisse et al., 2012; Serra et al., 2016; Honegger et al., 2013]. A similar experiment was performed more recently using on-board processing computer (Intel Atom Z530 1.6GHz), which computed Lucas-Kanarde optical flow at 10Hz [Serra et al., 2016]. Position estimation was done using landmarks [Serra et al., 2016].

Divergence of optical flow has been used to perform visual docking of a quadrotor [McCarthy et al., 2008]. In this experiment, the Lucas-Kanarde optical flow method was used [McCarthy et al., 2008]. This is done via acquiring an estimate of *time-to-contact* τ to a surface via measuring the optical flow field divergence, while taking into account rotations of the camera and surface [McCarthy et al., 2008].

Direct control of a quadrotor or micro aerial vehicles (MAV) for obstacle avoidance during flight through unknown environments has been implemented a few times [Eresen et al., 2012; Conroy et al., 2009; Peszor et al., 2017]. These methods primarily focused on using the magnitude and direction of optical flow to directly control a quadrotor.

Some methods for the visual autonomous control of quadrotors can often mix different visual sensors and methods. [Fraundorfer et al., 2012] mixes a face-down optical flow camera with a forward facing stereo camera. In this method, the stereo camera is used to do SLAM, while the optical flow is used to estimate velocity [Fraundorfer et al., 2012]. [Achtelik et al., 2009] combines a stereo camera and laser range finder to perform SLAM.

2.5 Summary

Since the inception of autonomous quadrotors and multi-rotor variants, they need to be able to get to their commanded location without colliding with obstacles [Hamel et al., 2002; Altug et al., 2002]. Potential obstacles can be detected using a range of sensors, including cameras and rangefinders. Obstacles can be avoided by planning a path using a map or reactively responding to detected obstacles using instantaneous sensor data.

Path planning requires knowledge of the environment to map a path. The map can be generated before the flight or by creating a map as the vehicle is moving via methods such as Simultaneous Localisation and Mapping (SLAM) [Weiss et al., 2011; Fu et al., 2014; Forster et al., 2013; Blösch et al., 2010; Valenti et al., 2014; Huang et al., 2017; Achtelik et al., 2009; Alpen et al., 2010; Fraundorfer et al., 2012; Schauwecker and Zell, 2013; Schaub et al., 2017] and Visual Inertial Odometry (VIO) [Bloesch et al., 2015, 2017; Weiss et al., 2012; Mohta et al., 2018].

Reactive obstacle avoidance, which is a focus in this thesis, is the near-instantaneous reaction of a mobile autonomous system to the detection of obstacles. Reactive control methods are generally simpler calculations compared to creating a map and planning a path using it such as in SLAM and VIO, allowing for faster responses to detected obstacles.

One of the most common methods for detecting obstacles used in reactive control, is optical flow. Optical flow is the measure of the movement of image gradients and intensities tangentially across the image plane [Lucas et al., 1981; Adarve and Mahony, 2016; Horn and Schunck, 1981]. It is suggested in bio-mimetic literature that many animals, insects, and even humans rely on optical flow for low level control of motion [Srinivasan et al., 1996; Bhagavatula et al., 2011; Warren et al., 2001; Srinivasan et al., 1999].

The most common methods of using optical flow for control is corridor centring and obstacle avoidance in flight via balancing optical flow over the image plane [Conroy et al., 2009; Eresen et al., 2012; Zingg et al., 2010; Lai et al., 2018; Eresen et al., 2012; Peszor et al., 2017], hovering over a target using the divergence and averaged optical flow [Herisse et al., 2012; Serra et al., 2016], and docking of a quadrotor on to a target using the divergence of the optical flow [McCarthy et al., 2008].

Little research has been done into using optical flow for reactive obstacle avoidance in complex and dense environments, such as forests or areas with wires or cables. Most optical flow based control experiments focused on environments with little to no complex obstacles like trees, including indoors and open areas.

Not all optical flow measurements represent an obstacle in the path of the quadrotor, while some small obstacles such as wires produce little relative optical flow that classical reactive control algorithms can use effectively. An example could be when a quadrotor is flying along a wire fence, with trees on the right. The trees are likely to have a much more significant optical flow presence in the image compared to the wires, and balancing optical flow will tell the quadrotor to fly towards the wire fence even though the trees may not be in the way.

This thesis has a focus on detecting and reactively avoiding obstacles using dense optical flow that are actually a threat to a quadrotor by fusing sensor data with the optical flow measurements.

Chapter 3

System Avionics, Hardware, and Computational Architecture

This chapter presents a quadrotor platform that was designed for use during this thesis. It will explain the system architecture of both the hardware and software systems. Figure 3.1 shows the fully constructed quadrotor.

Figure 3.1: Experimental Quadrotor

From Figure 3.1, a Pixhawk 2.1 flight controller is mounted on the top of the quadrotor with a HERE GNSS attached on top of the the Pixhawk 2.1. A NVIDIA Jetson TX2 and a Matrix-Vision mvBlueFOX-200w camera (with 3-axis gimbal) is mounted underneath, with the camera at the front of the quadrotor. Figure 3.2 is a view of the undercarriage close-up.

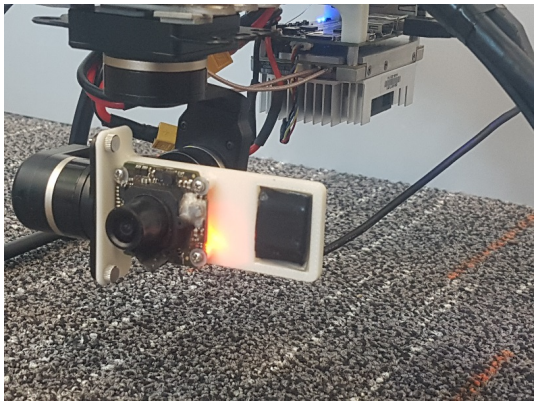

Figure 3.2: Quadrotor Undercarriage. Contains the mvBlueFOX-200w camera mounted to a 3-axis gimbal and the NVIDIA Tegra TX2 in the background

3.1 System Requirements

A quadrotor platform was designed to perform vision-based experiments with a monocular camera in Australian environments. This came with the following requirements:

1. The quadrotor must have an autonomous flight mode for the experiments, and a manual flight mode (via remote controller) in case the autonomous mode fails.

2. The quadrotor must have on-board compute capability, with vision processing ability. Remote computational systems have limited bandwidth, higher latency, and can become unusable when the wireless signal is lost.

3. The quadrotor must be less than 2kg, to meet the Australian Government Civil Aviation Safety Authority (CASA) under 2kg commercial regulation[1].

4. The quadrotor must must primarily use off-the-shelf parts for repeatability and reducing overall cost.

3.2 System Overview

This section presents the system overview of the quadrotor architecture. The overview in Figure 3.3 shows the design basis required to meet the system requirements.

[1]CASA under 2kg commercial quadrotor rulings: https://www.casa.gov.au/standard-page/commercial-unmanned-flight-remotely-piloted-aircraft-under-2kg

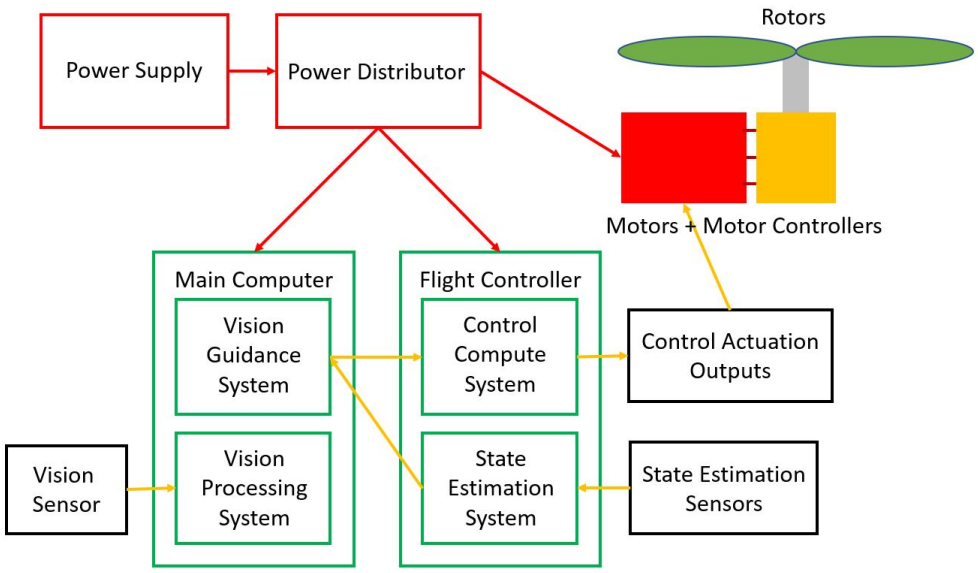

Figure 3.3: System overview of required architecture to meet system requirements

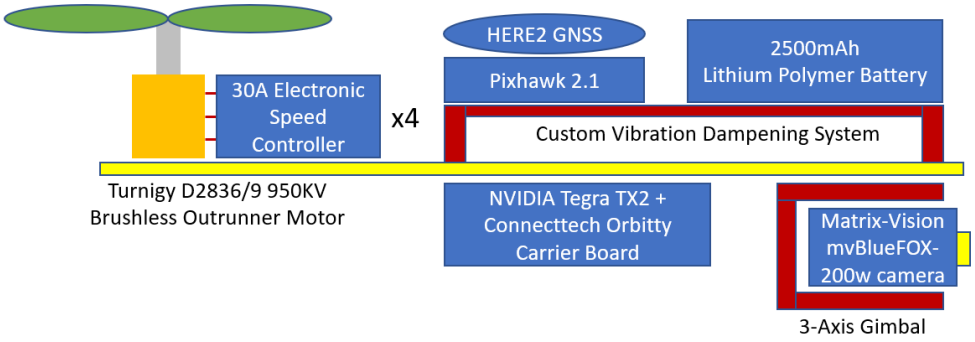

Figure 3.4: Hardware Architecture

As it is shown in Figure 3.3, 2 computational devices are required which are both mounted on-board the airframe. A flight controller is required to take data from state estimation sensors and a control command, turning these into control signal outputs used by the motors. The main computer must be used to take in state data from the flight controller and data from a vision sensor, while providing control commands to the flight controller.

3.3 Computational Architecture

The 2 different computational devices as described in the system overview are the Pixhawk 2.1 (PX4) as the flight controller and the NVIDIA Tegra TX2 as the main computer.

3.3.1 Pixhawk 2.1 (PX4) Architecture

The Pixhawk 2.1 is the flight controller and computer used to estimate the state of the quadrotor (rotation, velocity, position) and control the airframe via sending signals to the motor controllers. It includes an ARM STM32F427 microprocessor and heated Inertial Measurement Unit (IMU). The IMU includes an accelerometer, gyroscope, magnetometer, and barometer, which are used to estimate the state of the quadrotor. Additionally, a HERE2 GNSS[2] is another state estimation sensor used which uses the satellite networks (GPS, BeiDou, GLONASS, Galileo) to give a global position estimation.

This device was chosen largely based on the firmware it uses. The firmware used on the Pixhawk 2.1 is the open-source px4 development code[3][4]. This firmware is under a 3-clause BSD license[5]. It allows users to subscribe to data being constantly transmitted on the system via UORB, which allows modular development of all programs. This modular design allows for building programs that do not interfere with each other (except via CPU usage).

Figure 3.5: Pixhawk 2.1 with HERE2 GNSS mounted on top

The micro-controller estimates the state of the quadrotor which includes position, attitude, and attitude rates via an Extended Kalman Filter, IMU and GPS. It then can turn manual or autonomous control inputs/setpoints to motor commands for flight.

The Inertial Measurement Unit (IMU) consists of inertial, magnetic, and air pressure sensors for the PX4 system to use to estimate state. The 2 inertial sensors are the accelerometer and gyroscope which measure linear and angular acceleration respectively. The magnetic sensor called a magnetometer (or compass) is used to estimate the rotation of the quadrotor in respect to the Earth's magnetic field. The air pressure sensor called the barometer is used to estimate height from sea level via measuring the local air pressure, in which air pressure decreases as altitude increases.

The temperature compensated and physically dampened IMU on the Pixhawk 2.1 was one of the primary reasons it was chosen. In most flight controllers, the inbuilt IMUs are prone to having temperature affect the performance and calibration of the IMU allowing for drift in flight. Temperature compensating this by setting a constant temperature (60^oC), means that the calibration and noise is constant in flight meaning the calibration values do not require change during flight for keep maximum performance. Similarly, having a physically dampened IMU means that the measurements from the IMU are more trustworthy and less prone to noise caused by the rotors causing vibrations during flight.

[2]HERE2 GNSS available at: http://pixhawkstore.com.au/here-2-gnss/
[3]PX4 Developer Instructions: https://dev.px4.io/en/
[4]PX4 Developer Code: https://github.com/PX4/Firmware
[5]3-Clause BSD License: https://opensource.org/licenses/BSD-3-Clause

3.3.2 NVIDIA Tegra TX2 Architecture

The NVIDIA Jetson TX2 is the main computational device used to calculate control features on the quadrotor. The TX2 is used to perform vision-based guidance calculations used to control the velocity of the quadrotor. This is a credit-card sized board that can be mounted to different I/O (Input/Output) boards.

The specifications of the NVIDIA Tegra TX2 are:

- Central Processing Unit (CPU): HMP Dual Denver 2/2 MB L2 + Quad ARM A57/2 MB L2
- Graphics Processing Unit (GPU): NVIDIA Pascal with 256 CUDA cores
- RAM: 8 GB 128 bit LPDDR4 59.7 GB/s
- Storage: 32 GB eMMC, SDIO, SATA
- I/O: CAN, UART, SPI, I2C, I2S, GPIO

The NVIDIA Jetson TX2's firmware is the NVIDIA JetPack SDK which is a modification of Ubuntu 16.04[6]. This software allows access to the GPU of the TX2 via the inbuilt CUDA drivers, which is commonly used for developing programs for NVIDIA GPUs. Robot Operating System (ROS)[7] is capable of running on the TX2's software allowing for modular development of programs, which can is used on quadrotor platform.

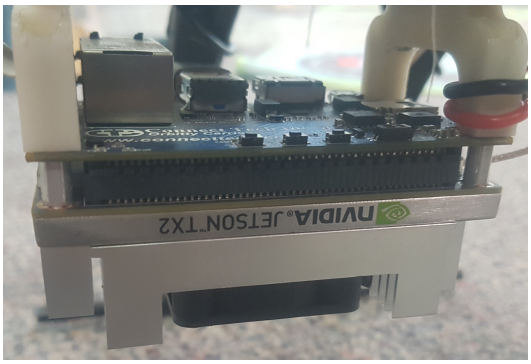

Figure 3.6: NVIDIA Tegra TX2 connected to the compact ConnectTech Orbitty Carrier Board

Since the main (default) development I/O board for the TX2 is large, another I/O carrier board is used. The I/O board used is the ConnectTech Orbitty Carrier Board[8] which matches the credit-card shape of the TX2, allowing for it to be mounted to the quadrotor platform with ease.

The design choice for using NVIDIA Tegra TX2 was based on the following reasons. Its low power consumption, but strong compute ability with its CPU and GPU allows for a good and compute heavy algorithm to run on the device at a desirable rate. The dense optical flow algorithm used can operate at 120Hz at half-VGA (376x240) resolution. As such, the control calculations works 50-60Hz allowing for rapid control methods. The small size of the TX2 with the carrier board attached means that it can fit onto a drone-sized platform without significantly affecting performance compared to similarly powerful devices. Having the

[6]NVIDIA Tegra TX2 operating system installer can be found through: https://developer.nvidia.com/embedded/jetpack

[7]Robot Operating System (ROS) website: http://www.ros.org/

[8]ConnectTech Orbitty Carrier Board information can be found at: http://connecttech.com/product/orbitty-carrier-for-nvidia-jetson-tx2-tx1/

commonly available Ubuntu 16.04 installed on the device also allows programs developed on Linux/Ubuntu systems should simply work on the device without much, if any, modifications.

3.4 Communications

The communications methods between all the on-board components is shown in Figure 3.7. A detailed description of Figure 3.7 is explained after.

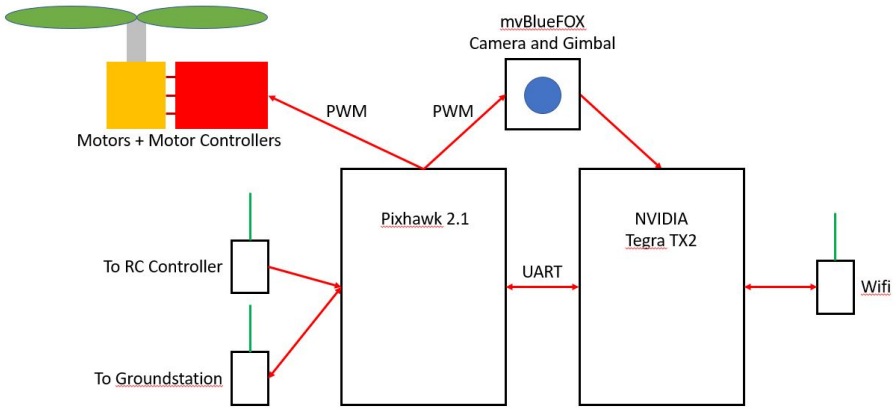

Figure 3.7: Complete Quadrotor Architecture and Communications

1. TX2 to Pixhawk: UART/Serial link at 921600 baud, transferring data to and from at 120Hz.
2. TX2 to Laptop: A serial connection (via ssh) can be established with the TX2 and a laptop are connected to the same wifi router. This is necessary to initiate and debug the TX2 in flight.
3. TX2 to Camera: The camera passes data to the TX2 via USB2.0, while having the ability to set parameters on the camera from the TX2.
4. Pixhawk to Camera Gimbal: The Pixhawk outputs Pulse-width modulated (PWM) signals to the gimbals inputs to give it position commands.
5. Pixhawk to Radio Controller: The Radio controller sends signals to a receiver attached to the Pixhawk, which allows the user to arm, manually control, and set flight modes.
6. Pixhawk to Groundstation: The Pixhawk can communicate to a ground-station via a 57600 baud radio link. The data is transferred between the 2 devices using the common Mavlink protocol, used by most aerial hobby devices.
7. Pixhawk to Motor: The Pixhawk sends PWM signals to the motor controller. This is used by the motor controller to set the average current from the battery to the motor for controlling motor speed and torque.

3.5 Camera and Gimbal

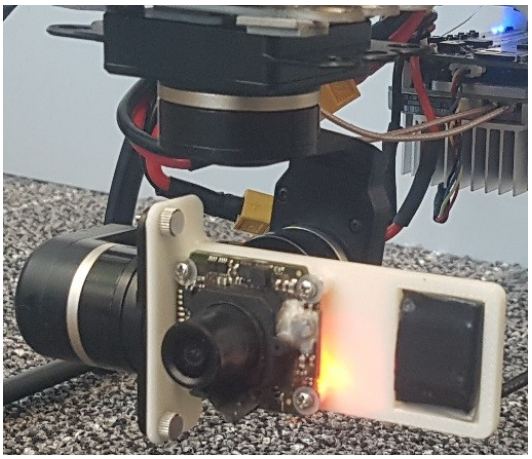

Figure 3.8: Matrix-Vision mvBlueFOX-200w mounted to an off the shelf 3-axis gimbal for motion stabilisation

The vision sensor (camera) used was the Matrix-Vision mvBlueFOX-200w[9] with a native resolution of 752x240, which was commonly used in the research field. This included an IR-cut filter and a 120^o field of view lens. This camera was chosen for multiple reasons. The camera is natively RGB, but it capable of running in grey-scale mode in which the chosen optical flow algorithm uses. This automatic ability from the camera to output grey-scale images saves compute resources on the TX2. Additionally, the camera can halve the resolution in both horizontal and vertical via H/V binning. This allowed the camera to reduce its resolution to 376x240, without consuming TX2 compute space. A resolution of 376x240 provides an advantage as it allows the TX2 to compute optical flow at 120Hz, rather than 70Hz, based on the performance of the TX2. This allowed the quadrotor to estimate the scene and make control decision closer to real-time reducing risk of collision from delayed response.

The 120^o field of view, IR-cut lens was used to get maximum viewing angle which allows for more obstacles being detected and a larger range for control. An IR-cut filter was necessary due to the high IR sensitivity of the Matrix-Vision mvBlueFOX-200w. During experiments, dirt appeared as extremely bright and texture-less with respect to grass and other vegetation. This is due to the dirt absorbing sunlight, heating up, then releasing infra-red radiation as a result of the heat. The IR-cut filter helped in solving this, as the camera then only received the visual spectrum which is sensitive to texture changes. The larger texture detection then helped the optical flow algorithm work, as optical flow algorithms require more texture to get more dense and accurate results.

The gimbal used was a 3-axis off-the-shelf gimbal. The design decision to use a 3-axis gimbal rather than hard-mounting the camera was based on experiments designed to test the chosen optical flow's response. In optical flow, there is a rotational and translational component measured. The translational component gives 3D information of the scene, while the rotational component provides little 3D scene information. If you the know camera calibration matrix and the rotational rate of the camera, you can in theory estimate the rotational component of the optical flow and subtract it from the optical flow to get the translational component.

The dense optical flow algorithm used, which is aimed at estimating optical flow with extreme speed and density, uses filtering principles which requires the flow to be properly constructed after a few frames. The relationship between the measured rotational rates of the

[9]Information on the Matrix-Vision mvBlueFOX-200w can be found at: https://www.matrix-vision.com/USB2.0-industrial-camera-mvbluefox.html

camera and the optical flow algorithm appears as a single pole linear response when experiments were performed to analyse this. Using this linear response, you can estimate the relative rotational rate of the gyroscope measurements compared to the optical flow. Compensating for optical flow induced by the rotation, which was filtered by the proposed optical flow algorithm, proved to be impossible. This meant a 3-axis gimbal had to be used to make sure only the translational component of the optical flow is observed, by constantly keeping the camera pointing straight while eliminating the roll/pitch.

3.6 Hardware vibration dampening system

A customised dampening system is used to filter the vibrations caused by the inertia of the rotors. Without the dampening system, the accelerometer picks up most of the noise caused by the vibrations felt by the quadrotor in flight. This causes attitude estimates and velocity estimates using (4.7) and (4.5) to be inaccurate. The dampening system consists of 2 plates with 4 rubber springs interconnecting the 2 plates. Mounted on the dampening system is the Pixhawk 2.1 and a Battery. With the Pixhawk 2.1 mounted on the dampening system, the accelerometer on its IMU picks up less noise caused by the rotor vibrations. An example of this is shown in Figure 3.9.

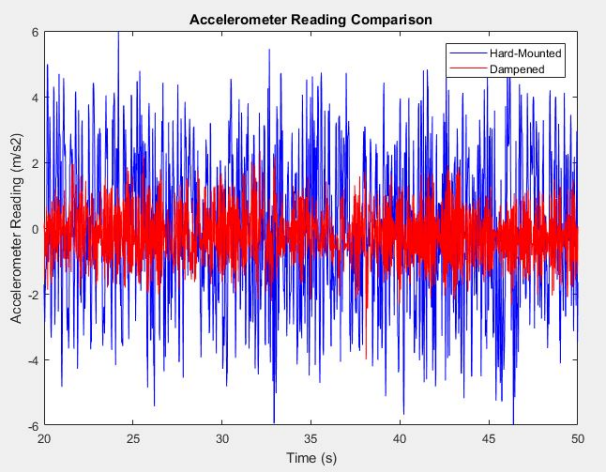

Figure 3.9: Accelerometer noise comparison. The blue curve is the noise observed when the Pixhawk 2.1 is hard-mounted directly to the frame. The red curve is the noise observed with the Pixhawk 2.1 mounted on the dampening system.

The Power Spectral Density of both the hard-mounted and dampened accelerometer is shown in Figures 3.10 and 3.11 respectively.

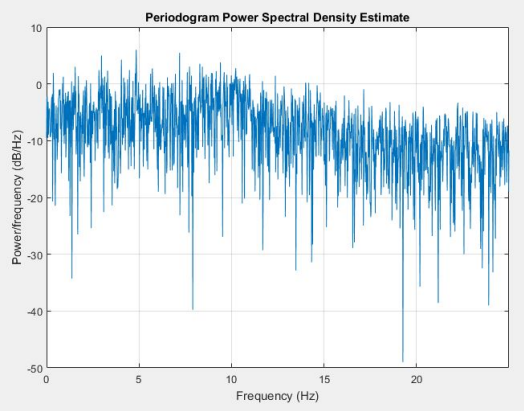

Figure 3.10: Power Spectral Density of the Hard-Mounted IMU

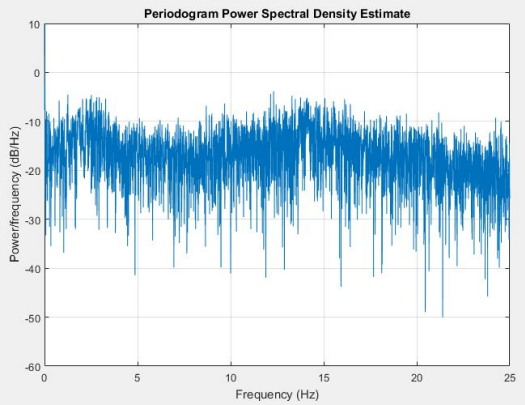

Figure 3.11: Power Spectral Density of the Dampened IMU

It is clear from Figures 3.10 and 3.11 that there is -10dB reduction in noise power caused by the dampening system. This means less energy is being fed from the vibrations caused by the rotors into the Pixhawk 2.1's IMU. This shows that the dampening system reduces accelerometer noise, and could allow for higher attitude and velocity accuracy.

3.7 System components and weights

To meet the Australian Government Civil Aviation Safety Authority (CASA) under 2kg commercial rulings, the quadrotor was designed to be less than 2kg. To meet this system requirements, we used parts to meet this. These parts and their weights are shown in Table 3.1.

Quadrotor Frame	Hobbyking X650F Glass Fiber Quadcopter Frame 550mm[10]	600g
4x motors	Turnigy D2836/9 950KV brushless outrunner motor[11]	70g each
Main Computational Device	NVIDIA Tegra TX2	250g
Flight Controller	Pixhawk 2.1	50g
Camera	Matrix-Vision mvBlueFOX-200w	40g
Gimbal	3-axis Aluminium cased	200g
Battery	Li-Po 2500mAh	350g
Additional Parts	Wiring, motor controllers, mounts	190g

Table 3.1: Table of quadrotor part weights

This system ended up with a total weight of 1.96kg. To meet the total weight budget, cables were routed through the device and cut to the minimal length required. The quadrotor frame had minor modifications to reduce weight, with the hardware dampening system designed to reduce the weight. The battery weight was reduced by using a smaller battery, however it had an effect on total flight time.

[10] https://hobbyking.com/en_us/hobbyking-x650f-glass-fiber-quadcopter-frame-550mm.html
[11] https://hobbyking.com/en_us/turnigy-d2836-9-950kv-brushless-outrunner-motor.html

Chapter 4

Software Architecture, Filtering and Performance Tuning

Software is required to perform any calculations required for controlling any robotic device. Software is used to filter various different signal to be able to form a response that drives control commands. An example is filtering the Inertial Measurement Unit (IMU) readings to estimate state, and use a command input to change the state to meet the command in the most efficient manner. To fly a drone completely autonomously, for example the quadrotor platform designed, all software is executed on-board the vehicle. This is all done on the Pixhawk 2.1 flight controller and the NVIDIA Tegra TX2 computational board.

4.1 Software Architecture

The code structure on the Pixhawk 2.1 is shown in Figure 4.1.

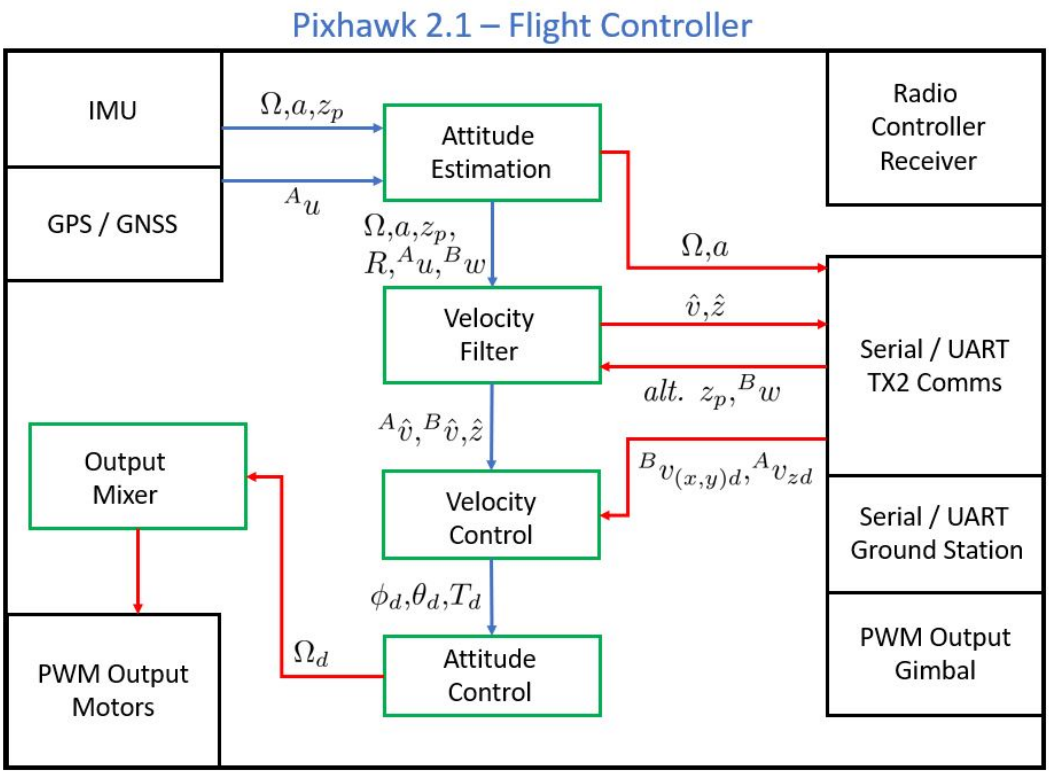

Figure 4.1: Pixhawk 2.1 Hardware and Code Structure. Black boxes are I/O or hardware, green boxes are code components.

The Pixhawk 2.1 has 4 main code structures used to control the quadrotor. The PX4 firmware is designed to modularise code as separate threads on PX4-based system. To make this possible, each module/block can be given a thread priority, while data can be subscribed and published between each module/block via a system called UORB (similar principle to ROS). The Attitude Estimation block includes the code structures used by the default open-source PX4 firmware to calculate attitude and position estimates using the IMU and GPS sensors. The Velocity Filter block calculates velocity estimates using (4.7), or other velocity measurements. The Velocity Control block is used to convert velocity commands to rotations using (4.20). The Output Mixer block is used to control the motors and gimbal Pulse Width Modulation (PWM) outputs, which are converted to power from external motor controllers. Code has also been developed to talk to and from the NVIDIA Tegra TX2 via a UART/Serial link.

The code structure on the NVIDIA Tegra TX2 is shown in Figure 4.2.

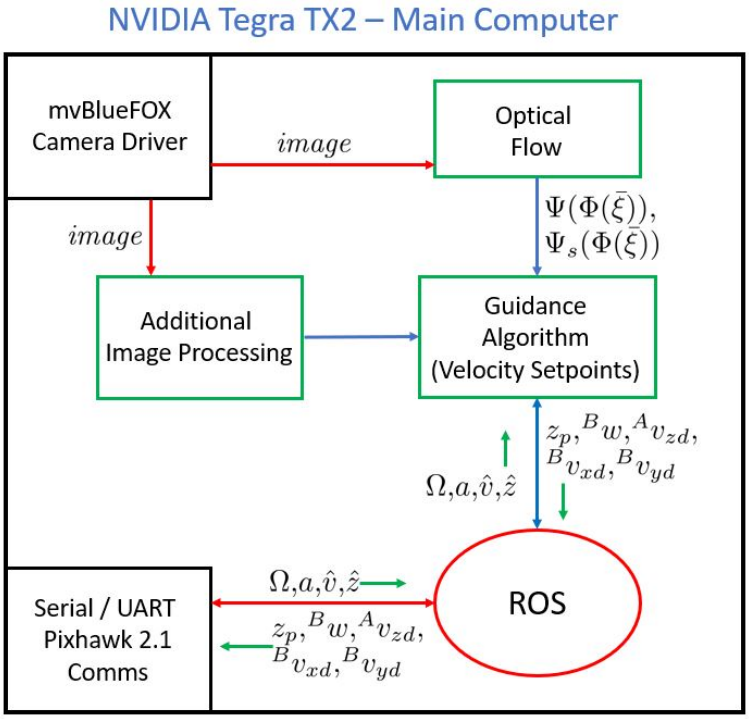

Figure 4.2: NVIDIA Tegra TX2 Hardware and Code Structure. Black boxes are I/O or hardware, green boxes are code components. The red ROS box is the internal ROS messaging system.

The NVIDIA Tegra has 3 main code structures used to develop control commands from camera input. The Optical Flow block calculates Optical Flow Φ and performs the conversion to Spherical Flow Φ_s from the camera data. The Additional Image Processing block computes anything additionally required from the camera data, which may include target finding or segmentation processes. The Control Algorithm block computes the velocity or attitude commands required. All of the attitude and control data goes to and from the Robot Operating System (ROS) messaging system running on-board the NVIDIA Tegra TX2. The data being written to and read from the ROS block is being transmitted to and from the Pixhawk 2.1 via a UART/Serial link.

4.2 Quadrotor Dynamics and Frames of Reference

This section presents the dynamics of a quadrotor and the different frames of reference for quadrotors. The frames of reference will make sure that the correct command input that is expected is applied. The known dynamics will allow the controller to be designed to allow commands to be translated to the motion expected.

The frames of reference for aerial vehicles, including the quadrotor, must be established to construct control algorithms. Most aerial vehicles, unlike ground based vehicles, follow the convention of using North-East-Down (NED) as their primary frame of reference. This is shown in Figure 4.3:

The dynamics of a quadrotor was originally derived by Hamel et al. [2002], however was later modified to account for aerodynamic drag resistance Bangura et al. [2015]. This is shown in (4.1):

$$\dot{\xi} = v \tag{4.1a}$$
$$m\dot{v} = mg\mathbf{e}_3 - TR\mathbf{e}_3 + RD, \tag{4.1b}$$
$$\dot{R} = R\Omega_\times \tag{4.1c}$$
$$I\dot{\Omega} = -\Omega_\times I\Omega + G_a + \tau, \tag{4.1d}$$

I is moment of inertia of the vehicle and D is the aerodynamic force due to translational motion in $\{B\}$. G_a is from the gyroscopic effects of each rotor and is shown in (4.2) [Hamel et al., 2002]:

$$G_a = -\sum_{i=0}^{3} I_r(\Omega \times \mathbf{e}_3)\omega_i \tag{4.2}$$

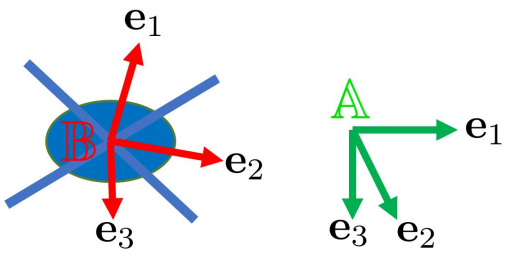

Figure 4.3: NED frame of reference is used on aerial vehicles such as the quadrotor. Split into both inertial (world) $\mathbb{A}$ and body-fixed $\mathbb{B}$ frames of reference. Note that $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$ represent the x, y, and z directions respectfully.

The gyroscopic effect shown in (4.2) is mostly caused by the speed of the rotors ω_i which typically resist change of speed and orientation during a maneuver.

The aerodynamic forces D of a quadrotor has also been investigated Bangura et al. [2012, 2015]. It is the sum of 4 different drag forces: Translational Drag, Parasitic Drag, Induced Drag and Blade Flapping. The drag force D is approximately linear at lower velocities, and can be given as:

$$D = -TK_r^B v \tag{4.3}$$

Where K_r represents the drag coefficient matrix of a symmetric quadrotor:

$$K_r = \begin{pmatrix} \bar{c} & 0 & 0 \\ 0 & \bar{c} & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{4.4}$$

While aerodynamic drag limits the ability of the quadrotor to move with speed, it is also beneficial in that it can be used to approximate the velocity v of the quadrotor. The unfiltered velocity in $\{B\}$, excluding vertical drag in $\mathbf{e}_3$, can be approximated during low speed flight as shown in (4.5) [Bangura et al., 2015]:

$${}^B v_x = \frac{a_x}{a_z \bar{c}} \tag{4.5a}$$
$${}^B v_y = \frac{a_y}{a_z \bar{c}} \tag{4.5b}$$

Where a_x, a_y, a_z are raw accelerometer values in $\{B\}$. This comes from the relationship that the accelerometer measures external forces acting on the quadrotor such that (4.6) can be derived

[Bangura et al., 2015]:

$$a = -\frac{1}{m}\begin{pmatrix} D \\ T \end{pmatrix} \tag{4.6}$$

4.3 Filtering of noisy GPS position and inertial velocity measurements to estimate velocity

A good estimate of the translational velocity v of the quadrotor is required to perform the control methods this thesis required. The GPS sensor provides a low frequency, reasonably accurate measure of the horizontal velocity of a quadrotor vehicle, at least assuming that the quadrotor is flying in an area where GPS signal is reliable. The height resolution of a GPS system is much poorer than its horizontal resolution and vertical velocities tend to be much smaller than horizontal velocities. The consequence is that the relative noise on the GPS vertical velocity component makes this unusable. We will only use the horizontal component ${}^{A}u$ of the GPS velocity measurement That is ${}^{A}u = {}^{A}u_1\mathbf{e}_1 + {}^{A}u_2\mathbf{e}_2$ with the third component in the $\mathbf{e}_3$ direction not measured. In addition to the GPS signal, the IMU can be used to provide a rough estimate of velocity by exploiting the dependence of the drag term on the linear translation of the vehicle [Bangura et al., 2015]. While GPS measurements work in the inertial frame, aerodynamic modelling is based on body-fixed frame and needs to be converted to the inertial frame. The model for IMU measurement of horizontal body-fixed frame velocity is ${}^{B}w = \frac{a_x}{c_x a_z}\mathbf{e}_1 + \frac{a_y}{c_y a_z}\mathbf{e}_2$ where c_x,c_y are the flapping angle drag constants. In the absence of noise, the relationships between the velocity measurements and the quadrotor velocity can be written as

$$\mathbb{P}_{\mathbf{e}_3}{}^{A}v = {}^{A}u \text{ and } \mathbb{P}_{R\mathbf{e}_3}{}^{A}v = R^{B}w,$$

where $\mathbb{P}_a = \mathbb{I} - aa^\top$ is the projector onto the subspace orthogonal to a unit vector $a \in \mathbb{R}^3$, $|a| = 1$ and $R\mathbf{e}_3$ is the z-axis of the quad expressed in the inertial frame coordinates.

The vertical velocity and relative height of the quadrotor are also required for the control. There is no direct velocity measurement for vertical velocity, although the barometer provides a measurement z_p of the vehicle height with respect to its takeoff height. Thus, to estimate vertical velocity we will estimate height and use the derivative of this estimate for the velocity estimate.

The following filter for estimation of translational velocity ${}^{A}\hat{v}$ is proposed:

$${}^{A}\dot{\hat{v}} = g\mathbf{e}_3 + Ra - \Delta_{v1} - \Delta_{v2} - k_{vz}\Delta_z\mathbf{e}_3, \quad {}^{A}\hat{v}(0) = {}^{A}\hat{v}_0 \tag{4.7a}$$

$$\dot{\hat{z}} = \langle \mathbf{e}_3, {}^{A}\hat{v}\rangle - k_z\Delta_z, \quad \hat{z}(0) = 0. \tag{4.7b}$$

The innovation terms are defined as:

$$\Delta_{v1} = k_{v1}(\mathbb{P}^{A}_{R\mathbf{e}_3}\hat{v} - R^{B}w) \tag{4.8}$$

$$\Delta_{v2} = k_{v2}(\mathbb{P}^{A}_{\mathbf{e}_3}\hat{v} - {}^{A}u) \tag{4.9}$$

$$\Delta_z = \hat{z} - z_p \tag{4.10}$$

Theorem 4.1. *Consider the system* (4.1) *and assume that* R *and* Ω *are known. Assume that*

measurements of horizontal inertial velocity ${}^{A}u$ and horizontal body-fixed frame velocity ${}^{B}w$ are available along with a measure z_p of relative height to a ground reference. The estimates ${}^{A}\hat{v}$ and $\hat{z}$ provided by the filter (4.7) *with innovations* (4.8), (4.9), *and* (4.10) *converge asymptotically to the true values ${}^{A}v$ and z for any initial conditions ${}^{A}\hat{v}$.*

Proof. Consider the Lyapunov functional

$$\mathcal{L} = \frac{1}{2}E_v^2 + \frac{\gamma}{2}E_z^2 \tag{4.11}$$

for

$$E_v = {}^{A}\hat{v} - v, \quad E_z = (\hat{z} - z_p)\mathbf{e}_3. \tag{4.12}$$

and gain $\gamma = k_{vz} > 0$. Computing the derivative of the Lyapunov function gives;

$$\begin{aligned}
\dot{\mathcal{L}} =& E_v^\top((g\mathbf{e}_3 + Ra - \Delta_{v1} - \Delta_{v2} - k_{vz}\Delta_z\mathbf{e}_3) - (g\mathbf{e}_3 + Ra)) \\
&+ \gamma E_z^\top((\langle \mathbf{e}_3, {}^{A}\hat{v}\rangle - k_z\Delta_z)_-(\langle \mathbf{e}_3, v\rangle))\mathbf{e}_3 && \text{(4.13a)} \\
=& E_v^\top(-k_{v1}(\mathbb{P}^A_{R\mathbf{e}_3}\hat{v} - {}^{A}w) - k_{v2}(\mathbb{P}^A_{\mathbf{e}_3}\hat{v} - {}^{A}u) \\
&- k_{vz}(\hat{z} - p)\mathbf{e}_3) + \gamma E_z^\top(\langle \mathbf{e}_3, {}^{A}\hat{v} - v\rangle - k_z(\hat{z} - p))\mathbf{e}_3 && \text{(4.13b)}
\end{aligned}$$

It is easily verified that

$$\mathbb{P}^A_{R\mathbf{e}_3}\hat{v} - {}^{A}w = \mathbb{P}_{R\mathbf{e}_3}E_v, \quad \mathbb{P}^A_{\mathbf{e}_3}\hat{v} - {}^{A}u = \mathbb{P}_{\mathbf{e}_3}E_v. \tag{4.14}$$

Substituting into (4.13b) one obtains

$$\begin{aligned}
\dot{\mathcal{L}} =& -k_{v1}E_v^\top\mathbb{P}_{R\mathbf{e}_3}E_v - k_{v2}E_v^\top\mathbb{P}_{\mathbf{e}_3}E_v - k_{vz}E_v{}^\top E_z \\
&+ \gamma E_z^\top\langle \mathbf{e}_3, E_v\rangle\mathbf{e}_3 - \gamma k_z E_z^\top E_z. && \text{(4.15a)}
\end{aligned}$$

Recalling that $\gamma = k_{vz}$ then $-k_{vz}E_v{}^\top E_z + \gamma E_z^\top(\langle \mathbf{e}_3, E_v\rangle)\mathbf{e}_3 = 0$. It follows that

$$\dot{\mathcal{L}} = -k_{v1}E_v^\top\mathbb{P}_{R\mathbf{e}_3}E_v - k_{v2}E_v^\top\mathbb{P}_{\mathbf{e}_3}E_v - \gamma k_z E_z^\top E_z \tag{4.15b}$$

The projector matrices $\mathbb{P}_{R\mathbf{e}_3}$ and $\mathbb{P}_{\mathbf{e}_3}$ are positive definite and it follows that $\dot{\mathcal{L}}$ is negative semi-definite and negative definite if $R \neq I_3$. If the attitude $R(t)$ of the quadrotor is persistently exciting in the sense that there exists two constants $k_{v1}, k_{v2} > 0$ such that

$$\int_t^{t+\delta}(k_{v1}\mathbb{P}_{R(\tau)\mathbf{e}_3} + k_{v2}\mathbb{P}_{\mathbf{e}_3})\mathrm{d}\tau > \gamma\tau,$$

then exponential global stability of the error signals follows directly from Lyapunov's principle along with an averaging argument [Khalil, 1996] since the decrease function is negative definite in the error variables. In more generality, a Barbalat argument can be used to prove global asymptotic stability of the filter and a linearisation argument can be used to demonstrate local exponential stability.

4.4 Optical Flow Characteristics

Optical flow is the transport of image intensities across the image plane. As such, optical flow can be estimated from a brightness constancy PDE (4.16) [Horn and Schunck, 1981; Adarve and Mahony, 2016]:

$$\frac{\partial E}{\partial x}u + \frac{\partial E}{\partial y}v + \frac{\partial E}{\partial t} = 0 \tag{4.16}$$

Where $u = \frac{\mathrm{d}x}{\mathrm{d}t}$ is the optical flow along the horizontal axis and $v = \frac{\mathrm{d}y}{\mathrm{d}t}$ is the optical flow in the vertical axis.

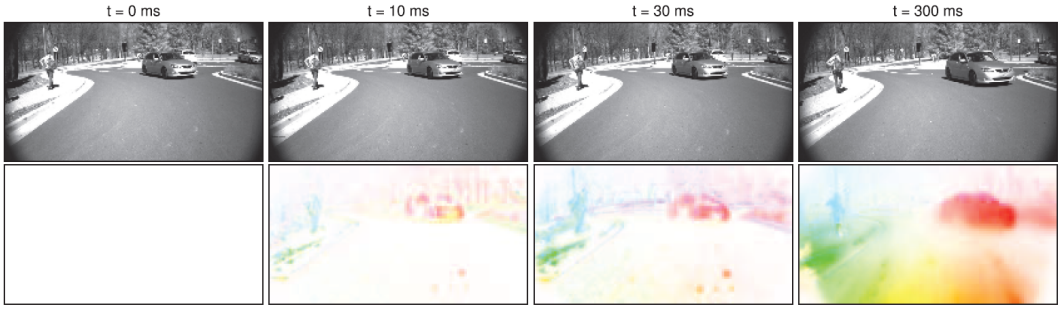

Figure 4.4: Example from the paper 'A Filter Formulation for Computing Real Time Optical Flow' [Adarve and Mahony, 2016]. Real time optical flow computed using a 300Hz camera input. The optical flow filter forms over a number of frames based on texture quality on a GPU.

The optical flow chosen is the real-time filter formulation proposed by Adarve and Mahony [2016]. It is designed for real-time implementation at multi-hundred rates on graphics processing units (GPU) or field-programmable gate arrays (FPGA), which are designed for mass parallel programming. To be able to be run in real time on a GPU, it is a locally optimised algorithm using texture close to the pixel to determine the optical flow. To get texture differences, it uses the common Pyramidal method. The algorithm operates in 3 primary stages. The first stage, the image pre-processing stage, the brightness parameters of the current frame are determined, including brightness and brightness gradients. During the second stage, the propagation stage, the filter predicts the next optical flow values for the next time iteration. The third stage, the update stage, combines the brightness parameters with the predicted optical flow to get a new estimate of the optical flow field. This method keeps temporal and spacial consistency unlike the Lucas-Kanade method, as it relies of multiple frames to form estimates rather than just two frames.

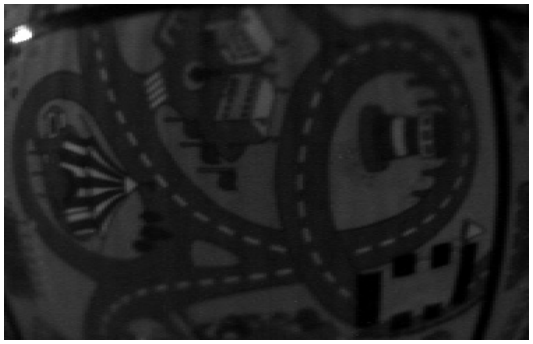

Figure 4.5: Semi-textured surface raw image. From a hovering quadrotor with a downwards facing camera while computing optical flow.

Figure 4.6: Semi-textured surface, using optical flow filter by Adarve and Mahony [2016] with additional low texture cut-off constraint.

Due to the locally optimised nature of the optical flow algorithm, good texture quality is required to properly develop the the optical flow estimate Adarve and Mahony [2016]. In forest-like environments, there is enough texture from all of the tree textures and leaves to ensure the algorithm should work as expected. In the cases there is not much texture, such as the blue sky or low texture tree surfaces, we have added an extra condition that sets the optical flow to 0 so it does not feed false information into the control algorithms (see Figure 4.5 and 4.6). This was implemented as the temporal and spacial consistency typically pushes optical flow over untextured regions in the images, which means the sky then acts as a obstacle to avoid.

4.5 De-rotation of Optical Flow

Optical flow Φ can be split into its translational Ψ and rotational Θ components. Rotational optical flow however gives no useful scene information, as its value is independent of distances to a targets. This requires the removal of the rotational component of the optical flow to get any useful scene estimation. The rotational optical flow Θ produced from the optical flow field can be predicted if the rotational velocity Ω of the camera relative to the world frame is known [Honegger et al., 2013].

In pinhole cameras, the camera frame is warped due to the focal point and the lens attached to it. The rotational flow Θ for each individual pixel can be estimated using (4.17):

$$\Theta = (K - \xi_p \mathbf{e}_3^\top)\Omega_\times K^{-1}\xi_p \tag{4.17}$$

This is where K is the camera intrinsic matrix and ξ_p is the pixel coordinate (relative to centre of the camera plane).

Equation (4.17) is capable of estimating the rotational optical flow in the ideal circumstance where the whole scene has no distortion, scene is fully textured, and optical flow estimation is instantaneous. While it is effective at estimating optical flow on the average of optical flow vectors, due to its temporal and spacial consistency, it is very rarely the case that it can estimate

the rotational optical flow for each individual pixel required by the algorithm from Adarve and Mahony [2016]. For this reason, we use a 3-axis stabilised gimbal to remove rotational optical flow.

4.6 Average Inertial Spherical Flow w for Quadrotor Control

The Average Inertial Spherical Flow w is calculated from Spherical Flow Φ_s (4.18). The average inertial spherical flow is a measure of the scaled velocity of the camera in all 3 directions (x, y, z). The z-direction estimate is based on the divergence and convergence of the spherical flow field [Herisse et al., 2012]. Let η be a vector heading in any chosen direction from the spherical camera's position $p_{\eta 0}$. For the experiments in this thesis, this vector will point directly downwards to the ground. w is calculated in a disk-like domain D expanding tangentially on and from the spherical camera's image surface. The expansion point (centre) of disk-like domain D is the intersection of the spherical camera's image surface and the vector η. The angle θ_D is the angle between η and a vector formed from $p_{\eta 0}$ to the disk-like domain's D edge. The rotation of the plane relative to the spherical camera is given as R_t [Herisse et al., 2012].

$$w = -\left(R_t \Lambda R_t^{-1}\right) R_c \left(\int_D \Phi_s(\bar{\xi}) \mathrm{d}t\bar{\xi} + \pi(\sin\theta_D)^2 \Omega \times R_c^\top \eta\right) \tag{4.18a}$$

$$\Lambda = \frac{\pi(\sin\theta_D)^4}{4} \begin{pmatrix} \frac{1}{\lambda} & 0 & 0 \\ 0 & \frac{1}{\lambda} & 0 \\ 0 & 0 & 2 \end{pmatrix} \tag{4.18b}$$

$$\lambda = \frac{(\sin\theta_D)^2}{4 - (\sin\theta_D)^2} \tag{4.18c}$$

However, since the optical flow Φ measured is based on a 2D image plane, we need to calculate the spherical flow Φ_s (4.19), based on the projection of the 2D plane onto a unit sphere.

$$\Phi_s(\bar{\xi}) = \frac{\sqrt{(\bar{\xi} - \bar{\xi}_0)^2 + f^2}}{f} \begin{pmatrix} \Phi(\bar{\xi}) \\ f\frac{|\Psi(\bar{\xi})|}{|\bar{\xi} - \bar{\xi}_0|} \end{pmatrix} \tag{4.19}$$

Where $\bar{\xi}_0$ is the camera centre pixel.

The Average Inertial Spherical Flow w is a necessary step in attempting to purely visually hover the quadrotor over the ground using optical flow, without using other methods such as Simultaneous Localisation and Mapping (SLAM), or Visual-Inertial Odometry (VIO) for state estimation. This is used during all optical flow control based experiments during this thesis.

4.7 Performance Gain Tuning

To achieve accurate and performance effective velocity control, a velocity controller based on a proportional-integral (PI) controller is used. This is designed to control the roll ϕ, pitch θ

and the thrust T of the quadrotor. This is shown in (4.20):

$$\phi_d = k_{py}\frac{s+k_{iy}}{s}({}^B v_{yd} - {}^B\hat{v}_y) \tag{4.20a}$$

$$\theta_d = -k_{px}\frac{s+k_{ix}}{s}({}^B v_{xd} - {}^B\hat{v}_x) \tag{4.20b}$$

$$T_d = mg - k_{pz}\frac{s+k_{iz}}{s}({}^A v_{zd} - {}^A\hat{v}_z), g = 9.81ms^{-2} \tag{4.20c}$$

In practice, the PI-based controller is implemented in the quadrotor code in its parallel form.

Calibration is required to achieve the maximum possible performance for the quadrotor. For calibration, we use a pole-zero based method for determining the best response the velocity controller (4.20) can achieve.

4.7.1 Velocity gain tuning for general flight

In normal flight conditions, such as for when using GPS or VICON based position and velocity estimates, these estimates can be assumed to be quick enough without having much of a delay. The gains determined in this section were used during the main obstacle avoidance experiments. The transfer function block diagram of a PI velocity controller for a quadrotor using close to or real-time position and velocity measurements is given as shown in Figure 4.7:+

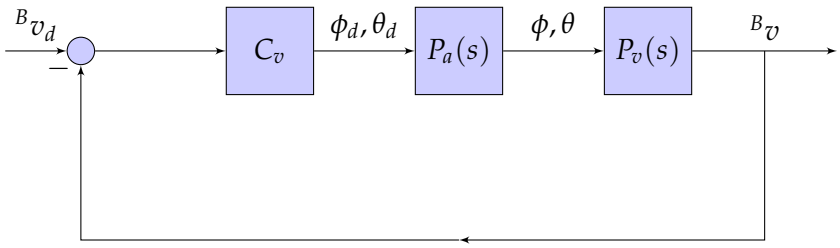

Figure 4.7: $H_v(s)$ Velocity Control Transfer Function

The transfer function $H_v(s)$ shown in Figure 4.7 is given as a larger block of 3 smaller transfer functions. C_v is given as the transfer function of the velocity controller as shown in (4.20). $P_a(s)$ is the transfer function that describes the transition of the attitude ϕ, θ to the desired attitude ϕ_d, θ_d based on the quadrotor's dynamics, flight controller software, and the four electronic speed controller and motor responses. $P_v(s)$ is the transfer function that describes the transition between attitude and the horizontal component of the body-fixed frame velocity ${}^B\dot{v}_{xy}$. The quadrotor is considered symmetrical, meaning the transfer function for $P_a(s)$ is considered equivalent for both roll and pitch.

We used the *systemIdentification* toolbox provided by MATLAB to estimate $P_a(s)$. Estimated to be a single pole system, $P_a(s)$ was determined to be approximately (4.21):

$$P_a(s) = \frac{9}{s+9} \tag{4.21}$$

To determine $P_v(s)$, we used the quadrotors kinematics/dynamics shown in (4.1). We know

from (4.1), that (4.22) can be formulated:

$$+ -^{B}\dot{v} = g\sin\theta, g = 9.81ms^{-2} \tag{4.22}$$

While the transfer function of $\sin\theta$ is complex, we know the attitude of the vehicle is never more than $\frac{\pi}{6}$ radians from design decisions. This means we invoke the rule that $\sin\theta \approxeq \theta$, when θ is small. This leads to (4.23):

$$^{B}\dot{v} = g\theta \tag{4.23}$$

From (4.23), we can get $P_v(s)$ (4.24):

$$P_v(s) = \frac{g}{s} \tag{4.24}$$

Since we now have the transfer functions of kinematics of the quadrotor $P_a(s)$ and $P_v(s)$, we can tune the velocity controller. This was done using the *sisotool* pole-zero editor toobox in MATLAB.

The tuned PI controller C_v is shown in (4.25) as an adjustment of (4.20):

$$\phi_d = C_v(^{B}v_{yd} -^{B}\hat{v}_y) \tag{4.25a}$$

$$\theta_d = -C_v(^{B}v_{xd} -^{B}\hat{v}_x) \tag{4.25b}$$

$$C_v = 0.657\frac{s + 0.039}{s} \tag{4.25c}$$

The step response to the system $H_v(s)$ is shown in Figure 4.8 after tuning, which has a corresponding rise time of 0.238s.

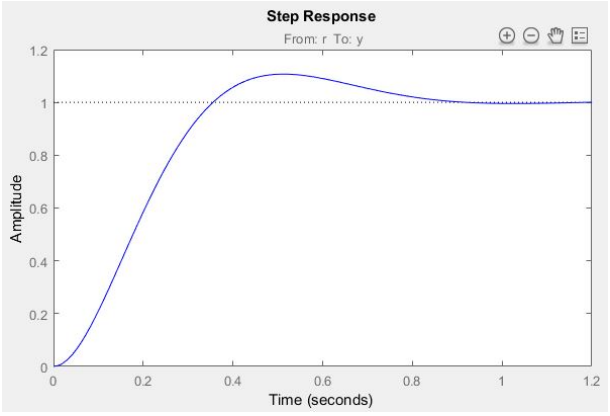

Figure 4.8: Step Response of the tuned Quadrotor

4.7.2 Velocity and Position Gain Tuning for Velocity Estimates using Optical Flow

During the optical flow based hovering experiments, the chosen optical flow algorithm was used to estimate the velocity of the vehicle. This optical flow algorithm has a linear response

delay due to its differentially-driven method. This adds an additional return transfer function when attempting to tune the system, compared the previous section's method. The quadrotor's system transfer function using w was determined so that the performance of the quadrotor was maximised. It can be summarised in the velocity controller transfer function (Figure 4.10) and position controller transfer function (Figure 4.9).

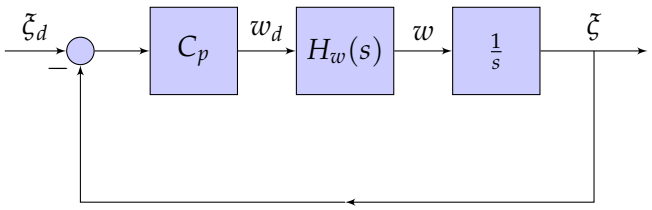

Figure 4.9: $H(s)$ Position Controller Transfer Function

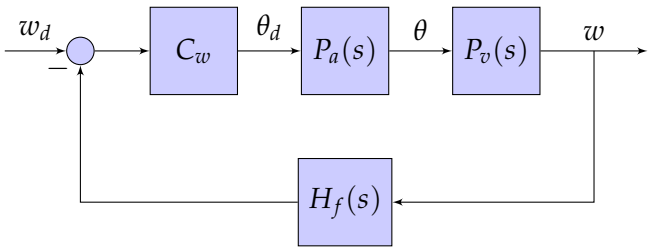

Figure 4.10: $H_w(s)$ Velocity and Attitude Response Transfer Function

The position controller transfer function $H(s)$ is defined by 3 small transfer functions. A proportional controller C_p is used to control the position of the quadrotor, and is defined as $C_p = w_d = -k_p(\xi - \xi_d)$. $H_w(s)$ is the velocity response transfer function. The system integrator is used to integrate the structural flow w to get a position estimate. However, when the object that we are visually servoing to is visible, we use the position estimate of that point as the position ξ feedback instead.

The velocity and attitude response $H_w(s)$ is comprised of 4 parts. The first part is the proportional controller C_w used to control the velocity of the quadrotor, by adjusting the attitude of the quadrotor. This is derived as $C_w = \theta_d = -k_w(w - w_d)$. The 2 plants are the same as (4.21) and (4.24), as it was the same quadrotor. The last component of $H_w(s)$ is the feedback of the structural flow w estimate having a linear response and offset due to the nature of the optical flow algorithm. The linear system response of w relative the measured velocity is defined below (4.26):

$$H_f(s) = \frac{7.49}{s+7} \tag{4.26}$$

$H_w(s)$ as shown in 4.7 was tuned to have a step response as shown in Figure 4.11. $H(s)$ as shown in 4.9 was tuned to have a step response as shown in Figure 4.12.

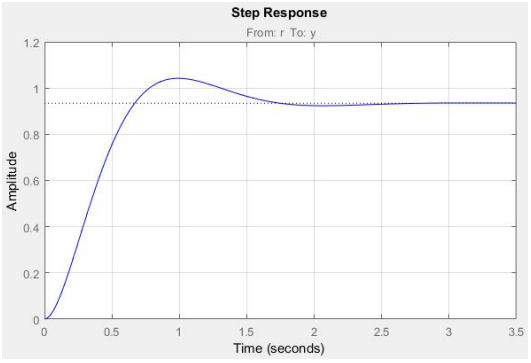

Figure 4.11: Step Response of $H_w(s)$ (Velocity Control)

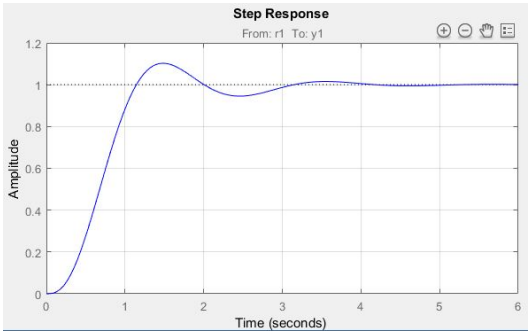

Figure 4.12: Step Response of $H(s)$ (Position Control)

The proportional controller gain k_w in C_w was set to 0.2820. This has a rise time of 0.4584 seconds. The proportional controller gain k_p in C_p was set to 1.6543. This has a rise time of 0.6970 seconds.

4.8 Summary

The quadrotor software architecture described during this chapter is used to control the quadrotor in the experiments later described in this thesis, which includes vision-based hovering experiments and reactive obstacle avoidance experiments using optical flow $\Phi(\bar{\xi})$.

The Pixhawk 2.1 uses the PX4 architecture and custom modules to allow the quadrotor to fly given velocity commands ${}^{B}v_d$,${}^{A}v_d$. The velocity filter (4.7) was used to filter IMU (Inertial Measurement Unit) data Ω, a, z_p and combined measurements from other systems such as GPS velocity ${}^{A}u$, or vision-based velocity and height measurements ${}^{B}w$, z_p to estimate velocity ${}^{B}\hat{v}$ and altitude $\hat{z}$ in real-time. From these measurements and velocity commands from the TX2 ${}^{B}v_{(x,y)d}$, ${}^{A}v_{zd}$, the system estimates the required thrust T_d and attitude ϕ_d, θ_d using the PI controller (4.20) to control the quadrotor.

Using the image input from the Matrix-Vision mvBlueFOX-200w camera, optical flow $\Phi(\bar{\xi})$ was calculated. The NVIDIA Tegra TX2 then used the optical flow measurements, and any additional image processing, to determine the desired velocity ${}^{B}v_{(x,y)d}$, ${}^{A}v_{zd}$. The algorithms to determine the desired velocity are described later in the the thesis.

Chapter 5

Vision-based Hover Control

Obtaining position estimates via visual odometry allows for autonomous flying robots to control its position without the need for GPS-based systems, which can be easily obscured in complex environments. A few of the experiments performed to test the quadrotor platform's performance was based on hovering the quadrotor above the ground, with some manoeuvres being performed. All of the experiments used dense optical flow calculated on a NVIDIA Tegra TX2 at 70 Hz to estimate the velocity of the quadrotor above ground. To perform these hovering tests, we used the Average Inertial Spherical Flow w approximation from previous conference and journal papers via projecting the 2D optical flow onto a unit sphere Herisse et al. [2012]. Via the filters (5.2) and (5.4), we integrated optical flow estimates to estimate horizontal and vertical position. A known landmark was used correct for drift when the vehicle was close to its initial position and the landmark was in view.

5.1 Position Estimation from a known landmark

Landmarks are known features that can be used to verify a vehicle's location. Landmarks can used to provide a position estimate of the camera that has detected the landmark, if the position of the marker in the environment is known. Fiducial markers are examples of a landmark designed for use with computer/robotic vision algorithms. Some fiducial marker detection algorithms, such as for the ArUco marker, are openly available via OpenCV. In our experiments, we used the ArUco fiducial marker algorithm used by OpenCV to correct the estimate of position when in good view.

Estimating the horizontal position of an ArUco marker can be done via (5.1). Using the pixel position of the corners detected by the OpenCV algorithm, you can estimate the centre pixel ζ_p of the ArUco marker by the average corner of the pixel positions. Using the known camera focal length f, image centre pixel ζ_{0p}, and the height estimate $\hat{z}_L$ (5.2), you can estimate the horizontal position of ArUco marker using (5.1).

$$\hat{\zeta}_s = -\hat{z}_L \frac{\zeta_p - \zeta_{0p}}{f} \tag{5.1}$$

You can estimate the height of the ArUco marker z_f using the known ArUco marker size (eg. 30cm target), corner pixel positions, and focal length. The corner pixel separation scales with height of the camera from the marker, where the further away markers have less apparent

Figure 5.1: ArUco Marker Detection. The grey dot in the centre of the ArUco fiducial marker is the calculated centre of the ArUco marker using the marker corners.

corner pixel separations. The height can be estimated using the OpenCV ArUco detection algorithms.

5.2 Vertical Position Estimation via a Logarithmic Filter of w and a landmark

Equation (4.7) uses the barometer to determine vertical velocity. This filter is however not adequate to determine true height due to barometer measurements being susceptible to pressure and temperature variations. This tends to result in variation between $\pm 1.5m$, more so in an indoor environment with air-conditioning systems. However this is accurate enough to get an approximate velocity estimate $\hat{v}_z$.

We implement another filter $\hat{L}_z$ that uses the vertical component of the structural flow representation $w_z = w\mathbf{e}_3$ from (4.18). The structural flow is equivalent to $w = \frac{v}{d}$, where d is the height from the ground when using a gimbal stabilised camera pointing face-down. This suggests it is logarithmic in nature to the height from the ground. k_t is a gain used to represent the texture of the surface and k_{Lz} is a convergence gain which can be adjusted depending on the reliability of the estimated height z_f of a ground based landmark (e.g. ArUco marker or landing pad). Filter (5.2) is used to approximate height:

$$\dot{\hat{L}}_z = -k_t w_z - k_{Lz}\left(\hat{L}_z - \log(z_f)\right) \tag{5.2a}$$

$$\hat{z}_L = -e^{\hat{L}_z} \tag{5.2b}$$

Note that $\hat{z}_L$ has a positive upwards direction, in contradiction to standard flying robotics, due to logarithmic values being greater than 0. As such, $\hat{L}_z(0)$ is the estimated height of the vehicle when the filter is started, as a positive value. When $\hat{L}_z \leq 0$, $\hat{L}_z$ is reset to $0.1m$ to stop the logarithmic value $\hat{L}_z$ going beyond the logarithmic limits. If the surface is not well textured, k_t has to be increased to compensate for lost optical flow. This can be calibrated before flight.

The filter (5.2) allows for the structural flow w to control height. Since w_z responds to all changes in altitude, a proportional height controller $v_{zd} = -k_z(\hat{z}_L - z_d)$ can be used to

stabilise height in indoor and outdoor environments.

This method can only be implemented after a certain height is achieved after launch, as estimates of w when the quadrotor's camera is very close to the ground are inaccurate. To deal with this situation, the quadrotor is launched to a height above the landmark, specifically an ArUco marker, before switching to the $\hat{z}_L$ (5.2) filter. The height of the ArUco marker can be detected via the OpenCV libraries. The OpenCV algorithm uses the camera parameters, marker corner pixel coordinates, and known marker size to estimate the height.

5.3 Horizontal Position Estimation via the integration of w and correction via the ArUco fiducial markers

The spherical average inertial flow w (4.18) is capable of estimating the horizontal position of the camera if the height $\hat{z}_L$ (5.2) is known, which can be calculated using (5.3).

$$\xi_w = k_t \hat{z}_L \int w \mathrm{d}t \tag{5.3}$$

This is simply the integral of w with reference to height $\hat{z}_L$ and the texture quality k_t.

The main disadvantage of integrating w directly is that the position estimate drifts over time, without anything to correct it. To deal with drift caused by (5.3), we used the horizontal position estimate of the ArUco marker (5.1) in a filter (5.4).

$$\dot{\hat{\xi}}_v = k_t \hat{z}_L w - k_s \left(\hat{\xi}_v - \xi_s \right) \tag{5.4}$$

The output horizontal position estimation algorithm (5.4) relies on the optical flow to forward estimation position. If the ArUco marker is detected, then k_s is set to a constant positive value. If the ArUco marker is not detected, then k_s is set to 0, effectively ignoring old ArUco marker position estimates. Ultimately this visual odometry method is designed to work at the high frequency (70Hz) of the camera, and can navigate away from the marker for a period of time.

5.4 Experimental Results

The experimental results for a hovering experiment using the average inertial spherical flow estimate and the ArUco marker for drift correction is shown in Figures 5.3 to 5.6. The VICON motion capture system was used to act as a ground-truth to verify the results. The experiment consisted of a figure of 8 flight in the air.

Figure 5.2: Quadrotor Experiment with assistance of the ArUco marker

The results show that the system can perform visual navigation[1]. The integration of the average spherical flow estimates w with the ArUco fiducial marker as a backup origin reference, via the filters (5.2) and (5.4), does allow the system to visually navigate.

The drift in horizontal position could be explained by the average spherical flow estimate integration error and the minor mismatch between the horizontal position with the ground truth. Since filter (5.4) relies on a good vertical position estimate, this could create the scaling difference between vision position estimates and ground-truth.

The vertical position estimate appears linked with the horizontal position estimates. The peaks and valleys of the vertical estimates match the peaks and valleys of the horizontal position estimates. This could be explained by the camera intrinsics and heavy distortion not being consistent, and also by the camera gimbal not having a perfect response.

There were a few known difficulties during the experiment in regards to the environment's texture. Low texture environments as perceived by the camera reduce the reliability of the optical flow algorithm. This often causes velocity estimates from the optical flow to have a large amount of error, which leads on to high error in position estimates. These low textures as perceived by the camera can include tiles, dirt, or when the camera is over or under exposed. The purpose of the landmark origin or ArUco marker was to be used to reduce the cumulative error.

5.5 Comparison of the ArUco fiducial marker position estimate with other methods

Two potential replacements for the ArUco fiducial marker were first investigated, but later ignored. One is via using a chessboard target, and another using a coloured target. The chessboard target used an OpenCV algorithm to detect it, while the coloured target used HSV parameters on a coloured image to get a masked output.

The ArUco marker method was ultimately used due to its high reliability and performance compared to both the coloured target and chessboard detection methods. The colour segmen-

[1] Video of the Vision-based Hover and Figure of Eight experiment can be found via https://www.youtube.com/watch?v=gUgmRTYIjJA&t=57s

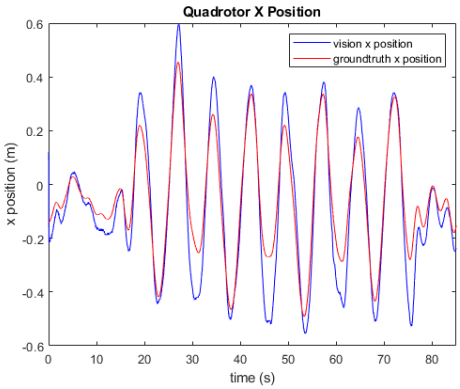

Figure 5.3: Hover Experiment X Position Estimate. The blue line is the vision position estimate and the red line is the groundtruth.

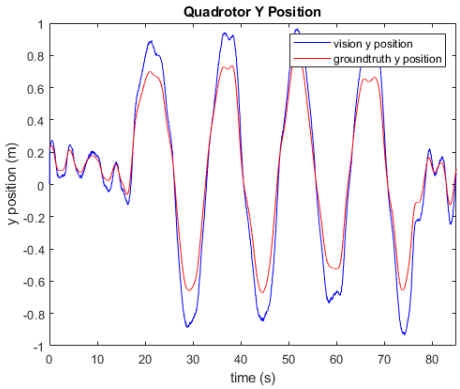

Figure 5.4: Hover Experiment Y Position Estimate. The blue line is the vision position estimate and the red line is the groundtruth.

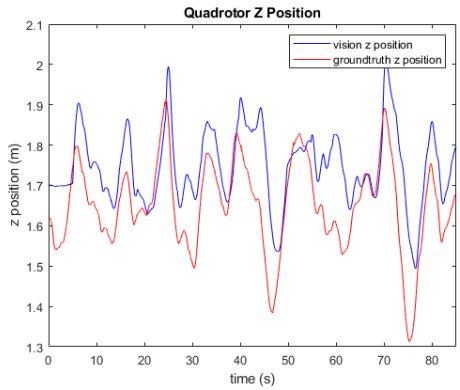

Figure 5.5: Hover Experiment Z Position Estimate. The blue line is the vision position estimate and the red line is the groundtruth.

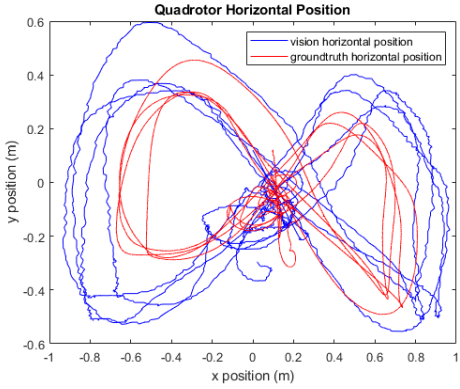

Figure 5.6: Hover Experiment Horizontal Position Estimate. The blue line is the vision position estimate and the red line is the groundtruth.

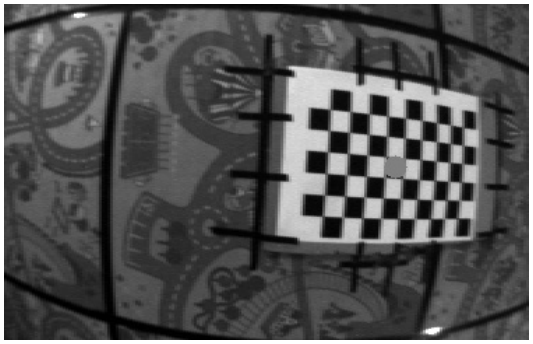

Figure 5.7: Chessboard Centre Detection. The grey dot in the centre of the chessboard is the calculated centre of the chessboard using the chessboard corners.

Figure 5.8: Colour Segmented Output. The grey dot in the centre of the white area is the calculated centre of the coloured target.

tation algorithm relied on constant adjustment of HSV parameters as the day progressed, while also needing adjustment during different weather conditions. This was particular noticeable in outdoor environments or rooms with windows in which outdoor light can enter indoors. This made it unreliable to use this method over a period of time. The colour segmentation method also suffered when points outside of the colour target were within the HSV parameters, causing an apparent shift of the origin.

While the chessboard detection algorithm successfully detected the centre, the ArUco marker outperformed the chessboard algorithm. Both the ArUco and chessboard methods were robust to lighting conditions, assuming the lighting was good enough to make the white and black squares on the ArUco and chessboard markers distinguishable. The ArUco marker detection algorithm almost never lost track of its position, even when near the borders of the highly distorted image from the 120^o field-of-view (FOV) camera. The chessboard algorithm was susceptible to failed detection when near the borders of the highly distorted image. The chessboard detection algorithm also took more time ($\approx 10ms$) to estimate compared to the ArUco marker method ($< 5ms$).

Chapter 6

Vision based Forward Sensitive Reactive Control for a Quadrotor VTOL

Deployment of aerial robotic vehicles for real world tasks such as home deliveries, close range aerial inspection, etc., require robotic vehicles to fly through complex and cluttered 3D environments such as forests, shrubbery or into balconies, garages, or sheds. Dense high-speed optical flow can provide real-time motion cues for obstacle avoidance that does not require 3D full reconstruction of the environment. However, classical reactive control does not 'look ahead' and tends to bounce off obstacles rather than generating a smooth trajectory that anticipates and avoids upcoming obstacles. In this chapter, we consider deriving a fully image based control criteria that forward predicts a cylinder of free space into the image flow representation of the environment and steers the vehicle by manoeuvring this cylinder through the upcoming environment. The length and radius of the cylinder provide a guarantee that the vehicle can indeed fly through the space identified and the fact that it is predicted forward into the environment leads to smooth anticipation of upcoming obstacles. Results are obtained for a quadrotor flying autonomously through a forest environment. This chapter is based off our published conference paper Vision Based Forward Sensitive Reactive Control for a Quadrotor VTOL [Stevens and Mahony, 2018].

6.1 Forward Vision Tunnel-based Optical Flow Controller

Real-time dense optical flow is used as the basic sensing modality for obstacle avoidance. The flow algorithm used is that developed by Adarve and Mahony [2016]. We compute the dense optical flow at 120Hz for the control algorithm for a 376x240 pixel image sequence.

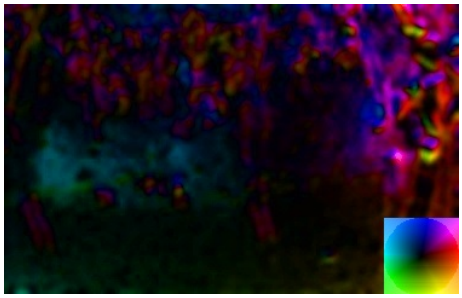

Figure 6.1: Dense Optical Flow. This was taken when a quadrotor was flying past a tree. Optical flow is a measurement of how much a texture moves each frame, measured in pixels. Each colour represents a direction, while the intensity of the colour represents the magnitude of the optical flow. The colour wheel at the bottom right of the image describes the direction and magnitude of the colours, where the optical flow vector goes from the centre of the colour wheel outwards.

Conceptually the control design is based on the principle of steering into free space. The novelty of the design is to reformulate this principle into a purely image flow based criterion. Dense optical flow is a vector field $\Phi(\bar{\xi},t)$ that assigns an infinitesimal image motion vector to each point $\bar{\xi}$ in the image at each time t. Given a point $\bar{\xi}=(\bar{\xi}_x,\bar{\xi}_y)$ in a perspective image the optical flow, for a calibrated camera with focal length f, can be written:

$$\begin{aligned}\Phi(\bar{\xi},t) &= \Psi(\bar{\xi},y)+\Theta(\bar{\xi},t) \\ &= \frac{1}{Z}\begin{pmatrix}-fV_x+\bar{\xi}_xV_z\\-fV_y+\bar{\xi}_yV_z\end{pmatrix}+\begin{pmatrix}\frac{1}{f}\bar{\xi}_x\bar{\xi}_y\Omega_x-(f+\frac{1}{f}\bar{\xi}_x^2)\Omega_y+\bar{\xi}_y\Omega_z\\(f+\frac{1}{f}\bar{\xi}_y^2)\Omega_x-\frac{1}{f}\bar{\xi}_x\bar{\xi}_y\Omega_y-\bar{\xi}_x\Omega_z\end{pmatrix}\end{aligned} \tag{6.1}$$

where $\Psi(\bar{\xi},t)$ is the optical flow due to translation of the camera and $\Theta(\bar{\xi},t)$ is optical flow due to rotation of the camera. Note that the rotational optical flow depends only on pixel coordinates $\bar{\xi}=(\bar{\xi}_x,\bar{\xi}_y)$ and the angular velocity $\Omega=(\Omega_x,\Omega_y,\Omega_z)$ and does not provide any information on the motion of the vehicle through the local environment.

In this paper we will use the stabilised optical flow

$$\Psi(\bar{\xi},y)=\Phi(\bar{\xi},t)-\Theta(\bar{\xi},t).$$

We use a 3 axis gymbal to physically de-rotate the camera, allowing us to apply the optical flow algorithms directly to the image sequence to generate stabilised optical flow. Although it is conceptually attractive to use a physically attached camera and de-rotate the flow Ψ (computed from the measurement Ω) by subtracting the rotational flow, in practice, Θ can be considerably larger than the translational flow Ψ and numerically computing the difference can introduce significant errors. We note that most animals and insects that use vision as a primary motion sensor also use a physical de-rotation of the eye, rather than relying on post processing of the flow signal.

A key aspect of the control design is the use of the known vehicle velocity. This velocity

lies in a direction which can be mapped to an image pixel coordinate

$$\bar{V} = \left(\frac{f {}^{C}v_x}{{}^{C}v_z}, \frac{f {}^{C}v_y}{{}^{C}v_z} \right) \tag{6.2}$$

${}^{C}v = R_{\text{cam}} {}^{B}v$ is the vehicle velocity written in the camera frame and R_{cam} is the rotation of the camera frame with respect to the body-fixed-frame. Note that $\Psi(\bar{V}, t) = 0$ since there is no optical flow generated along the axes of motion (focus of expansion) in the image.

We conceptualize a finite cylinder in 3D-space of radius r and depth d that projects forward from the camera frame of reference in the direction of motion $\bar{V}$ of the vehicle (Fig. 6.2).

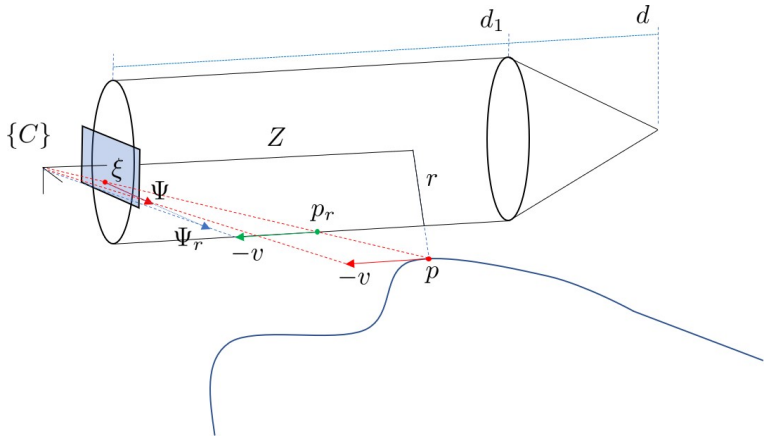

Figure 6.2: Geometry of the cost function construction. The cylinder and cone section of the free space estimate are shown. A point p in the environment is observed by the camera at pixel coordinates $\bar{\xi}$ on the image plane. The optical flow of the real environment point is denoted $\Psi(\bar{\xi})$. A virtual point p_r lies on the radius of the cylinder. The virtual optical flow, that would be measured if environment was touching the boundary of the cylinder is denoted $\Psi_r(\bar{\xi})$. Note that $|\Psi_r| > |\Psi|$ for point p outside the cylinder.

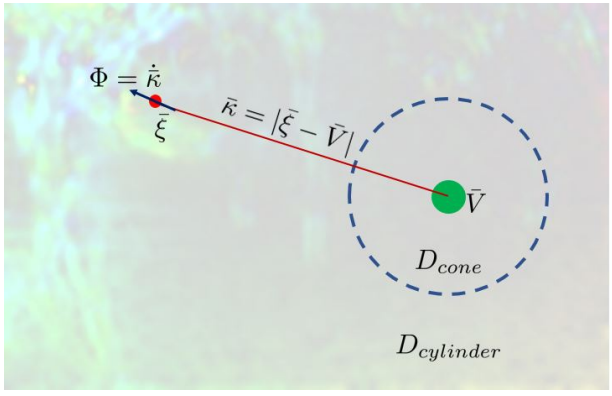

Figure 6.3: Visual representation of the cylinder $D_{cylinder}$ and cone D_{cone} projected on to the 2D image plane. Denoting $\bar{\kappa} = |\bar{\xi} - \bar{V}|$, the optical flow Φ would appear as a vector leading away from the point $\bar{V}$, which is the same as the derivative of $\bar{\kappa}$.

Consider a point p (see Figure 6.2) in the environment and assume that it is stationary. Its relative velocity with respect to the camera is $-v$. Let $\bar{\xi}$ denote the calibrated pixel coordinates of this point as observed in the image, then (6.1) can be written

$$\Psi(\bar{\xi},t) = \frac{1}{Z}\left(-f\mathbb{P}\,{}^{C}v + \bar{\xi}\mathbf{e}_3^{\top}\,{}^{C}v\right) \tag{6.3}$$

where

$$\mathbb{P} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

is the calibrated image projection matrix. Collecting terms, it is straightforward to verify that

$$\frac{\Psi(\bar{\xi},t)}{\mathbf{e}_3^{\top}\,{}^{C}v} = \frac{1}{Z}(-\bar{V} + \bar{\xi}) = \frac{\bar{\xi} - \bar{V}}{Z}. \tag{6.4}$$

Given that the velocity ${}^{C}v$ is known it should in principle be possible to invert the relationship (6.4) and estimate the depth Z of the environment point. However, the flow Ψ is noisy, and the velocity estimate ${}^{C}\hat{v}$ is also uncertain. The consequence is that the raw depth data extracted in this manner is unsuitable for standard free space path planning algorithms. However, we can consider a criteria directly on the flow by noting that the flow increases as Z decreases for a fixed pixel coordinate $\bar{\xi}$. That is for points p closer to the camera along the ray associated with pixel coordinates $\bar{\xi}$, the value of the flow $\Psi(\bar{\xi})$ will be less than flow Ψ_r associated with a virtual point p_r that has the same pixel coordinates but lies on the cylinder (Fig. 6.2). Let Z_r denote the depth of the virtual point p_r. Then for a camera of focal length f and a cylinder of radius r one has

$$\frac{Z_r}{r} = \frac{f}{|\bar{\xi} - \bar{V}|} \tag{6.5}$$

Let d denote the look ahead distance to be considered Fig. 6.2. Define a cost function

$$\sigma(\bar{V},t;\bar{\xi},\Psi(\bar{\xi},t),f,r) = \frac{|\Psi(\bar{\xi},t)|}{\mathbf{e}_3^{\top}\,{}^{C}v}.\frac{rf}{|\bar{\xi} - \bar{V}|^2}\mu\left(\frac{rf}{|\bar{\xi} - \bar{V}|}\right) \tag{6.6}$$

where $\mu : \mathbb{R} \to \mathbb{R}$ is a weighting function we define in the sequel. Note that σ depends only on image criteria and the velocity direction $\bar{V}$.

To understand σ, consider the virtual point p_r and its associated flow Ψ_r. Computing the cost function for this scenario one has

$$\begin{aligned}\sigma(\bar{V},t;\bar{\xi},\Psi_r,f,r) &= \frac{|\bar{\xi} - \bar{V}|}{Z}\frac{Z_r}{|\bar{\xi} - \bar{V}|}\mu(Z_r) \\ &= \frac{Z_r}{Z}\mu(Z_r) = \mu(Z_r).\end{aligned}$$

Since $Z_r < Z$ for points outside the cylinder then $\sigma(\bar{\xi},t) < \mu(Z_r)$ for such points. If $\mu(Z_r) = 1$ is set to unity then one has a constant criteria $\sigma(\bar{\xi},t) = 1$ for points lying on the cylinder. If $\mu(Z_r)$ is allowed to vary then $\sigma(\bar{\xi},t) = 1$ can be thought of as a criteria $r(Z_r) = r\mu(Z_r)$

where the virtual point p_r is now considered to lie on a new cylinder of radius $r(Z_r)$ at depth Z_r. In this way, we can generalise the cylinder to any revolute shape as long as a given ray (pixel coordinates $\bar{\xi}$) does not intersect the surface of the surface more than once. In practice, the cylinder of free space works well and is simple to understand and develop.

Let $d_1 \in (0, d)$ be a positive number in the range 0 to d. We will choose a scaling function $\mu : [0, d] \to \mathbb{R}$ as follows

$$\mu(Z_r) = \begin{cases} Z_r \in [0, d_1] & \mu(Z_r) = 1 \\ Z_r \in [d_1, d] & \mu(Z_r) = \frac{d - Z_r}{d - d_1} \end{cases} \tag{6.7}$$

This choice is associated with a constant radius r cylinder with a depth range of $[0, d_1]$ in front of the vehicle followed by a cone that tapers to a point at depth d (cf. Fig. 6.2).

The cost function introduced is a barrier function construction based on keeping $\sigma(\bar{\xi}, t) < 1$. Classical barrier functions are often based on logarithmic growth functions in order that the derivative of the barrier is well conditioned as the barrier conditions is reached. In practice, we found that introducing an infinite barrier function was too aggressive, particularly for noisy flow measurements. As a consequence we have introduced a finite "barrier like" function to scale the indicator function σ.

Define a 'barrier' function $\gamma : [0, 1] \to [0, \infty]$ by

$$\gamma(\sigma) := A\,(e^{\alpha\sigma} - 1) \tag{6.8}$$

for constants $B > 0$, $G > 0$ and where $A = \frac{B}{G-1}$ and $\alpha = \log(G)$. In particular, one has

$$\gamma(0) = 0, \quad \gamma(1) = B, \quad \frac{d}{d\sigma}\gamma(0) = \frac{B(\log(G) - 1)}{G - 1}$$

Thus, the constant B fixes the maximum value of the barrier function while G implicitly fixes the gradient of the cost function at the origin.

The flow based criteria σ (6.6) is combined with the barrier function construction γ (7.8) to generate a total scene cost function $\Gamma : \mathrm{S}^2 \times \mathbb{R} \to \mathbb{R}$ defined by

$$\Gamma(\bar{V}, t) = \int_{\mathbb{R}^2} \gamma(\sigma(\bar{V}, \bar{\xi}; \Psi(\bar{\xi}, t), f, r))\mathrm{d}\bar{\xi} \tag{6.9}$$

where the flow $\Psi(\bar{\xi}, t)$ is an exogenous input to the cost based on the changing local environment as the quadrotor flies through the environment, and f and r are constant parameters. The total scene cost function Γ is minimized for a direction $\bar{V}$ such that the cylinder/cone construction is separated from the environment. In this way we can choose a desired steering direction $\bar{V}$ which steers through the environment - this provides the path planning like aspects of the proposed control. The total scene cost function is maximal when points in the environment approach the cylinder/cone boundary. This provides the natural reactive control properties of flow based criteria.

The proposed control is a steering control for the velocity direction $\bar{V}$. We first compute the gradient of Γ with respect to $\bar{V}$. We divide the image into two regions, Y_1 corresponding to all image points viewed through the walls of the cylinder for the free space construction, and Y_2

corresponding to image point viewed through the conic section of the free space construction Fig. 6.2. Thus, on Y_1 then $\mu(Z_r) = 1$, while on Y_2 then $\mu(Z_r) = (d - Z_r)/(d - d_1)$. The derivative of Γ is given by

$$
\begin{aligned}
&\nabla_{\bar{V}}\Gamma(\bar{V},t) \\
&= \int_{\mathbb{R}^2} \frac{\mathrm{d}}{\mathrm{d}\sigma}\gamma(\sigma(\bar{V},\bar{\xi}))\frac{\mathrm{d}}{\mathrm{d}\bar{V}}\sigma(\bar{V},\bar{\xi})\mathrm{d}\bar{\xi} \\
&= \int_{\mathrm{Y}_1} A\alpha e^{\alpha\sigma(\bar{V},\bar{\xi})} 2rf \frac{|\Psi|}{\mathbf{e}_3^\top C v}\frac{\bar{\xi}-\bar{V}}{|\bar{\xi}-\bar{V}|^4}\mathrm{d}\bar{\xi} \\
&\quad + \int_{\mathrm{Y}_2} A\alpha e^{\alpha\sigma(\bar{V},\bar{\xi})}\frac{rf}{d-d_1}\frac{|\Psi|}{\mathbf{e}_3^\top C v}\frac{\bar{\xi}-\bar{V}}{|\bar{\xi}-\bar{V}|^2} \\
&\qquad\qquad \left(\frac{2d}{|\bar{\xi}-\bar{V}|^2} - \frac{3rf}{|\bar{\xi}-\bar{V}|^3}\right)\mathrm{d}\bar{\xi} \\
&= A\alpha \int_{\mathrm{Y}_1} 2e^{\alpha\sigma}\sigma(\bar{V},\bar{\xi})\frac{\bar{\xi}-\bar{V}}{|\bar{\xi}-\bar{V}|^2}\mathrm{d}\bar{\xi} \\
&\quad + A\alpha \int_{\mathrm{Y}_2} e^{\alpha\sigma}\sigma(\bar{V},\bar{\xi})\frac{\bar{\xi}-\bar{V}}{d-Z_r}\left(\frac{2d}{|\bar{\xi}-\bar{V}|^2} - \frac{3rf}{|\bar{\xi}-\bar{V}|^3}\right)\mathrm{d}\bar{\xi}
\end{aligned}
\tag{6.10}
$$

The negative gradient of Γ indicates the direction in which $\bar{V}$ should be moved and is mapped into a force tangential force input on the motion of the quadrotor. That is we define control input to the quadrotor to be

$$F = \epsilon + mg\mathbf{e}_3 + u_1\mathbf{e}_1$$

where u_1 is a control for speed regulation, $mg\mathbf{e}_3$ compensates for the gravitational term, and ϵ is the obstacle avoidance term

$$\epsilon := -k_\epsilon \nabla\Gamma(\bar{V},t) \tag{6.11}$$

An experiment to test Forward Sensitive Vision Tunnel-based Reactive Optical Flow Controller (6.10) discussed in Chapter 6. The aim was to test to see if the system can detect obstacles such as trees and their small branches and leaves, and then perform reactive evasion using the detected obstacles.

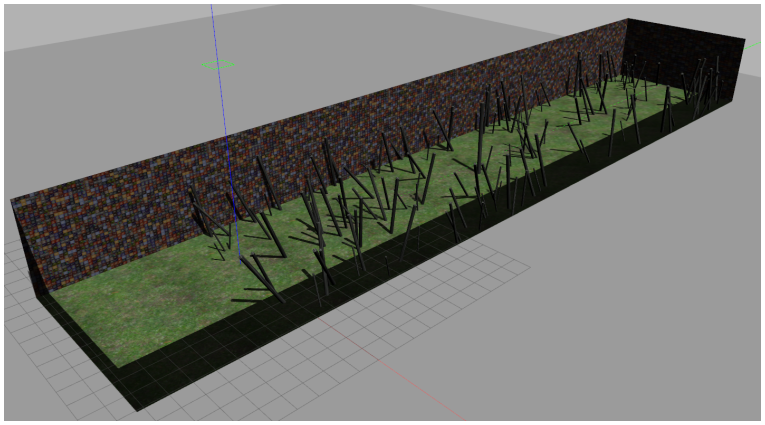

Figure 6.4: Gazebo 7 environment used to test reactive avoidance algorithm. The environment is setup to mimic trees along a path with a boundary. The simulated quadrotor would start in the middle of the open area to the left.

6.2 Experimental Results

This section presents experiments performed to test the forward vision tunnel-based optical flow controller (6.11). Simulations were first conducted to test the theory, followed by real experiments on the design quadrotor platform.

6.2.1 Simulations

Simulations were performed to test whether or not the visual guidance algorithm (6.11) would work as expected. The simulations were used as a proof of concept before applying it to a real quadrotor. The simulation was set up in Ubuntu 16.04, running Robot Operating System (ROS) and the Gazebo physics simulation environment (see Figure 6.4). The environment is set up so that it would be like a corridor in a forest. Textured poles were implemented to act as trees. While in Australian forests trees have a large amount of low-lying branches, this was not implemented due to 3D texture complexity being difficult to create and too computationally expensive to simulate in the Gazebo environment with the available resources.

The results from the simulated experiments showed that the visual guidance algorithm is somewhat successful at performing reactive avoidance. Example results of the output are shown in the Figures 6.5, 6.6, and 6.7.

From the results in Figures 6.5, 6.6, and 6.7, the visual guidance algorithm (6.11) can be seen working. From Figure 6.5, you can see optical flow is dense, with the poles and the floor having optical flow produced. However the floor and the wall were never a threat to the quadrotor's motion, which can be clearly seen in Figure 6.6, where the largest threat is the pole in front. The response of this can be seen in Figure 6.7, where the current velocity (red dot) is being shifted towards the desired velocity (blue dot).

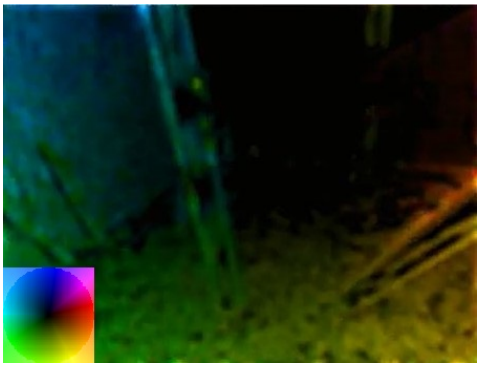

Figure 6.5: Simulation dense optical flow output. The colour wheel at the bottom left represents the colour that refers to the direction and magnitude of the optical flow.

Figure 6.6: Simulation γ (7.8) cost. The brighter pixels refer to higher values of γ 7.8, which mean the points are more of a threat and the controller must respond as such.

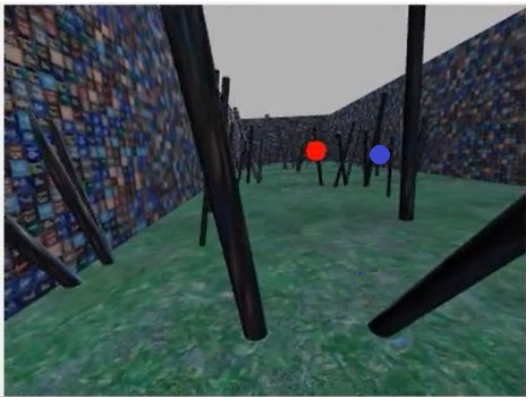

Figure 6.7: Raw simulation output image, via ROS interface. The red dot is the current velocity, while the blue dot is the visual guidance algorithm (6.11) velocity setpoint.

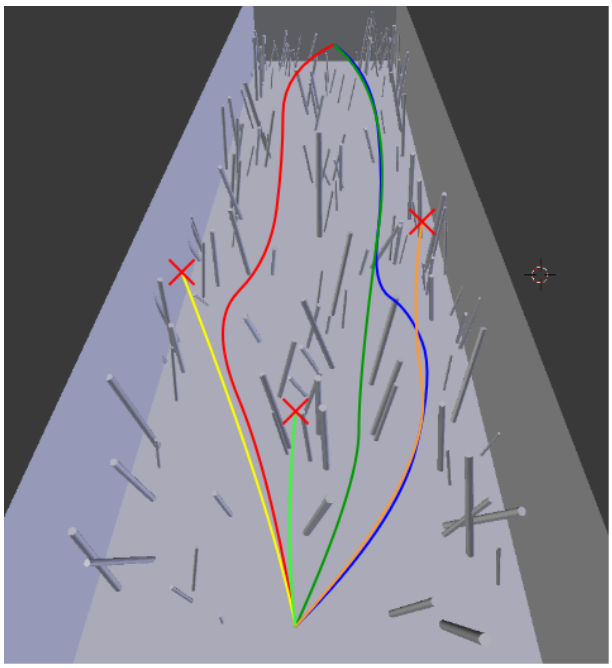

Figure 6.8: Simulation Results: Six paths for the simulated quadrotor. Three paths failed due to entering parts of the environment from which there was no exit.

The paths that were often followed by the simulated quadrotor are as in Figure 6.8, which includes success and failure cases. From multiple trials, the most common paths followed were the red and blue paths. Both paths found corridors of low amounts of poles to go down, however had to make a significant steering correction just before the middle of the environment. The dark green line was the most successful case, where little variation in direction was required, however it was not a common chosen path at first.

Both the yellow and the orange paths were the most common failure cases. In both situations, the visual guidance algorithm got stuck in a local minima between either 2 poles or a pole and a wall. While the visual guidance algorithm often overcame this, once the quadrotor got too close there is no area to avoid collisions. A rarer failure case is the light green line, where the algorithm got stuck between the poles and did not steer when required.

6.2.2 Environmental Settings

The main setting of this experiment was aimed to simulate Australian bushland, which can be defined as:

1. Trees and shrubs are close together.
2. Many large and small branches (<1cm) with leaves in irregular shapes, unlike a pine forest with straight and predictable branch patterns.

Figure 6.9: Reactive obstacle avoidance experiment environment setting

The weather conditions had an impact on performance. Performance was noticeably better during times with large amounts of cloud. Sunny days created large amounts of shadows which cause the camera to have lighting issues, which is a normal issue for non-specialised cameras. Sunny patches appeared extremely bright while shaded patches appeared dark. This inherently reduced the ability for the optical flow algorithm to work correctly, as it reduced texture resolution. Cloudy days reduced the dynamic range of light via constant shade, meaning the camera could be well calibrated to the conditions.

The system performed better during low wind days for two primary reasons. The quadrotor is susceptible to drift due to wind pushing the vehicle around. This could lead to velocity estimates using the aerodynamic modelling (4.5) to have a large error, and the PI velocity controller (4.20) not performing optimally. The second reason is that branches and trees will sway in wind. This can cause the optical flow produced by the branches and trees to be different from when it is stationary, leading to position estimates of the objects having error.

6.2.3 Control Sequence

The experiments follow specific control sequence.

1. The quadrotor with the camera face down computing optical flow, hovers above its launch point using Equations (5.2) and (5.3) for position control.
2. The quadrotor switches the camera to a stabilised face forward mode aiming the camera in direction of expected travel.
3. The quadrotor performs the obstacle avoidance experiment using (6.10) and (6.11).
4. The quadrotor stops and faces the camera down to hover again.
5. The quadrotor then performs a landing, using a constant downwards velocity command and stabilises horizontal position using a zero velocity command rather than (5.3) due to error of optical flow very close to ground leading to heavy sideways movement.

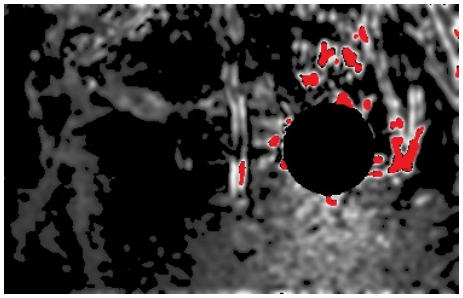

Figure 6.12: Cost Function Output. The dark circle is the area of pixels around $\bar{V}$ that is ignored due to the noise from Ψ. The cost function γ is represented by grey scale where the whiter pixels represents higher values of γ. This scaling is true, except in the case when $\gamma = B$ which is represented by red areas.

6.2.4 Results and Analysis

A scene from a typical experiment is shown in Figures 6.10, 6.11, 6.12, and 6.13.

Figure 6.10: Raw image output of the mvBlueFOX-200w camera at a resolution of 376x240. The grey dot is the vector ${}^{A}\hat{v}$ plotted on the camera frame, while the white dot is the desired velocity vector determined from ϵ.

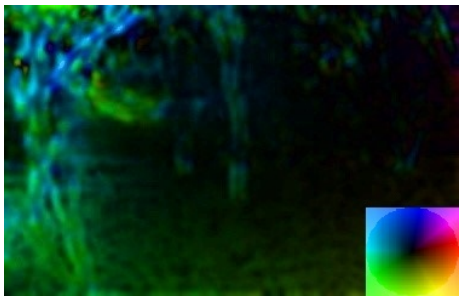

Figure 6.11: Example optical flow output Ψ from the reactive control experiment.

In these figures you can see the effect the cost function has on determining whether or not an object is a threat. Comparing the optical flow output Fig. 6.11 and the cost function output Fig. 6.12, one can see the cost function reduces the influence of objects that are not a threat to the control. The tree on the left of the images has a very large optical flow output, however

Figure 6.13: Flight path taken by the quadrotor by an experiment over a distance of 12 meters. The yellow path is the GPS-tracked flight path, and the blue dots are the approximate tree (obstacles) locations.

the cost function does not respond to the flow since it is in a zone that will not intersect the vehicle's motion. However, the trees slightly to the left and right of the grey dot $\bar{V}$ are given a stronger cost even though they have less optical flow.

Although the algorithm is providing a level of look ahead for the vehicle, the quadrotor can still become stuck if it flies into an open space which then narrows and traps the vehicle. An example of a failure case can be found in Appendix B - Vision Based Forward Sensitive Reactive Control for a Quadrotor VTOL Failure Case. It is a straightforward matter to add an additional control term for the forward velocity of the vehicle that regulates the speed to zero in the case where the environment encroaches on the free space cylinder. Of course, as the speed of the vehicle decreases the flow will decrease also allowing the vehicle to squeeze through narrow spaces[1].

[1] Video of the 'Vision based forward sensitive reactive control for a quadrotor VTOL' can be found via https://www.youtube.com/watch?v=mwuqyz9cEEA

Chapter 7

Conclusion and Future Work

This chapter summarises the major achievements of the thesis and describe possible future research.

7.1 Conclusion

This thesis described a quadrotor system for visual navigation in complex and dense environments. It was designed to meet a set of requirements consistent with functionality required for future UAV applications. The quadrotor platform achieved completely autonomous flight in complex and dense environments, and also demonstrated the ability to hover using only visual cues. It was demonstrated that an effective manual function was in place. The quadrotor used primarily off-the-shelf components for repeatability and cost saving. The quadrotor was fitted with the proposed camera and on-board computational capability, which was used with a visual guidance system. The quadrotor with the complete assembly described in this thesis met the Australian Government Civil Aviation Safety Authority (CASA) under 2kg commercial regulations regarding licencing.

The 5 primary goals of this thesis were to demonstrate that:

1. A design of a quadrotor system for utility aerial robotic experiments using off-the-shelf parts: This was achieved by using a hobby quadrotor kit, using the Pixhawk 2.1 open-source flight controller and software, using the commercially available NVIDIA Tegra TX2 with Ubuntu (Linux) operation system, 3-axis hobby gimbal, and the Matrix-Vision mvBlueFOX-200w camera. These were all off-the-shelf components, however still using the latest mobilised computational architecture design for robotics.

2. Software architecture and on-board computational hardware for utility aerial robotics experimental work: This was achieved by using the open-source PX4 code for the Pixhawk 2.1 flight controller and the NVIDIA Jetpack software, a variation of Ubuntu 16.04, for the NVIDIA Tegra TX2. The state of the quadrotor, including attitude and velocity, was estimated on the Pixhawk 2.1. The Pixhawk 2.1 was also used to convert autonomous (from the TX2) or manual commands into motor set-points, used by the motor controllers to control rotor speed. The visual guidance system was done on the NVIDIA Tegra TX2 using the input from the Matrix-Vision mvBlueFOX-200w camera and trasmitted to the Pixhawk 2.1 for control.

3. Sensor fusion for quadrotor state estimation: The state of the quadrotor, including attitude and velocity, was estimated on the Pixhawk 2.1 using its on-board inertial measurement unit (IMU) and its HERE2 GNSS (GPS) for position measurements. A customised velocity filter, using Lyapunov stability theory, was used to get accelerometer measurements (from the IMU), GPS position estimates, and aerodynamic velocity estimates to get a real-time filtered velocity estimate. Additionally, a customised hardware filtering system was designed to reduce noise picked up by the IMU due to the motors, meaning IMU measurements were more reliable.

4. Implementation of image based hover and flight control: Using the NVIDIA Tegra TX2 to get camera inputs and then calculating optical flow in real-time, a position estimate was calculated. Additionally, with a known landmark such as specific tree or an ArUco marker, the position estimate could be compensated for drift. The NVIDIA Tegra TX2 would then use a PI controller to give velocity commands to the Pixhawk 2.1 for flight control.

5. A novel new obstacle avoidance algorithm for forward looking reactive control using optical flow: Using the dense optical flow algorithm and velocity measurements, points in 3D space in the direction of the quadrotor were determined whether they were a threat or not to the quadrotor's motion. A cylinder in 3D space was projected onto the camera plane, which determined what points were a threat or not. The cylinder was then steered to a minimum, to perform forward looking reactive control. This algorithm is novel as it predicts future possible collisions, other than just reacting purely to optical flow.

7.2 Future Work

The results of this thesis suggest possible research extensions to improve the performance of the system.

1. **Optical Flow:** Develop a dense optical flow algorithm which can run on mobilised computational hardware, such as the NVIDIA Tegra TX2 or an FPGA, which is capable of having the rotational component of the optical flow removed. This would allow the removal of the 3-axis gimbal which contributes to a significant weight increase in the system, while also allowing the quadrotor to rotate without have to stop and move the gimbal.

2. **Event Camera:** Using an event camera to estimate optical flow would provide the benefit of extreme dynamic range, allowing the quadrotor to fly during the night or in cave-like environments. The event camera would also provide the benefit of real-time control as the data can come from the event camera asynchronously, which could remove the control delay due to having to process individual frames.

3. **Completely Customised Quadrotor:** A customised quadrotor using customised parts could reduce the system weight and make the system more compact. This would increase the performance of the system, allowing it to fly through smaller gaps and extend flight

time. While this would provide a weight and size benefit, it would reduce the ability to reproduce and potentially increase the costs of the system as it is no longer using off-the-shelf parts.

4. **Image Warp Conditioning:** In high FOV camera, there is a significant lens distortion causing a warped image present. Removing this would increase system performance, however it is computationally expensive to do so. Having a system on the camera such as an FPGA could remove it when the image is picked up by the camera, the transmitting the un-warped image to the main computational device.

5. **Speed Set-points:** Add a control term that sets the maximum speed of the quadrotor dependant on the amounts of threats detected. This means the quadrotor will have more time to react to targets, and steer more effectively. It also means that the quadrotor would speed up when there is no real threats.

Appendix A - Detailed derivation of Vision Based Forward Sensitive Reactive Control for a Quadrotor VTOL

Current optical flow controlled quadrotors directly use optical flow to steer. However, with the known velocity of the vehicle ${}^{B}\hat{v}$, we can project a cylinder with a cone end into free space ξ, which is mapped onto the camera frame $\bar{\xi}$. This tunnel is in the direction of the vector of ${}^{B}v$. Using the cylinder method, it is possible to predict a collision with a point using the translational optical flow Ψ, while ignoring or reducing the significance of optical flow points that will not cause a collision even if the optical flow vector is large.

7.3 Mapping the velocity ${}^{B}v$ onto the camera frame

The first part required is mapping the velocity ${}^{B}\hat{v}$ onto the camera plane $\bar{\psi}$. This mapping $\bar{V}$ can be achieved using (7.1):

$$\bar{V} = \left(\frac{f\,{}^{C}v_x}{{}^{C}v_z}, \frac{f\,{}^{C}v_y}{{}^{C}v_z} \right) \tag{7.1}$$

Note that ${}^{C}v = R_{\text{cam}}\,{}^{B}v$ is the vehicle velocity written in the camera frame and R_{cam} is the rotation of the camera frame with respect to the body-fixed-frame ${}^{B}\hat{v}$.

7.4 Notation for Cost Function σ Definition

The cost function proposed σ is used to derive the steering controller ϵ. Figure 7.1 shows the characteristics of the cylinder with a cone end that the steering controller ϵ is based on. d_1 is the distance (in metres) from the quadrotor to the cone's starting point, while d is the total length of the cylinder with the cone end. r is the desired radius of the proposed cylinder. r should be slightly larger than the maximum distance between any point on the quadrotor, to maximise the chance of the quadrotor not colliding with a point.

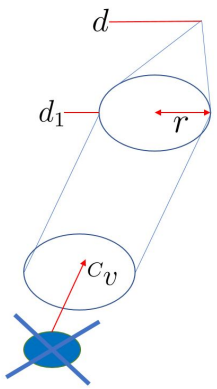

Figure 7.1: Cylinder-Cone Visualisation and Terminology

Figure 7.2 shows the mapping of the projected cyclinder with cone end and optical flow onto the image plane. $D_{cylinder}$ and D_{cone} are the domains in which the cylinder and cone appear on the camera frame. Φ is the optical flow vector in its continuous form (Φ is generally in its time-based form, which is the measure of pixel motion over a period of time). $\bar{\xi}$ is a pixel coordinate on the image plane and $\bar{V}$ is the pixel coordinate that the velocity appears travelling towards on the image plane. $\bar{\kappa}$ is the magnitude of the distance from $\bar{\xi}$ to $\bar{V}$. As such, the derivative $\dot{\bar{\kappa}}$ of $\bar{\kappa}$ is the continuous form of the optical flow vector.

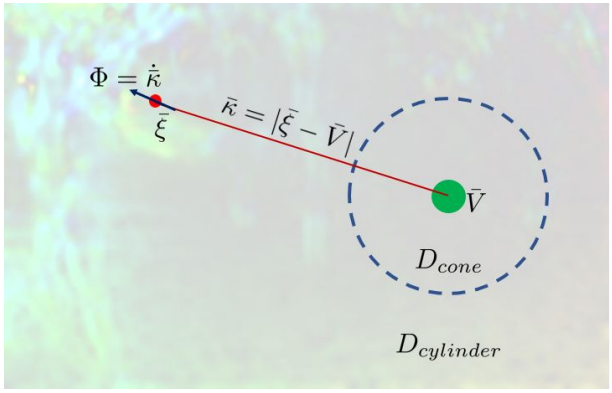

Figure 7.2: Optical Flow and Camera Terminology

7.5 Cost Function σ Definition

A dimensionless cost function σ is developed to give an indication of whether or not a point on the image place $\bar{\xi}$ is a threat to the quadrotors motion. Let z_r denote the depth of the point ξ_r on the cylinder wall. This cost function is based of the projection of the cylinder with cone end (7.2):

$$\sigma(\bar{V}, t; \bar{\xi}, \Psi(\xi, t), f, r) = \frac{|\Psi(\bar{\xi}, t)|}{\mathbf{e}_3^\top {}^C v} \frac{rf}{|\bar{\xi} - \bar{V}|^2} \mu(z_r) \tag{7.2}$$

This is where $\mu(z_r)$ is based on the cylinder and cone shape (7.3), and z_r is the distance in $\mathbf{e}_3$ from the quadrotor to a point:

$$\mu(z_r) = \begin{cases} z_r \in [0, d_1] & \mu(z_r) = 1 \\ z_r \in [d_1, d] & \mu(z_r) = \frac{d - z_r}{d - d_1} \end{cases} \tag{7.3}$$

7.6 Cost Function σ Derivation

The cost function σ is developed from the properties of similar triangles, projecting on to the cylinder wall of radius r. A point in space $p \in \mathbb{R}^3$ can be projected onto the camera plane $\bar{\xi}$ using the properties of similar triangles as shown in Figure 7.3. Using the properties of similar triangles we get (7.4):

$$\frac{z_r}{r} = \frac{f}{\bar{\kappa}} \tag{7.4}$$

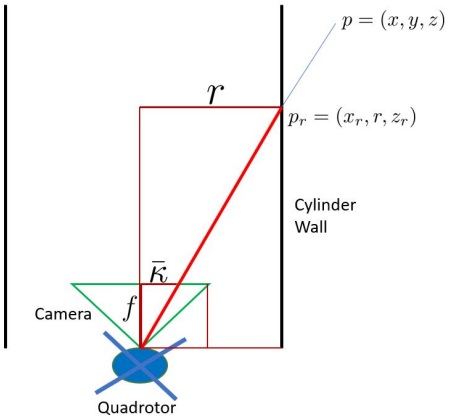

Figure 7.3: Properties of similar triangles, based on the cylinder

Where $\bar{\kappa} = |\bar{\xi} - \bar{V}|$ is the relative position on the pixel $\bar{\xi}$ compared to the mapped velocity $\bar{V}$. Rearranging (7.4) and taking the derivative of $\bar{\kappa}$ we get (7.5):

$$\dot{\bar{\kappa}} = -rf\frac{\dot{z_r}}{z_r^2} \tag{7.5}$$

Substituting $z_r = \frac{rf}{\bar{\kappa}}$ (7.4) into (7.5) we get :

$$\dot{\bar{\kappa}} = -\frac{\bar{\kappa}^2}{rf}\dot{z_r} \tag{7.6}$$

The quadrotor's velocity in the camera frame ${}^{C}v$ is travelling to the point $\bar{V}$, we can take the velocity in the z-direction of the camera frame is the same as equivalent to the velocity $\mathbf{e}_3{}^{C}v = -\dot{z_r}$. $\dot{\bar{\kappa}}$ is the normalised translational flow Ψ over time (pixels per second). From this

we get (7.7):

$$\frac{|\Psi(\xi,t)|}{\mathbf{e}_3^\top C v}\frac{rf}{|\bar{\xi}-\bar{V}|^2}=1 \tag{7.7}$$

σ (7.2) is essentially a function of the variation of the measured relationship $\frac{|\Psi(\xi,t)|}{\mathbf{e}_3^\top C v}$.

7.7 Cost Function γ

σ (7.2) however is not effective enough to be used to steer the quadrotor. Since points outside the cylinder wall are not a major problem, we can remap σ on to the 'barrier' cost function γ so that different points can be scaled. Defining the 'barrier' cost function $\gamma : [0,1] \to [0,\infty]$ (7.8):

$$\gamma(\sigma) := A\left(e^{\alpha\sigma}-1\right) \tag{7.8}$$

for constants $B>0$, $G>0$ and where $A=\frac{B}{G-1}$ and $\alpha=\log(G)$. In particular, one has

$$\gamma(0)=0, \quad \gamma(1)=B, \quad \frac{d}{d\sigma}\gamma(0)=\frac{B(\log(G)-1)}{G-1}$$

Thus, the constant B fixes the maximum value of the barrier function while G implicitly fixes the gradient of the cost function at the origin. γ can be visualised in Figure 7.4:

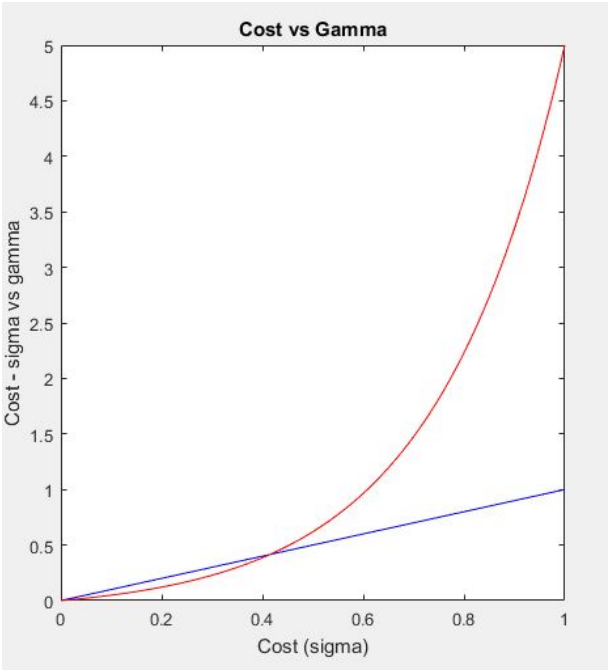

Figure 7.4: γ remap

7.8 Total Scene Cost Γ and its derivative $\nabla\Gamma$ defintion and derivation

To achieve the control we need to look at an integration $\Gamma(\bar{V},t)$ over all the surface of the image plane $\bar{\xi}$ (7.9).

$$\Gamma(\bar{V},t) = \int_{\mathbb{R}^2} \gamma(\sigma(\bar{V},\xi;\Psi(\xi,t),f,r))\mathrm{d}\xi \tag{7.9}$$

For steering control ϵ based on the direction of $\bar{V}$, we use the derivative of $\Gamma(\bar{V},t)$ as defined in (7.10):

$$\nabla\Gamma(\bar{V},t) = \int_{\mathbb{R}^2} \frac{\mathrm{d}}{\mathrm{d}\sigma}\gamma(\sigma(\bar{V},\xi))\frac{\mathrm{d}}{\mathrm{d}\bar{V}}\sigma(\bar{V},\xi)\mathrm{d}\xi \tag{7.10}$$

For simplicity, we split $\nabla\Gamma$ into its cylinder $\nabla\Gamma_{cyl}$ and cone $\nabla\Gamma_{co}$ components such that (7.11) is formed:

$$\nabla\Gamma = \nabla\Gamma_{cyl} + \nabla\Gamma_{co} \tag{7.11}$$

This is where $\nabla\Gamma_{cyl}$ and $\nabla\Gamma_{co}$ are defined as:

$$\nabla\Gamma_{cyl} = A\alpha \int_{D_{cyl}} 2e^{\alpha\sigma}\sigma\frac{\bar{\xi}-\bar{V}}{|\bar{\xi}-\bar{V}|^2}\mathrm{d}\bar{\xi} \tag{7.12}$$

$$\nabla\Gamma_{co} = A\alpha \int_{D_{co}} e^{\alpha\sigma}\sigma\frac{\bar{\xi}-\bar{V}}{d-d_1}\left(\frac{2d}{|\bar{\xi}-\bar{V}|^2} - \frac{3rf}{|\bar{\xi}-\bar{V}|^3}\right)\mathrm{d}\bar{\xi} \tag{7.13}$$

The following process to form $\nabla\Gamma_{cyl}$ and $\nabla\Gamma_{co}$, we need to find out the components for each. Since $\frac{\mathrm{d}}{\mathrm{d}\sigma}\gamma$ is independent of variations of σ, $\frac{\mathrm{d}}{\mathrm{d}\sigma}\gamma$ is determined (7.14):

$$\frac{\mathrm{d}}{\mathrm{d}\sigma}\gamma = A\alpha e^{\alpha\sigma} \tag{7.14}$$

For the domain of the cylinder we can compute $\frac{\mathrm{d}}{\mathrm{d}\bar{V}}\sigma_{cyl}$ via this sequence:

$$\sigma_{cyl} = \frac{|\Psi|}{\mathbf{e}_3^\top C v}\frac{rf}{|\bar{\xi}-\bar{V}|^2} \tag{7.15a}$$

$$\frac{\mathrm{d}}{\mathrm{d}\bar{V}}\sigma_{cyl} = 2rf\frac{|\Psi|}{\mathbf{e}_3^\top C v}\frac{1}{|\bar{\xi}-\bar{V}|^3} \tag{7.15b}$$

$$= 2rf\frac{|\Psi|}{\mathbf{e}_3^\top C v}\frac{\bar{\xi}-\bar{V}}{|\bar{\xi}-\bar{V}|^4} \quad \text{for direction} \tag{7.15c}$$

$$= 2\sigma_{cyl}\frac{\bar{\xi}-\bar{V}}{|\bar{\xi}-\bar{V}|^2} \tag{7.15d}$$

For the domain of the cone we can compute σ_{co} in its expanded form via this sequence:

$$\sigma_{co} = \frac{|\Psi|}{\mathbf{e}_3^\top C v} \frac{rf}{|\bar{\xi} - \bar{V}|^2} \frac{d - z_r}{d - d_1} \tag{7.16a}$$

$$= \frac{|\Psi|}{\mathbf{e}_3^\top C v} \frac{rf}{|\bar{\xi} - \bar{V}|^2} \frac{d - \frac{rf}{|\bar{\xi} - \bar{V}|}}{d - d_1} \tag{7.16b}$$

$$= \frac{rf}{d - d_1} \frac{|\Psi|}{\mathbf{e}_3^\top C v} \left(\frac{d}{|\bar{\xi} - \bar{V}|^2} - \frac{rf}{|\bar{\xi} - \bar{V}|^3} \right) \tag{7.16c}$$

Then we can compute $\frac{\mathrm{d}}{\mathrm{d}\bar{V}} \sigma_{co}$ via this sequence:

$$\frac{\mathrm{d}}{\mathrm{d}\bar{V}} \sigma_{co} \tag{7.17a}$$

$$= \frac{rf}{d - d_1} \frac{|\Psi|}{\mathbf{e}_3^\top C v} \left(\frac{2d}{|\bar{\xi} - \bar{V}|^3} - \frac{3rf}{|\bar{\xi} - \bar{V}|^4} \right) \tag{7.17b}$$

$$= \frac{rf}{d - d_1} \frac{|\Psi|}{\mathbf{e}_3^\top C v} \frac{\bar{\xi} - \bar{V}}{|\bar{\xi} - \bar{V}|^2} \left(\frac{2d}{|\bar{\xi} - \bar{V}|^2} - \frac{3rf}{|\bar{\xi} - \bar{V}|^3} \right) \quad \text{for direction} \tag{7.17c}$$

$$= \frac{\bar{\xi} - \bar{V}}{d - d_1} \sigma_{cyl} \left(\frac{2d}{|\bar{\xi} - \bar{V}|^2} - \frac{3rf}{|\bar{\xi} - \bar{V}|^3} \right) \tag{7.17d}$$

Since we now have $\frac{\mathrm{d}}{\mathrm{d}\sigma}\gamma$, $\frac{\mathrm{d}}{\mathrm{d}\bar{V}}\sigma_{cyl}$, and $\frac{\mathrm{d}}{\mathrm{d}\bar{V}}\sigma_{co}$, we can compute $\nabla\Gamma_{cyl}$ and $\nabla\Gamma_{co}$ with the results shown in (7.12) and (7.13) respectively.

7.9 Steering Control ϵ

Using (7.10), we can calculate our steering controller ϵ with a gain k_ϵ as (7.18):

$$\epsilon = -k_\epsilon \nabla\Gamma(\bar{V}, t) \tag{7.18}$$

ϵ is used as a force controller F_d component that drives the quadrotor in $\{B\}$. This controller is described in (7.19):

$$F_d = \epsilon \mathbf{e}_2 + mg\mathbf{e}_3 + u_1\mathbf{e}_1 \tag{7.19}$$

Where u_1 is a desired forward force controller.

Appendix B - Vision Based Forward Sensitive Reactive Control for a Quadrotor VTOL Failure Case

The most common failure case observed during the main experiments occurs when the cost function has an equal distribution across the image around $\bar{V}$ (7.1). This can either be when there are 2 border trees causing the algorithm to navigate in between the two but they are too tight to fit between, or when the algorithm navigates towards the centre of a tree causing the total scene cost $\Gamma(\bar{V}, t)$ (7.9) to have no significant control value due the spread of the tree and its affect on the optical flow. Figures 7.5, 7.6, and 7.7 are an example output of when the costs on both sides of $\bar{V}$ (7.1) is near equal which causes $\Gamma(\bar{V}, t)$ (7.9) to be near zero when moving towards the centre of a tree.

To deal with this situation, an additional control term is necessary to increase or decrease the velocity of the quadrotor, or to move it out of the path. This was not explored during this thesis.

Figure 7.5: Raw image from failed reactive control experiment.

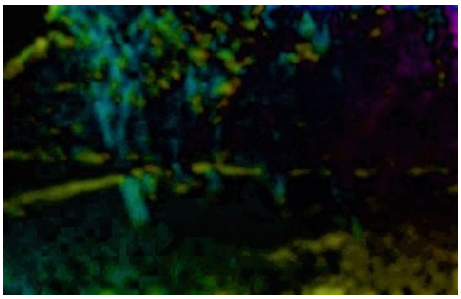

Figure 7.6: Optical flow image from failed reactive control experiment.

Figure 7.7: Cost output from failed reactive control experiment.

Bibliography

ACHTELIK, M.; BACHRACH, A.; HE, R.; PRENTICE, S.; AND ROY, N., 2009. Autonomous navigation and exploration of a quadrotor helicopter in gps-denied indoor environments. In *First Symposium on Indoor Flight*, 2009. Citeseer. (cited on pages 3, 9, 12, and 13)

ADARVE, J. D. AND MAHONY, R., 2016. A filter formulation for computing real time optical flow. *IEEE Robotics and Automation Letters*, 1, 2 (July 2016), 1192–1199. doi:10.1109/LRA.2016.2532928. (cited on pages 3, 10, 11, 13, 33, 34, 35, and 47)

ALPEN, M.; WILLRODT, C.; FRICK, K.; AND HORN, J., 2010. On-board slam for indoor uav using a laser range finder. In *Unmanned Systems Technology XII*, vol. 7692, 769213. International Society for Optics and Photonics. (cited on pages 3, 9, and 13)

ALTUG, E.; OSTROWSKI, J. P.; AND MAHONY, R., 2002. Control of a quadrotor helicopter using visual feedback. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 1, 72–77. IEEE. (cited on pages 6, 8, and 13)

BANGURA, M.; KUIPERS, F.; ALLIBERT, G.; AND MAHONY, R., 2015. Non-linear velocity aided attitude estimation and velocity control for quadrotors. In *Proceedings of the Australian Conference for Automation and Robotics*. (cited on pages 9, 30, and 31)

BANGURA, M.; MAHONY, R.; LIM, H.; AND KIM, H. J., 2014. An open-source implementation of a unit quaternion based attitude and trajectory tracking for quadrotors. In *Proceedings of the Australasian Conference on Robotics and Automation, Melbourne, Australia*, 2–4. (cited on page 8)

BANGURA, M.; MAHONY, R.; ET AL., 2012. Nonlinear dynamic modeling for high performance control of a quadrotor. In *Australasian conference on robotics and automation*, 1–10. (cited on page 30)

BHAGAVATULA, P. S.; CLAUDIANOS, C.; IBBOTSON, M. R.; AND SRINIVASAN, M. V., 2011. Optic flow cues guide flight in birds. *Current Biology*, 21, 21 (2011), 1794–1799. (cited on pages 12 and 13)

BLOESCH, M.; BURRI, M.; OMARI, S.; HUTTER, M.; AND SIEGWART, R., 2017. Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback. *The International Journal of Robotics Research*, 36, 10 (2017), 1053–1072. (cited on pages 8, 9, and 13)

BLOESCH, M.; OMARI, S.; HUTTER, M.; AND SIEGWART, R., 2015. Robust visual inertial odometry using a direct ekf-based approach. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, 298–304. IEEE. (cited on pages 8, 9, and 13)

BLÖSCH, M.; WEISS, S.; SCARAMUZZA, D.; AND SIEGWART, R., 2010. Vision based mav navigation in unknown and unstructured environments. In *2010 IEEE International Conference on Robotics and Automation*, 21–28. doi:10.1109/ROBOT.2010.5509920. (cited on pages 3, 9, and 13)

BORENSTEIN, J., 1992. The hoverbot–an electrically powered flying robot. *Unpublished white paper, University of Michigan, Ann Arbor, MI. Available FTP: ftp://ftp. eecs. umich. edu/people/johannb/paper99. pdf*, (1992). (cited on page 6)

BOUGUET, J.-Y., 2001. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5, 1-10 (2001), 4. (cited on page 10)

BRESCIANINI, D.; HEHN, M.; AND D'ANDREA, R., 2013. Quadrocopter pole acrobatics. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, 3472–3479. IEEE. (cited on page 9)

BRIOD, A.; ZUFFEREY, J.-C.; AND FLOREANO, D., 2016. A method for ego-motion estimation in micro-hovering platforms flying in very cluttered environments. *Autonomous Robots*, 40 (06 2016). doi:10.1007/s10514-015-9494-4. (cited on page 11)

CONROY, J.; GREMILLION, G.; RANGANATHAN, B.; AND HUMBERT, J. S., 2009. Implementation of wide-field integration of optic flow for autonomous quadrotor navigation. *Autonomous robots*, 27, 3 (2009), 189. (cited on pages 3, 12, and 13)

COOMBS, D.; HERMAN, M.; HONG, T.-H.; AND NASHMAN, M., 1998. Real-time obstacle avoidance using central flow divergence, and peripheral flow. *IEEE Transactions on Robotics and Automation*, 14, 1 (1998), 49–59. (cited on page 11)

DOSOVITSKIY, A.; FISCHER, P.; ILG, E.; HÄUSSER, P.; HAZIRBAS, C.; GOLKOV, V.; V. D. SMAGT, P.; CREMERS, D.; AND BROX, T., 2015. Flownet: Learning optical flow with convolutional networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, 2758–2766. doi:10.1109/ICCV.2015.316. (cited on page 11)

ERESEN, A.; İMAMOĞLU, N.; AND EFE, M. Ö., 2012. Autonomous quadrotor flight with vision-based obstacle avoidance in virtual environment. *Expert Systems with Applications*, 39, 1 (2012), 894–905. (cited on pages 3, 12, and 13)

FAN, L.; LAI, J.; LYU, P.; AND YUAN, C., 2018. Visual path following method for quadrotors based on structured edge detection*. In *2018 IEEE CSAA Guidance, Navigation and Control Conference (CGNCC)*, 1–6. doi:10.1109/GNCC42960.2018.9019154. (cited on page 3)

FORSTER, C.; LYNEN, S.; KNEIP, L.; AND SCARAMUZZA, D., 2013. Collaborative monocular slam with multiple micro aerial vehicles. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3962–3970. doi:10.1109/IROS.2013.6696923. (cited on pages 3, 9, and 13)

FRAUNDORFER, F.; HENG, L.; HONEGGER, D.; LEE, G. H.; MEIER, L.; TANSKANEN, P.; AND POLLEFEYS, M., 2012. Vision-based autonomous mapping and exploration using a quadrotor mav. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 4557–4564. IEEE. (cited on pages 3, 9, 12, and 13)

FU, C.; OLIVARES-MENDEZ, M. A.; SUAREZ-FERNANDEZ, R.; AND CAMPOY, P., 2014. Monocular visual-inertial slam-based collision avoidance strategy for fail-safe uav using fuzzy logic controllers. *Journal of intelligent & robotic systems*, 73, 1-4 (2014), 513–533. (cited on pages 3, 9, and 13)

GRABE, V.; BÜLTHOFF, H. H.; AND GIORDANO, P. R., 2012. On-board velocity estimation and closed-loop control of a quadrotor uav based on optical flow. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 491–497. IEEE. (cited on pages 3, 11, and 12)

HAMEL, T.; MAHONY, R.; LOZANO, R.; AND OSTROWSKI, J., 2002. Dynamic modelling and configuration stabilization for an x4-flyer. *IFAC Proceedings Volumes*, 35, 1 (2002), 217 – 222. doi:https://doi.org/10.3182/20020721-6-ES-1901.00848. http://www.sciencedirect.com/science/article/pii/S1474667015392697. 15th IFAC World Congress. (cited on pages 6, 8, 13, and 30)

HERISSE, B.; HAMEL, T.; MAHONY, R.; AND RUSSOTTO, F. X., 2012. Landing a vtol unmanned aerial vehicle on a moving platform using optical flow. *IEEE Transactions on Robotics*, 28, 1 (Feb 2012), 77–89. doi:10.1109/TRO.2011.2163435. (cited on pages 3, 11, 12, 13, 35, and 41)

HO, H. W.; DE CROON, G. C. H. E.; VAN KAMPEN, E.; CHU, Q. P.; AND MULDER, M., 2018. Adaptive gain control strategy for constant optical flow divergence landing. *IEEE Transactions on Robotics*, 34, 2 (2018), 508–516. doi:10.1109/TRO.2018.2817418. (cited on page 3)

HONEGGER, D.; MEIER, L.; TANSKANEN, P.; AND POLLEFEYS, M., 2013. An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 1736–1741. IEEE. (cited on pages 3, 11, 12, and 34)

HORN, B. K. AND SCHUNCK, B. G., 1981. Determining optical flow. *Artificial intelligence*, 17, 1-3 (1981), 185–203. (cited on pages 10, 13, and 33)

HUANG, A. S.; BACHRACH, A.; HENRY, P.; KRAININ, M.; MATURANA, D.; FOX, D.; AND ROY, N., 2017. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Robotics Research*, 235–252. Springer. (cited on pages 3, 9, and 13)

ILG, E.; MAYER, N.; SAIKIA, T.; KEUPER, M.; DOSOVITSKIY, A.; AND BROX, T., 2017. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE conference on computer vision and pattern recognition (CVPR)*, vol. 2, 6. (cited on page 11)

KHALIL, H. K., 1996. *Nonlinear Systems*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edn. (cited on page 32)

KROO, I.; PRINZ, F.; SHANTZ, M.; KUNZ, P.; FAY, G.; CHENG, S.; FABIAN, T.; AND PARTRIDGE, C., 2000. The mesicopter: A miniature rotorcraft concept phase ii interim report. *Stanford university*, (2000). (cited on page 6)

LAI, S.; LAN, M.; AND CHEN, B. M., 2018. Efficient safe corridor navigation with jerk limited trajectory for quadrotors. In *2018 37th Chinese Control Conference (CCC)*, 10065–10070. doi:10.23919/ChiCC.2018.8483213. (cited on pages 12 and 13)

LEISHMAN, J. G., 2002. The breguet-richet quad-rotor helicopter of 1907. *Vertiflite*, 47, 3 (2002), 58–60. (cited on page 5)

LIM, H.; PARK, J.; LEE, D.; AND KIM, H. J., 2012. Build your own quadrotor: Open-source projects on unmanned aerial vehicles. *IEEE Robotics Automation Magazine*, 19, 3 (Sept 2012), 33–45. doi:10.1109/MRA.2012.2205629. (cited on page 8)

LUCAS, B. D.; KANADE, T.; ET AL., 1981. An iterative image registration technique with an application to stereo vision. (1981). (cited on pages 10 and 13)

MCCARTHY, C.; BARNES, N.; AND MAHONY, R., 2008. A robust docking strategy for a mobile robot using flow field divergence. *IEEE Transactions on Robotics*, 24, 4 (Aug 2008), 832–842. doi:10.1109/TRO.2008.926871. (cited on pages 3, 12, and 13)

MCGUIRE, K.; DE CROON, G.; WAGTER, C. D.; TUYLS, K.; AND KAPPEN, H., 2017. Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone. *IEEE Robotics and Automation Letters*, 2, 2 (April 2017), 1070–1076. doi:10.1109/LRA.2017.2658940. (cited on page 11)

MOHTA, K.; SUN, K.; LIU, S.; WATTERSON, M.; PFROMMER, B.; SVACHA, J.; MULGAONKAR, Y.; TAYLOR, C. J.; AND KUMAR, V., 2018. Experiments in fast, autonomous, gps-denied quadrotor flight. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 7832–7839. doi:10.1109/ICRA.2018.8463214. (cited on pages 9 and 13)

PENG, X. Z.; LIN, H. Y.; AND DAI, J. M., 2016. Path planning and obstacle avoidance for vision guided quadrotor uav navigation. In *2016 12th IEEE International Conference on Control and Automation (ICCA)*, 984–989. doi:10.1109/ICCA.2016.7505408. (cited on page 11)

PESZOR, D.; WOJCIECHOWSKA, M.; WOJCIECHOWSKI, K.; AND SZENDER, M., 2017. Fast moving uav collision avoidance using optical flow and stereovision. In *Lecture notes in computer science: Intelligent information and database systems*, 572–581. Springer. (cited on pages 3, 12, and 13)

POUNDS, P.; MAHONY, R.; AND CORKE, P., 2010. Modelling and control of a large quadrotor robot. *Control Engineering Practice*, 18, 7 (2010), 691–699. (cited on page 6)

POUNDS, P. E. I. ET AL., 2007. Design, construction and control of a large quadrotor micro air vehicle. (2007). (cited on page 6)

RANJAN, A. AND BLACK, M. J., 2017. Optical flow estimation using a spatial pyramid network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2, 2. IEEE. (cited on page 11)

REN, Z.; YAN, J.; NI, B.; LIU, B.; YANG, X.; AND ZHA, H., 2017. Unsupervised deep learning for optical flow estimation. In *AAAI*, vol. 3, 7. (cited on page 11)

SCHAUB, A.; BAUMGARTNER, D.; AND BURSCHKA, D., 2017. Reactive obstacle avoidance for highly maneuverable vehicles based on a two-stage optical flow clustering. *IEEE Transactions on Intelligent Transportation Systems*, 18, 8 (Aug 2017), 2137–2152. doi: 10.1109/TITS.2016.2633292. (cited on pages 3, 9, and 13)

SCHAUWECKER, K. AND ZELL, A., 2013. On-board dual-stereo-vision for autonomous quadrotor navigation. In *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, 333–342. IEEE. (cited on pages 3, 9, and 13)

SERRA, P.; CUNHA, R.; HAMEL, T.; CABECINHAS, D.; AND SILVESTRE, C., 2016. Landing of a quadrotor on a moving target using dynamic image-based visual servo control. *IEEE Transactions on Robotics*, 32, 6 (2016), 1524–1535. (cited on pages 3, 11, 12, and 13)

SRINIVASAN, M.; CHAHL, J.; K.WEBER; S.VENKATESH; M.G.NAGLE; AND S.W.ZHANG, 1999. Robot navigation inspired by principles of insect vision. *Robotics and Autonomous Systems*, 26, 2–3 (1999), 203–216. (cited on pages 11, 12, and 13)

SRINIVASAN, M.; ZHANG, S.; LEHRER, M.; AND COLLETT, T., 1996. Honeybee navigation en route to the goal: visual flight control and odometry. *Journal of Experimental Biology*, 199, 1 (1996), 237–244. (cited on pages 12 and 13)

STEVENS, J. AND MAHONY, R., 2018. Vision based forward sensitive reactive control for a quadrotor vtol. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5232–5238. doi:10.1109/IROS.2018.8593606. (cited on pages 4 and 47)

VALENTI, R. G.; DRYANOVSKI, I.; JARAMILLO, C.; STRÖM, D. P.; AND XIAO, J., 2014. Autonomous quadrotor flight using onboard rgb-d visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, 5233–5238. IEEE. (cited on pages 3, 9, and 13)

WARREN, W. H.; KAY, B. A.; ZOSH, W. D.; DUCHON, A. P.; AND SAHUC, S., 2001. Optic flow is used to control human walking. *Nature neuroscience*, 4, 2 (2001), 213. (cited on pages 12 and 13)

WEISS, S.; ACHTELIK, M. W.; LYNEN, S.; CHLI, M.; AND SIEGWART, R., 2012. Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 957–964. IEEE. (cited on pages 9 and 13)

WEISS, S.; SCARAMUZZA, D.; AND SIEGWART, R., 2011. Monocular-slam–based navigation for autonomous micro helicopters in gps-denied environments. *Journal of Field Robotics*, 28, 6 (2011), 854–874. (cited on pages 3, 9, 11, and 13)

XU, J.; RANFTL, R.; AND KOLTUN, V., 2017. Accurate optical flow via direct cost volume processing. *arXiv preprint arXiv:1704.07325*, (2017). (cited on page 11)

ZHANG, K.; CHEN, J.; CHANG, Y.; AND SHI, Y., 2016a. Ekf-based lqr tracking control of a quadrotor helicopter subject to uncertainties. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, 5426–5431. doi:10.1109/IECON.2016.7794149. (cited on page 8)

ZHANG, X.; YANG, Z.; ZHANG, T.; AND SHEN, Y., 2016b. An improved kalman filter for attitude determination of multi-rotor uavs based on low-cost mems sensors. In *2016 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*, 407–412. doi:10.1109/CGNCC.2016.7828820. (cited on page 8)

ZINGG, S.; SCARAMUZZA, D.; WEISS, S.; AND SIEGWART, R., 2010. Mav navigation through indoor corridors using optical flow. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 3361–3368. IEEE. (cited on pages 3, 10, 11, 12, and 13)