Aerial Vehicle Formation Sensing and Control

Lachlan Deakin
U4847846

Supervised by Professor Robert Mahony

November 2013

A thesis submitted in part fulfilment of the degree of

Bachelor of Engineering
The Department of Engineering
Australian National University

Lachlan Deakin
1 November 2013

Acknowledgements

I would first like to thank my supervisor Professor Robert Mahony. Your advice and support kept me on track and was critical to the success of this project. I am grateful that you gave me the flexibility and resources to take this project as far as I could.

Geoffrey Stacey, thank you for your wide support throughout my project. You really helped me in getting my head around the field of formation control and bond-graphs. I very much appreciate that you offered to proofread my thesis. Best of luck with your project and I hope my research is particularly useful for you.

Moses Bangura, I would not have made it so far in experimentation without your support and helpful discussions regarding the quadrotor, thank you. Thanks to Xiaolei (Eric) Hou for always being available to fly my quadrotor and managing to never crash it. Thank you Evan Slatyer for your technical advice and for looking over some of my earlier designs. Best of luck to all of you with your PhDs! Also a thank you to Alex Martin for your technical advice and assistance with the quadrotor build.

Thanks to Engineers Australia for declaring me the winner of ITEESPAN 2013 for this work. I appreciate the recognition.

Thank you to all the people I have met in engineering over the last 4 years. A notable mention to the best engineering teams ever - Piscean Developments and Crackety Wacketys. The stories of the advanced systems we developed will no doubt be passed through the generations. ANU has been a great learning experience and my passion for engineering has only grown stronger each year.

Thank you Sharon Tran for keeping me sane through this project and for being so awesome.

Finally, I would like to give a special thanks to my mum, Vicki Deakin, for editing my thesis and supporting me over the years.

Abstract

The objective of this study was to develop low-cost, robust and accurate sensor systems on a quadrotor for the implementation of a sensor-based vehicle formation control algorithm. The algorithm was recently developed at the ANU and was only ever tested in simulation.

A new quadrotor was assembled and mounted with a custom omnidirectional vision system to acquire bearings measurements of other vehicles. A custom omnidirectional lens was used which had a better vertical field of view compared with tested commercial solutions. The vision system, coupled with an efficient marker detection algorithm, detected bearings of other vehicles at 55Hz with a typical error of less than 5° in azimuth and elevation.

A widely used driver package for the Vicon motion capture system was modified to provide robust velocity measurements over a poor wireless channel. This was experimentally evidenced by a significant reduction in variability of velocity measurements when compared with a previously used method. An adaptation to the formation control algorithm was proposed which could function without velocity measurements for system damping and its performance was verified in simulation.

The robustness and accuracy of these key sensor measurements was demonstrated in a flight test where the quadrotor was flown under manual control. Support for offboard control was added to the flight controller so that the vehicle could be controlled by the vision system. Unfortunately, the quadrotor was not flown under the control of the formation control algorithm because of time constraints.

Contents

List of Figures

List of Tables

Glossary of Terms

ANU	Australian National University
API	Application Programming Interface
CPU	Central Processing Unit
FOV	Field of View
GPS	Global Positioning System
GPU	Graphical Processing Unit
HDD	Hard Disk Drive
HSV	Hue, Saturation and Value
IMU	Inertial Measurement Unit
IR	Infrared
I2C	Inter-Integrated Circuit
LP-180	Commell LP-180 Pico-ITX single board computer
MDF	Medium-Density Fibreboard
PD	Proportional-Derivative
RAM	Random Access Memory
RC	Radio Control
ROI	Region of Interest
ROS	Robot Operating System
SSD	Solid State Drive
UART	Universal Asynchronous Receiver/Transmitter
Vicon	Vicon Moture Capture System

Chapter 1

Introduction

1.1 Description and Scope

Vehicle formation control encompasses the problem of arranging groups of autonomous vehicles in specific relative geometries. The primary goal of this project is to develop a vehicle and sensor systems for the implementation of a sensor-based vehicle formation control algorithm. The algorithm was developed in the computer vision and robotics research group at The Australian National University (ANU) by Stacey et al. (2013).

Vehicles working in formation provide improved redundancy, robustness, efficiency and flexibility over a single vehicle (Balch and Arkin, 1998). Formation control of vehicles has applications in spacecraft interferometry (Beard et al., 2001), environment mapping, surveillance, sensor distribution, and distributed manipulation of objects (Das et al., 2002; Michael et al., 2011). Early vehicle formation control research was conducted on ground vehicles, but recently research has focused on aerial vehicles. Formation control algorithms on aerial vehicles are often dependent on external sensors to locate other vehicles which are only available in laboratory environments.

The formation control algorithm developed at ANU can operate with only partial state measurements - such as bearing-only or range-only measurements - of other vehicles (Stacey et al., 2013). These kinds of measurements are obtainable with low cost and low weight sensor systems that can be implemented onboard an aerial vehicle. The algorithm is formalised using the bondgraph modelling framework which is well described in (Borutzky, 2006). Bondgraph modelling allows a graphic representation of the flow of energy through a system and can give a detailed perspective on system behaviour and stability. The algorithm applies virtual mechanical couplings between vehicles - such as springs and dampers - which reach a minimum energy state when all vehicles are in formation. The algorithm was previously only demonstrated in simulation and did not address some of the practical issues associated with a physical implementation. A physical implementation of the algorithm will provide insight into the impact of practical issues such as sensory limitations and under-actuation of control forces.

This project aimed to develop robust sensors for the implementation of this algorithm on a vehicle known as a quadrotor. A quadrotor was assembled with a custom omnidirectional vision system to provide bearing measurements of other vehicles. The quadrotor used a cutting-edge open source flight controller which was modified to support the implementation. The algorithm was adapted to improve its practicality for implementation on an aerial vehicle. The system developed for this project was intended to provide a suitable platform for implementation and analysis of the formation control algorithm.

1.2 Thesis Contributions

This primary contributions of this thesis are summarised as:

- A quadrotor was assembled with a custom omnidirectional vision system.
- A custom omnidirectional lens was designed capable of detecting bearings of other vehicles at high accuracy and over a large vertical field of view.
- Support for offboard control was added to the flight controller of the quadrotor.
- A widely-used driver for the Vicon motion capture system was modified to improve the reliability of velocity measurements in situations where measurements were lost due to vehicle occlusion or packet loss.
- A vehicle marker detection algorithm capable of running at high speed onboard the quadrotor was developed and tested.
- An alternative formation control algorithm was proposed in which damping occurred in the image space rather than with velocity measurements.

1.3 Thesis Outline

This thesis is divided into six chapters following this introduction. Chapter 2 presents an overview of the fields of visual servo control and formation control and explains the utility of quadrotors as a research platform. Chapter 3 formulates the formation control algorithm being implemented and develops a modification to the algorithm. Chapter 4 details the hardware developments, such as the development of an omnidirectional vision system, and construction of a new quadrotor. Chapter 5 covers the software developments and gives a detailed analysis on the quality of sensor measurements. Chapter 6 discusses simulations of the formation control algorithm and addresses the future developments required before the algorithm can be applied. Chapter 7 discusses the primary contributions of this project and future research prospects.

Chapter 2

Background

2.1 Formation Control

The control strategies for formation control typically fall under three general classifications - behaviour-based and potential field approaches; the leader-follower approach and the virtual structure method. Research is attempting to address formation control issues such as formation stability, sensory limitations and complex vehicle topologies (Chen and Wang, 2005).

Followers track a leader in coordinated arrangements in the leader-follower approach. Wang (1991) developed simple and robust control laws using a leader-follower approach. His method relied on nearest neighbour tracking and inter-vehicle communication. A downside of early leader-follower approaches was that there was a single point of failure - the leader - and complications if followers fell behind. An experimental implementation of manipulating and transporting objects using quadrotors flying in formation was reported by Michael et al. (2011). Their method was dependant on global state information and could only follow trajectories slowly due to uncertainties in sensing and limited control performance.

Behaviour-based control schemes depend on a set of desired behaviours that are weighted depending on the current circumstances of the vehicle. Balch and Arkin (1998) developed a behavioural-based control scheme to maintain and move between specific geometric formations for a small number of car-like robots. The method required direct perception of other robots and transmission of global coordinates. A robust behavioural-based approach implementing artificial potentials from virtual leaders was proposed by Leonard and Fiorelli (2001). Their approach demonstrated simple group control, although it was for schooling and flocking behaviour rather than formation control.

The virtual structure method treats the entire formation as a rigid body and the structure moves while accurately maintaining a specific arrangement among agents. Beard et al. (2001) introduced a virtual structure approach that could be combined with leader-following or behavioural-control techniques for spacecraft interferometry with further work by Ren and Beard (2004). Their research improved existing virtual structure approaches by implementing a decentralised approach to improve robustness. However, the virtual structure method is subject to poor handling in dynamic environments because of slow response times.

A recent approach is passivity-based control in which a system is described by its real and virtual energy. Virtual energy can include sources such as deviation from a goal state. Hatanaka and Igarashi (2012) developed a passivity-based approach which performed well and could function under the presence of communication delays and objects in the environment. The method could function in varying topologies, such as when an agent loses connectivity with the rest of the formation. Passivity-based approaches benefit from their simplicity and robustness and are often formulated in such a way that stability is easily demonstrated.

Formation control mechanisms often require global position and velocity measurements, although, it can be difficult to obtain these measurements, particularly on an aerial vehicle (Stacey et al., 2013). Onboard sensors typically only provide partial relative state measurements of other vehicles (Franchi et al., 2012). Das et al. (2002) described a scalable framework for formation control based on leader-follower chains utilising decentralised reactive controllers. Controllers were modularised into performing simple tasks, such as object avoidance, and more complex tasks such as formation control. The framework was demonstrated on ground vehicles using only relative state measurements from an omnidirectional camera. The vehicles were able to manipulate objects and navigate through small openings.

A decentralised behaviour-based control scheme for the teleoperation of groups of quadrotors using only relative-bearing measurements from a camera was recently reported (Franchi et al., 2012). A human operator controls the scaling and placement of the vehicles as a bearing-constrained formation is scale, rotational and translational invariant. A leader-follower approach is employed with behaviour-based control schemes in place to avoid objects and inter-vehicle collisions. The use of a human-in-the-loop control scheme allows robots to navigate an environment that cannot be handled with complete autonomy.

A formation control framework realised using bondgraph modelling and port-Hamiltonian theory was proposed by Stacey et al. (2013). Port-Hamiltonian theory is a modelling approach for physical engineering systems with complex interactions (Schaft, 2006). Bondgraph modelling is a powerful graphical modelling methodology for examining the flow of energy through a system (Borutzky, 2006). The algorithm could function in complex vehicle topologies with bearing-only or range-only sensors. The implementation lacked common features seen in formation control such as object avoidance and teleoperation. A proof of stability was provided which was easily formulated from the bondgraph model. Methods from the field of visual servo control were used in adapting sensor measurements into the model.

2.2 Visual Servo Control

The use of vision information in a feedback loop for a mechanical control system is known as visual servo control. This field broadly combines research from computer vision, kinematics and dynamics. Visual servo control has classically addressed the control of robotic manipulators into a specific position based on observed image features (Hutchinson et al., 1996; Shirai and Inoue, 1973). Controlling based on an error from a goal state in the image space is known as image-based visual servo control (IBVS) (Chaumette and Hutchinson, 2006, 2007; Corke and Hutchinson, 2001). Another important field is position-based visual servo control (PBVS) where the pose of objects in Cartesian space are estimated and used in the control algorithms (Wilson et al., 1996).

Chaumette and Hutchinson (2006) described an image Jacobian which relates image feature velocities with the linear and angular velocity of a camera. The image Jacobian is dependent on a measurement of the depth of a feature relative to the camera frame which is often unavailable. The depth of image features can be obtained through stereoscopic reconstruction, although this requires more than one camera. Often the depth is set to a constant value, such as the desired range, which can provide reasonable performance. Several methods exist for depth estimation such as partial pose estimation, adaptive depth estimation and image Jacobian estimation (Hutchinson et al., 1996).

An IBVS control design targeted at under-actuated dynamics systems that could operate with no feature depth measurements was shown in (Hamel and Mahony, 2002). The algorithm was a full dynamic control design suited to high-performance systems where under-actuation must be considered. Mahony and Stramigioli (2012) developed an IBVS control method for a dynamic system presented with the bondgraph formalism. Their algorithm could operate in the absence of translational camera velocity measurements and feature depth measurements.

IBVS controllers often require position or velocity measurements of the camera. In many systems it can be difficult to obtain direct velocity measurements and they must determined from position measurements. Numerically differentiation position is one method of computing velocity from position, however it is sensitive to noise. In early work by Berghuis and Nijmeijer (1993), a proportional-derivative (PD) controller for the regulation of robotic manipulators which utilised only position feedback was introduced. This was a novel approach as previously a controller with a derivative term would be dependent on a velocity measurement. Mahony and Stramigioli (2012) adapted the controller into a bondgraph in order to avoid direct measurement of feature velocities.

A branch of modern vehicle formation control research is attempting to address the difficulty with using formation control algorithms on systems with cheap and limiting sensors. IBVS control schemes are available which can operate in the absence of both feature depth and camera velocity measurements (Mahony and Stramigioli, 2012). These algorithms show applicability to image-based formation control algorithms which often already utilise IBVS based approaches for regulation of feature positions.

2.3 Quadrotors

A quadrotor (also known as a quadcopter or quadrocopter) is an aerial vehicle with four rotors. They benefit over other rotorcraft such as helicopters with their simplicity of mechanical operation. Small scale quadrotors have become a popular platform for robotics research because of their low cost and ability to fly indoors. Quadrotors have a high power requirement which has delayed their widespread use, however, recent improvements in battery technology has increased their popularity.

Purpose-built commercial quadrotor systems are available for applications such as surveillance and hobbyist flying. Custom quadrotors are often assembled from base components for tailored and high performance applications. The key components of a quadrotor are:

Flight controller
Measures the attitude of the vehicle and controls motor actuation for stable flight. Flight controllers can run in autonomous flight modes or connect to a receiver for manual radio control (RC).

Motors
Four electric motors allow propellers to spin which generates thrust.

Electronic speed controllers (ESCs)
ESCs control the speed of the motors as set by the flight controller.

Power system
Powers the motors and onboard electronic systems.

Frame
Secures the major subsystems of the vehicle.

A quadrotor configured in the diamond layout is shown in Figure 2.1. In this figure the roll, pitch and yaw axis of the vehicle are the x, y and z axis respectively and the forward direction of the vehicle is $+x$. Quadrotors have two clockwise rotating rotors on one axis and two counter-clockwise rotors on the other which are

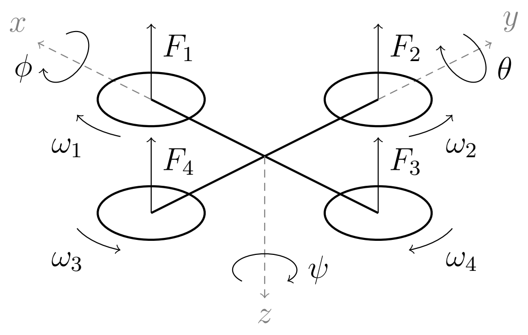


Figure 2.1: Diagram of a quadrotor showing axis of rotation, rotor directions and thrust directions.

all pitched to give upwards thrust. This configuration induces a zero net torque and naturally stabilises the yaw of the quadrotor when the net thrust from each axis is equal. The vehicle yaws by changing the balance of thrust from each axis.

Roll and pitch are independent and are controlled by manipulating the thrust between diametrically opposite rotors. If the imbalance of thrust is equal and opposite, the total vehicle thrust remains constant. The altitude of the quadrotor is controlled by adjusting the net thrust of all of the rotors. A quadrotor will approximately maintain altitude if manoeuvres are confined to small pitch and roll angles.

Minor rotor imbalances and interaction with the environment can lead to instabilities. Quadrotors use stability controllers to deal with the fast vehicle dynamics based on attitude measurements from onboard Inertial Measurement Units (IMUs). IMUs use a variety of complementary sensors to provide an accurate attitude estimate. Accelerometers measure the acceleration of the vehicle which can be used to approximate velocity when assisted by a reference measurement.

Chapter 3

Formation Control Architecture

3.1 Modeling a Quadrotor and Sensors

This chapter develops the formation control architecture described in (Stacey et al., 2013). First, a dynamic model of a vehicle is developed and the concept of bondgraphs is introduced. The formation control algorithm is gradually developed as further bondgraph theory is introduced. This chapter concludes with a discussion of some of the practical issues with the architecture and introduces a modified algorithm for improved practicality.

A vehicle is modelled as point masses moving in $\mathbb{R}^3$. Each vehicle i has an associated mass m_i, position $p_i \in \mathbb{R}^3$ and orientation $R_i \in SO(3)$. The energy of vehicle i can be decomposed into kinetic energy,

$$T_i(\dot{p}_i) := \frac{1}{2m_i} \|m_i \dot{p}_i\| , \tag{3.1}$$

and potential energy $U_i(p_i)$,

$$U_i(p_i) := -m_i g p_i^z. \tag{3.2}$$

The vehicle is driven by a control force $F_i \in \mathbb{R}^3$ which is assumed to be fully-actuated by the vehicle. The dynamics of the system are defined by

$$m_i \ddot{p}_i = \frac{\delta U_i(p_i)}{\delta p_i} + F_i. \tag{3.3}$$

The model is defined in the inertial reference frame for simplicity of analysis.

Stacey et al. (2013) represented a group of vehicles as a network graph which described the topology of the vehicles. A topology describes the sensor relationships between vehicles. Let $\mathcal{E}_i$ represent the immediate neighbours of vehicle i with a relative position available. Let the relative position of vehicle i with respect to vehicle j be $q_k = p_i - p_j$ with $k \in \mathcal{E}_i$. The relative velocity is $\dot{q}_k = \dot{p}_i - \dot{p}_j$.

The space in which a measurement exists is termed the sensor space. We define a measurement Jacobian, N_k, which relates the sensor space measurement to the relative velocity between vehicles.

$$\dot{y}_k := N_k \dot{q}_k. \tag{3.4}$$

The range between two vehicles is

$$r_k := \sqrt{q_k^\top q_k}. \tag{3.5}$$

A bearing is defined as a unit vector in Cartesian space according to

$$\mathbb{S}^2 = (x, y, z)^\top : x, y, z \in \mathbb{R}, x^2 + y^2 + z^2 = 1. \tag{3.6}$$

The bearing is

$$s_k := \frac{q_k}{r_k} \in \mathbb{S}^2, \tag{3.7}$$

and the time derivative of the bearing is

$$\dot{s}_k = \frac{\dot{q}_k}{r_k} - \frac{\dot{r}_k}{r_k} s_k = \frac{1}{r_k}\left(I_3 - s_k s_k^\top\right)\dot{q}_k. \tag{3.8}$$

This shows that the associated measurement Jacobian (or Image Jacobian) relating a bearing measurement to $\mathbb{R}^3$ is

$$L_k = \frac{1}{r_k}\left(I_3 - s_k s_k^\top\right). \tag{3.9}$$

The goal formation can be specified by desired relative bearings in the inertial reference frame. The formation is invariant to scale and position if defined this way, although rotation is constrained by the goal bearings. Let a desired bearing be s_k^*. The error between the measured bearing and the desired bearing is then

$$\tilde{s}_k := s_k - s_k^*. \tag{3.10}$$

It is assumed that there exists at least one formation for which all $\tilde{s}_k = 0$. The objective of this formation control algorithm is to converge the vehicles into such a minimum error state.

3.2 Bondgraph Formalism

Bondgraph modelling is a graphical method of representing energy flows in systems. Bondgraphs were first devised by Professor H. Paynter at the Massachusetts Institute of Technology in 1959 (Borutzky, 2006). This section provides a brief introduction to bondgraphs using formalisms from Borutzky (2006) and introduces concepts relevant to the formation control algorithm (Stacey et al., 2013).

A bondgraph splits up a system by its subsystems and components into vertices, or nodes. Edges, or bonds, are formed between nodes and represent energy flows between them. Each bond has associated conjugate variables called effort, e, and flow, f. In a translational mechanics system, the effort is associated with a force and the flow with a velocity. Bondgraph modelling can be applied in many energy domains. Bondgraphs must obey the first principle of energy conservation. The instantaneous power transferred between components is the product of the effort and the flow.

As power can flow in either direction between nodes, a half arrow is used to indicate the direction of positive flow. Typically, the effort is displayed above for a horizontal bond, and to the left for a vertical bond. Convention is to place the flow on the same side as the half arrow.

The component in which the effort is used in the computation of the flow is indicated in a bondgraph by a perpendicular stroke at the end of an arrow known as the causal stroke. The model on the end of the arrow without the causal stroke is where the effort is computed and the flow must be known.

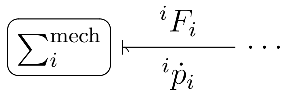


Figure 3.1: Bondgraph model of the vehicle mechanical system node.

Let $\boxed{\sum_i^{\text{mech}}}$ be the mechanical system of the quadrotor. This system is shown as a node and bond in Figure 3.1. The mechanical system is driven by a desired control force (effort) and has an associated velocity (flow). Note that the force, and the velocity is expressed in the body fixed frame as indicated by the left superscript. Control inputs are typically defined in the body fixed frame for a robot.

Consider a simplistic controller for a quadrotor implementing gravity compensation and velocity damping. The force and velocity variables must be transformed between the body fixed frame and the inertial reference frame to implement these controllers. The vehicle orientation is represented in the rotation matrix R_i. A TF (transform) element is used to represent a reversible transformation of energy in a bondgraph. Specifically, an MTF (modulated transform) is used if the transformation is a function of time. An MTF is used to apply the rotation matrix, R_i, to transform the force and velocity variables between the body fixed frame and the inertial reference frame.

A virtual damping force can be implemented as

$$\delta_i = D_i \dot{p}_i, \tag{3.11}$$

where D_i is a positive definite constant matrix. This can be implemented in a bondgraph with a dissipative element. The direction of positive flow is chosen such that δ_i applies a force which opposes the direction of motion of the vehicle. The virtual damping force is required for the system to converge, otherwise the system will oscillate indefinitely due to no loss in energy. It is not straightforward to obtain a robust velocity measurement on an aerial vehicle and this issue will be addressed later in this chapter.

Gravity can be compensated with a storage element, C_i^{comp}, in the bondgraph. The energy in the storage element can be described by a Hamiltonian, $H_i^{\text{comp}}(p_i) := G_i - U_i(p_i)$. G_i is a positive constant which is chosen such that $H_i^{\text{comp}} > 0$ in the region of operation of the vehicle. The energy will be constrained to be non-negative, although this system would not break down if it were to go negative. The gravity compensation element will stop providing energy to the system if the vehicle flies too high as a result of this constraint. The effort from the gravity compensation element is $\partial H_i^{\text{comp}}(p_i)/\partial p_i = m_i g^z$. The sign of this term is chosen according to the orientation of the inertial frame.

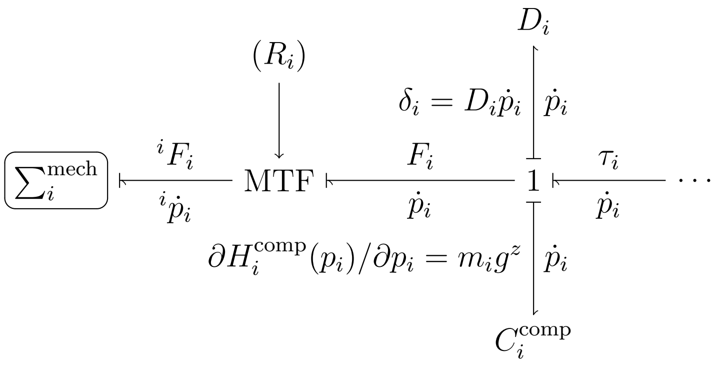


Figure 3.2: Bondgraph model of individual vehicle with gravity compensation and velocity damping.

There is a force term, τ_i, which will be the result of interactions with the other vehicles. The forces from all of these components will sum to give a control force,

$$F_i = \tau_i - D_i \dot{p}_i - m_i g^z. \tag{3.12}$$

The sum of forces can be modelled with a 1-junction in which the flows associated with each connected bond are all equal, and the efforts must sum to zero. The direction of positive flow must be considered when summing to zero. Another type of junction is a 0-junction in which the efforts are equal, and the flows sum to zero. The vehicle system incorporating the coordinate transformation, gravity compensation and velocity damping is shown in Figure 3.2. This entire system will be represented as $\boxed{\sum_i^{\text{vehicle}}}$ in future bondgraphs.

The next step is to develop the virtual mechanical couplings between vehicles which will drive the vehicles into formation. The goal is to minimise the error in bearing measurement (see Equation 3.10). Firstly, the relative velocity between vehicles is converted into a bearing using an MTF applying the image Jacobian

$$\begin{tikzcd}
 & & C_k & & \\
 & (L_k) \arrow[r] & \mathrm{MTF} \arrow[u, "e_k = \partial H_k/\partial \tilde{s}_k = c_k \tilde{s}_k"', "\dot{\tilde{s}}_k = L_k \dot{q}_k"] & & \\
\boxed{\sum_i^{\text{vehicle}}} \arrow[r, maps to, "\tau_i", "\dot{p}_i"'] & 1 \arrow[r, maps to, "\epsilon_k", "\dot{p}_i"'] \arrow[d, "\epsilon_h"', "\dot{p}_i"] & 0 \arrow[u, "\epsilon_k = L_k^\top e_k"', "\dot{q}_k = \dot{p}_i - \dot{p}_j"] \arrow[r, "\epsilon_k", "\dot{p}_j"'] & 1 \arrow[r, "\tau_j", "\dot{p}_j"'] \arrow[d, "\epsilon_h"', "\dot{p}_j"] & \boxed{\sum_j^{\text{vehicle}}} \\
 & \text{links } h \in \mathcal{E}_i & & \text{links } l \in \mathcal{E}_j &
\end{tikzcd}$$

Figure 3.3: Bondgraph model of simple formation control algorithm.

from Equation 3.9. By noting that $\dot{s}_k = \dot{\tilde{s}}_k$ if s_k^* is constant, $\dot{\tilde{s}}_k$ can be input into a storage element C_k. The associated Hamiltonian is then $H_k(\tilde{s}_k) := \frac{1}{2}\tilde{s}_k^\top c_k \tilde{s}_k$, where c_k is a positive definite matrix. This matrix acts as a gain on the virtual energy associated with the error in bearing and will need to be adjusted in balance with other system gains for suitable trajectories. The effort associated with this Hamiltonian is $e_k := \partial H_k/\partial \tilde{s}_k = c_k \tilde{s}_k$ which acts as a virtual force in the sensor space.

It is convenient to think of the storage element acting as virtual spring. The virtual force acts to direct the storage element to a minimum energy state which corresponds to the desired bearing constrained formation. The virtual force is then converted into a control force as

$$\epsilon_k := L_k^\top e_k = \frac{1}{r_k}\left(I_3 - s_k s_k^\top\right) e_k. \tag{3.13}$$

The damping in the vehicle model (see Equation 3.11) drains energy from the system to prevent oscillation around the minimum energy state. Conditions for convergence are addressed in a theorem from Stacey and Mahony (2013).

The bondgraph for this system is shown in Figure 3.3. This is a fundamentally simple and robust bearing-constrained formation control algorithm. Unfortunately, the image Jacobian (see Equation 3.9) depends on a range measurement which is not obtainable with a bearing-only sensor. This issue is addressed in the next section.

3.3 Adaptive Control

Stacey and Mahony (2013) appended a range observer to the formation control algorithm developed in the previous section. The bongraph model of this algorithm can be seen in Figure 3.4. The derivation of this algorithm follows.

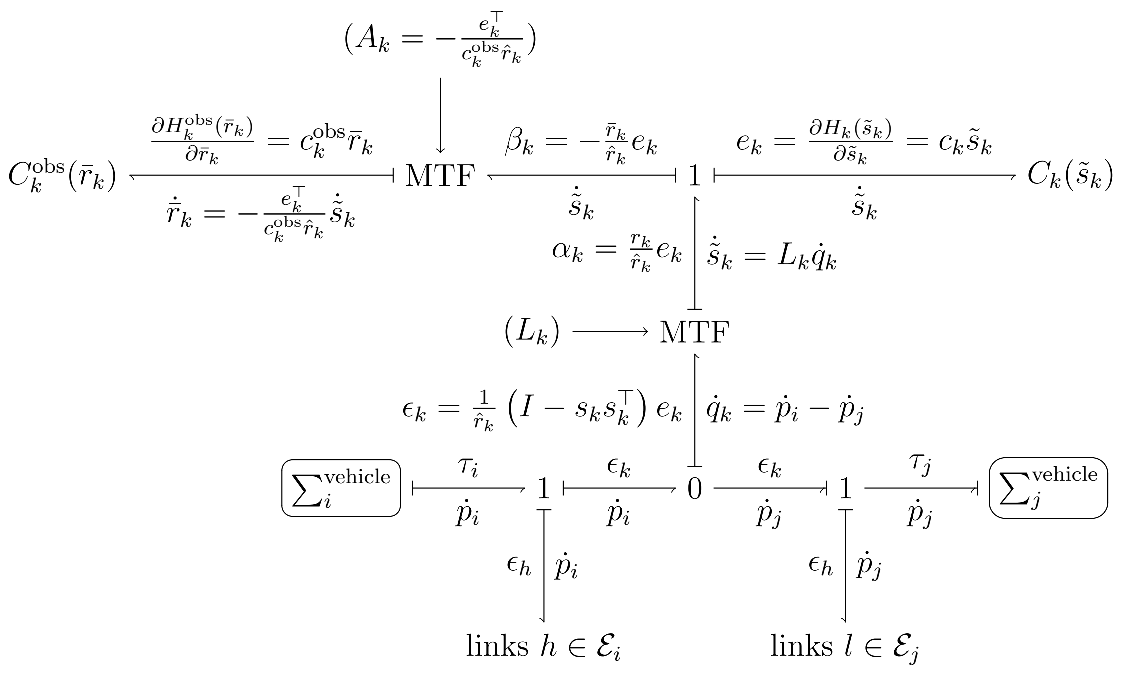

$(A_k = -\frac{e_k^\top}{c_k^{\text{obs}}\hat{r}_k})$
$C_k^{\text{obs}}(\bar{r}_k)$
$\frac{\partial H_k^{\text{obs}}(\bar{r}_k)}{\partial \bar{r}_k} = c_k^{\text{obs}}\bar{r}_k$
$\dot{\bar{r}}_k = -\frac{e_k^\top}{c_k^{\text{obs}}\hat{r}_k}\dot{\tilde{s}}_k$
MTF
$\beta_k = -\frac{\bar{r}_k}{\hat{r}_k}e_k$
$\dot{\tilde{s}}_k$
1
$e_k = \frac{\partial H_k(\tilde{s}_k)}{\partial \tilde{s}_k} = c_k\tilde{s}_k$
$\dot{\tilde{s}}_k$
$C_k(\tilde{s}_k)$
$\alpha_k = \frac{r_k}{\hat{r}_k}e_k$
$\dot{\tilde{s}}_k = L_k\dot{q}_k$
(L_k)
MTF
$\epsilon_k = \frac{1}{\hat{r}_k}\left(I - s_k s_k^\top\right)e_k$
$\dot{q}_k = \dot{p}_i - \dot{p}_j$
$\sum_i^{\text{vehicle}}$
τ_i
$\dot{p}_i$
1
ϵ_k
$\dot{p}_i$
0
ϵ_k
$\dot{p}_j$
1
τ_j
$\dot{p}_j$
$\sum_j^{\text{vehicle}}$
ϵ_h
$\dot{p}_i$
ϵ_h
$\dot{p}_j$
links $h \in \mathcal{E}_i$
links $l \in \mathcal{E}_j$

Figure 3.4: Bondgraph model of vehicle interactions with range estimation.

Let the range error be

$$\bar{r}_k := \hat{r}_k - r_k, \tag{3.14}$$

where $\hat{r}_k$ is the estimated range and r_k is the true range. The force ϵ_k is desired to be implemented with the range estimate rather than the true range. The bondgraph must be modified to maintain passivity of the system after making this change. A second storage element is appended to the system, C_k^{obs}, which compensates for the range error. The Hamiltonian associated with this storage element is $H_k^{obs}(\bar{r}_k) := \frac{1}{2}c_k^{\text{obs}}\bar{r}_k^2$, where c_k^{obs} is a scalar constant. This will have an associated effort of

$$\partial H_k^{\text{obs}}(\bar{r}_k)/\partial \bar{r}_k = c_k^{\text{obs}}\bar{r}_k. \tag{3.15}$$

A 1-junction is used to connect C_k^{obs} with C_k and the rest of the system as shown in Figure 3.4. The effort between the 1-junction and the image Jacobian MTF is denoted as α_k and the effort to the range error storage element is β_k. The constraint imposed in the efforts from this 1-junction is $e_k = \alpha_k - \beta_k$. A suitable choice for the efforts to satisfy this relation is

$$\alpha_k = \frac{r_k}{\hat{r}_k}e_k \tag{3.16}$$

$$\beta_k = -\frac{\bar{r}_k}{\hat{r}_k}e_k, \tag{3.17}$$

which is valid as $\hat{r}_k = \bar{r}_k + r_k$ from Equation 3.14. Rearranging Equation 3.15 for $\bar{r}_k$ and substituting into Equation 3.17, we can see the relation

$$\beta_k = -\frac{e_k}{c_k^{obs}\hat{r}_k}\frac{\partial H_k^{obs}(\bar{r}_k)}{\partial \bar{r}_k} \tag{3.18}$$

The effort β_k must be transformed prior to connecting with the observer's storage element. An MTF is used with transfer function $A_k := -\frac{e_k^\top}{c_k^{obs}\hat{r}_k}$. The flow associated with the observer is

$$\dot{\bar{r}}_k = A\dot{\tilde{s}}_k = -\frac{e_k^\top}{c_k^{obs}\hat{r}_k}\dot{\tilde{s}}_k. \tag{3.19}$$

From Equation 3.14, we have the additional relation

$$\dot{\bar{r}}_k = \dot{\hat{r}}_k - \dot{r}_k. \tag{3.20}$$

It follows that the update equation of the range observer is

$$\dot{\hat{r}}_k = \dot{r}_k - \frac{c_k\tilde{s}_k^\top}{c_k^{obs}\hat{r}_k}\dot{\tilde{s}}_k. \tag{3.21}$$

The range observer consists of a feedforward term $\dot{r}_k$ and an innovation term $-\frac{c_k\tilde{s}_k^\top}{c_k^{obs}\hat{r}_k}\dot{\tilde{s}}_k$. This method is similar to what was employed for the range observer proposed by Mahony and Stramigioli (2012) where a theorem was developed $\hat{r}$ will always be greater than zero. This is an important requirement as control forces become large as $\hat{r}$ approaches zero and a negative range estimate would lead to unpredictable behaviour.

To summarise, a formation control algorithm has been developed which uses only bearing measurements of other vehicles. An adaptive controller is applied to accommodate for the energy discrepancy from not having any range measurements. The key equations of the vehicle model and the algorithm from Figure 3.4 are summarised in Figure 3.5.

Simulation of the system in Figure 3.4 indicated that the range observer did not perform as expected. It was noted in Stacey and Mahony (2013) that the range error may not necessarily approach zero as the bearing error approached zero. The reason is the observer is partitioned off from the system as e_k approaches zero. The range is still tracked due to the feedforward term in the range observer. Thus the range observer is better viewed as an adaptive controller which adapts to the energy discrepancy from the lack of range measurement.

In an implementation of this algorithm, vehicles could determine $\dot{r}_k$ by communicating their velocities in the inertial reference frame. As previously discussed, it is difficult to compute velocity on an aerial vehicle which is a major drawback to this controller. The range observer requires optic flow $\dot{\tilde{s}}_k$ measurements which are

The vehicle dynamics are

$$m_i \ddot{p}_i = \frac{\delta U_i(p_i)}{\delta p_i} + F_i. \tag{3.3}$$

The control force is

$$F_i = \sum_{k \in \mathcal{E}_i} \epsilon_k - D_i \dot{p}_i - m_i g^z. \tag{3.12}$$

The virtual force from vehicle link k is

$$\epsilon_k = \frac{1}{\hat{r}_k} \left(I_3 - s_k s_k^\top \right) c_k \tilde{s}_k, \tag{3.13}$$

where $\tilde{s}_k$ is the error from the goal bearing,

$$\tilde{s}_k := s_k - s_k^*. \tag{3.10}$$

The update equation of the range estimate is

$$\dot{\hat{r}}_k = \dot{r}_k - \frac{c_k \tilde{s}_k^\top}{c_k^{obs} \hat{r}_k} \dot{\tilde{s}}_k. \tag{3.21}$$

Figure 3.5: Summary of formation control algorithm.

sensitive to noise such as from misdetections of markers and image quantisation. Fortunately, the control force is not directly dependent on the optic flow and the observer acts to low-pass filter the measurement noise in the optic flow.

This covers the extent of the research performed prior to the commencement of this project. The next section addresses the dependency on velocity measurements for damping the system. An alternative approach is proposed where damping occurs in the image space.

3.4 Avoiding Velocity Measurements

In the previous system, the virtual damping force is dependant on a measurement of the velocity of the vehicle in the inertial reference frame. As previously discussed, it is difficult to obtain accurate velocity measurements of an aerial vehicle without external references or costly sensor systems. A damping force is critical to the system as it is the only source of energy dissipation and leads to asymptotic convergence of the system. A suitable alternative is to damp the system based on the velocity of features in the image space.

Mahony and Stramigioli (2012) developed a bondgraph implementation of the controller found in (Berghuis and Nijmeijer, 1993). A virtual variable is introduced which is defined such that its derivative reconstructs the true velocity signal. This virtual variable is then used as the velocity term to damp the system. This is safer than using a simple dissipative element which would be dependent on noisy feature velocity measurements.

The bondgraph of this controller from (Mahony and Stramigioli, 2012) with some notational changes is shown in Figure 3.6. The virtual variable in this figure is $\sigma \in \mathbb{R}^3$. The virtual variable is not a unit vector like the bearing measurement due to its complex dynamics. A new node type is introduced called a conductance where the flow is computed from the effort according to

$$G \times \text{flow} = \text{effort}. \tag{3.22}$$

The conductance connects to a storage element, C_{im}, which can implement the desired regulation of the virtual variable. The dynamics of the virtual variable are

$$\dot{\sigma} = \frac{1}{G}\left(k(s-\sigma) - \frac{\partial H}{\partial \sigma}\right). \tag{3.23}$$

The remaining storage element drives the system to minimise the error between the measured bearing s and the virtual variable σ by applying an image effort $s - \sigma$. The conductance acts to dissipate energy from the system and essentially dampens the image effort supplied by C_{im}.

$$\begin{tikzcd}[column sep=huge]
G & 1 \arrow[l, "s-\sigma-\frac{\partial H}{\partial \sigma}"', "\dot{\sigma}=\frac{1}{G}\left(s-\sigma-\frac{\partial H}{\partial \sigma}\right)"] \arrow[r, "\frac{\partial H}{\partial \sigma}", "\dot{\sigma}"'] & C_{im} \\
& 0 \arrow[u, "s-\sigma", "\dot{\sigma}"'] \arrow[r, "s-\sigma", "\dot{s}-\dot{\sigma}"'] & C :: \frac{1}{2}\|s-\sigma\|^2 \\
& \vdots \arrow[u, "s-\sigma", "\dot{s}"']
\end{tikzcd}$$

Figure 3.6: Feature velocity observer bondgraph (Mahony and Stramigioli, 2012).

All of the effort in this velocity controller must be computable in order for it to be physically implementable. The bondgraph has bonds which are dependent on the true range, which cannot be measured. The only sensible position to implement this observer was before the storage element which implements e_k seen in Figure 3.4.

A new variable σ_k is defined as the virtual variable for the controller as in the derivation. Define

$$\tilde{\sigma}_k = \sigma_k - s_k^*, \tag{3.24}$$

where $\tilde{\sigma}_k$ is the error between the virtual variable and the goal bearing. The C_{im} storage element from Figure 3.6 when appended to the formation control algorithm has an effort $\frac{d_k}{2}\left\|\sigma_k - s_k^*\right\|^2$ which drives the virtual variable to the desired bearing.

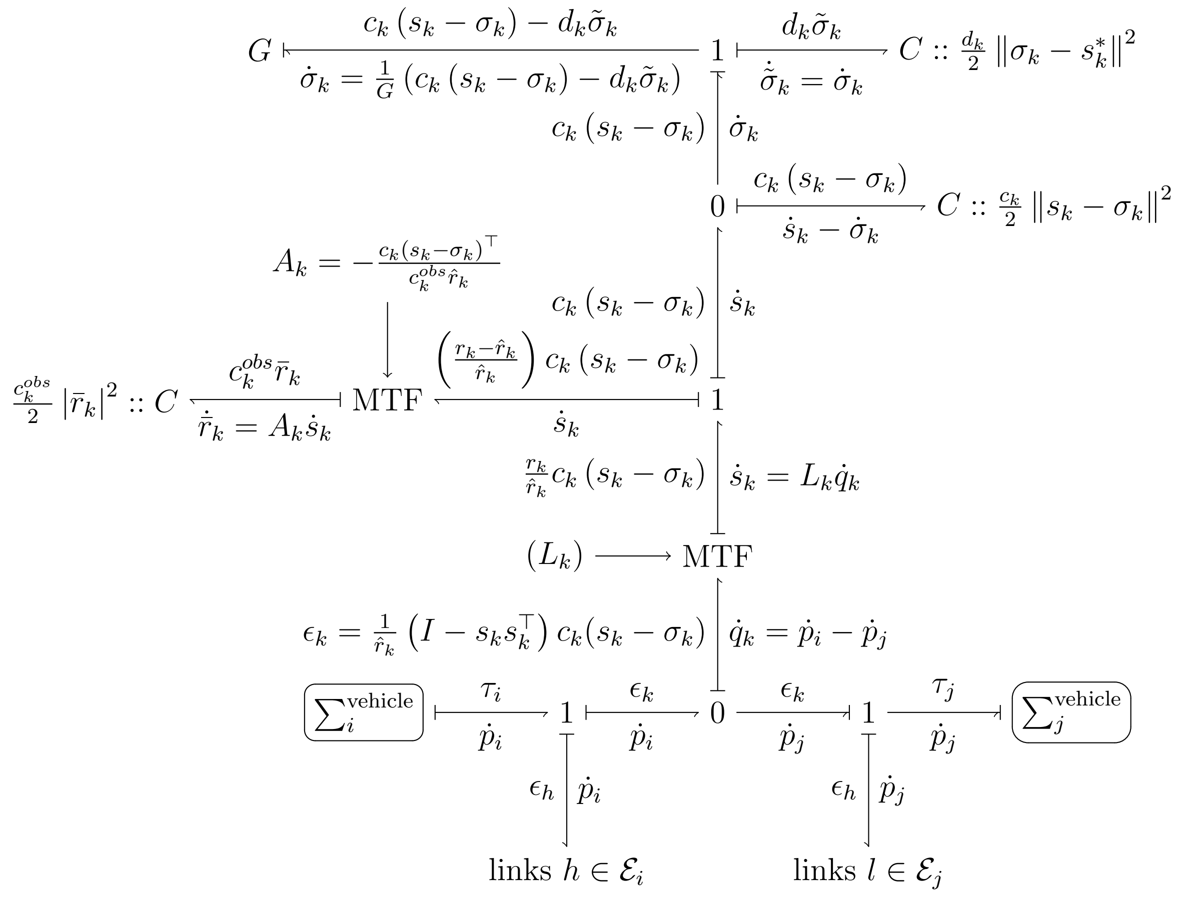

G
$c_k(s_k-\sigma_k)-d_k\tilde{\sigma}_k$
$\dot{\sigma}_k=\frac{1}{G}(c_k(s_k-\sigma_k)-d_k\tilde{\sigma}_k)$
1
$d_k\tilde{\sigma}_k$
$\dot{\tilde{\sigma}}_k=\dot{\sigma}_k$
$C :: \frac{d_k}{2}\|\sigma_k-s_k^*\|^2$
$c_k(s_k-\sigma_k)$
$\dot{\sigma}_k$
0
$c_k(s_k-\sigma_k)$
$\dot{s}_k-\dot{\sigma}_k$
$C :: \frac{c_k}{2}\|s_k-\sigma_k\|^2$
$A_k=-\frac{c_k(s_k-\sigma_k)^\top}{c_k^{obs}\hat{r}_k}$
$c_k(s_k-\sigma_k)$
$\dot{s}_k$
$\frac{c_k^{obs}}{2}|\bar{r}_k|^2 :: C$
$c_k^{obs}\bar{r}_k$
$\dot{\bar{r}}_k=A_k\dot{s}_k$
MTF
$\left(\frac{r_k-\hat{r}_k}{\hat{r}_k}\right)c_k(s_k-\sigma_k)$
$\dot{s}_k$
1
$\frac{r_k}{\hat{r}_k}c_k(s_k-\sigma_k)$
$\dot{s}_k=L_k\dot{q}_k$
(L_k)
MTF
$\epsilon_k=\frac{1}{\hat{r}_k}(I-s_ks_k^\top)c_k(s_k-\sigma_k)$
$\dot{q}_k=\dot{p}_i-\dot{p}_j$
$\sum_i^{\text{vehicle}}$
τ_i
$\dot{p}_i$
1
ϵ_k
$\dot{p}_i$
0
ϵ_k
$\dot{p}_j$
1
τ_j
$\dot{p}_j$
$\sum_j^{\text{vehicle}}$
ϵ_h
$\dot{p}_i$
ϵ_h
$\dot{p}_j$
links $h\in\mathcal{E}_i$
links $l\in\mathcal{E}_j$

Figure 3.7: Bondgraph model of vehicle interactions with image space damping.

The flow coming out of the storage element between $\dot{s}_k$ and $\dot{\sigma}_k$ is $c_k(s_k-\sigma_k)$ which replaces the image effort, e_k, applied in the original velocity damped system. This applies an image effort to drive the measured bearing to the goal bearing which is represented within the virtual variable. The equation of the virtual variable observer is

$$\dot{\sigma}_k=\frac{1}{G}\left(c_k(s_k-\sigma_k)-d_k\tilde{\sigma}_k\right). \tag{3.25}$$

After this system is appended, the virtual force from the vehicle link and the range observer are the same aside from a substitution of e_k with $c_k(s_k-\sigma_k)$ (see Figure 3.8).

The control force is no longer simply proportional to the error from the goal bearing and its operation is not as clear as in the simpler system. A demonstration that this bondgraph does in fact converge to a minimum error state and an analysis of its performance is shown in Chapter 6. This modification has removed the dependence on velocity measurements to damp the system. Communication of velocity measurements remains a suitable method for computing $\dot{r}$ measurements between vehicles.

The virtual force from vehicle link k is

$$\epsilon_k = \frac{1}{\hat{r}_k}\left(I - s_k s_k^\top\right) c_k(s_k - \sigma_k). \tag{3.26}$$

The virtual variable has update equation

$$\dot{\sigma}_k = \frac{1}{G}\left(c_k\left(s_k - \sigma_k\right) - d_k\tilde{\sigma}_k\right), \tag{3.25}$$

where the error between the virtual variable and the goal bearing is

$$\tilde{\sigma}_k = \sigma_k - s_k^*. \tag{3.24}$$

The update equation of the range estimate is

$$\dot{\hat{r}}_k = \dot{r}_k - \frac{c_k\left(s_k - \sigma_k\right)^\top}{c_k^{obs}\hat{r}_k}\dot{s}_k. \tag{3.27}$$

Figure 3.8: Summary of formation control algorithm with image space damping.

A quirk with this implementation and the previous implementation is that control forces are inversely proportional to the range between vehicles. This means if a formation is very large, control forces can be relatively negligible. Effective trajectories and performance are still achievable with appropriate algorithm gains as shown in Chapter 6.

In summary, the original formation control algorithm has been adapted to remove its dependence on velocity measurements for system damping. The full bondgraph model of this new controller is shown in Figure 3.7 with a summary of equations presented in Figure 3.8. The algorithm is implementable with only measurements of the the relative bearings and range velocities of other vehicles. The next chapter develops the hardware systems required to implement the algorithms developed in this chapter on a quadrotor.

Chapter 4

Hardware Development

4.1 Hardware Systems Overview

This chapter covers the hardware developments required to implement the formation control algorithm on a quadrotor. The quadrotor hardware and assembly is discussed as well as the interaction with an external position sensor called Vicon. The primary achivement in the hardware development is the design of a low-cost omnidirectional vision system to detect the bearings of other vehicles.

4.2 Building a Quadrotor

A high end research oriented quadrotor was recently developed at the ANU. The quadrotor is powered by the PX4FMU, a high end open source flight management unit (PIXHAWK, 2013). It was developed by the PIXHAWK team from ETH Zurich which is a research group focused on aerial vehicle localisation using computer vision (PIXHAWK, 2013). It runs a Unix-like real-time operating system (NuttX) and includes many useful features beyond motor control such as attitude and position estimators and support for autonomous control.

The PX4FMU based quadrotor used AutoQuad ESC32 electronic speed controllers for motor control (AutoQuad, 2013). The ESC32 firmware is open source can be adjusted to obtain a variety of motor measurements typically not available from an ESC, such as current and power draw. This is well suited to future and current research at the ANU for advanced motor control algorithms that can utilise these measurements. Custom firmware was developed for the AutoQuad ESC32s to interface over an (Inter-Integrated Circuit) I2C bus by other researchers at ANU in parallel with this project.

This quadrotor design was well suited to this project, although, the quadrotor itself was not available as it was being used in other research. A new quadrotor was assembled using the existing quadrotor as a reference design. The quadrotor was particularly well suited due to its highly adaptable flight controller firmware and capability of supporting a relatively large payload.

Many quadrotor components are partially manufactured so that they can assembled into a wide variety of different systems. The PX4FMU and the ESC32 required cables and connectors to be soldered on for power and data connections. Due to a supply issue, a custom center plate was manufactured based off the existing design. An I2C distribution board was built to connect all of the ESC32s to the PX4FMU which was modified over the original design for modularity. A new battery mount was designed and laser cut which could hold two batteries which is further discussed in Section 4.8. The arrangement of the power board and the PX4FMU was adjusted to reduce their height above the quadrotor frame.

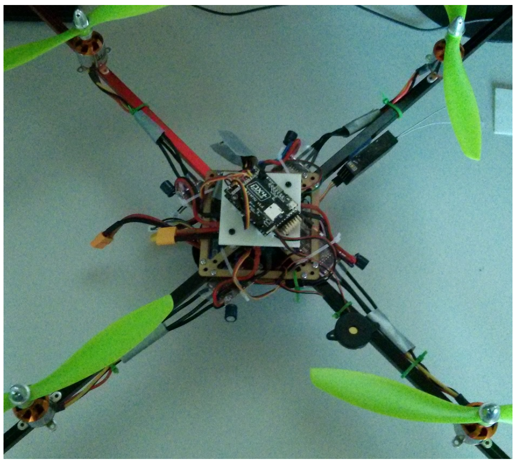


Figure 4.1: Photo of assembled quadrotor.

The accelerometers of the PX4FMU are calibrated by setting it level on the positive and negative normal to each axis. Previously the PX4FMU was secured in a makeshift shoebox design for this process which did not give very good calibration accuracy. A box was designed and laser cut to better secure the flight controller and ensure accurate axis alignment. The calibration was suitably accurate after only one attempt.

Figure 4.1 shows the assembled quadrotor. In this state, the quadrotor capable of supporting current and future research in autonomous control and high performance control. The following sections outline the developments to append an omnidirectional vision system to this quadrotor.

4.3 Vision System

The purpose of the vision system was to measure the bearings of other vehicles in a panoramic view around the quadrotor. A major requirement for this system was that it be low-cost so that future implementation on multiple vehicles would be feasible. It was critical that the vision system be low weight to minimise the impact on the vehicle dynamics. The system needed to be robust to light impacts and cheap to repair in the event of a crash. The vision system could also not impact on the already low flight time of the quadrotors and had to be housed in an easily detachable module.

The vision system was broken down into several fundamental subsystems:

Camera
Captures vision data.

Omnidirectional lens
Expands the field of view of the camera to view 360° around the vehicle.

Processing board
Processes vision data and also implements formation control algorithm.

Visual identification markers
Used to uniquely identify quadrotors with the vision system.

Power supply
Powers the processing board and the camera.

Mounting
Safely houses the vision system and can be easily detached from the quadrotor.

The following sections outline the hardware development process for each subsystem.

4.3.1 Processing Board

The processing board function is to process data from the camera and other sensors and compute control forces from the formation control algorithm. Several critical system requirements had to be met to ensure the vision system was practical for implementing the algorithm. A key requirement was support for a software framework called Robot Operating System (ROS) to align with existing systems and simplify development. Additionally, support for the Open Source Computer Vision Library (OpenCV) was highly desirable for the system.

The metrics for the processing board, listed in descending order of importance were:

1. Architecture/Operating System - A x86 or x64 processor architecture and Linux support was essential for use with the desired tools.
2. Interfaces - Suitable interfaces for communicating with a camera and the PX4FMU must be available.

3. Size/Weight - Small size and low weight were desirable to minimise the impact on the dynamic performance of the quadrocopter.
4. Power - Low voltage is beneficial to simplify regulation and reduce the size of the power supply. Low power draw improves the flight time.
5. Processing Power - A high performance Central Processing Unit (CPU) supports the use of more advanced computer vision algorithms at high frame rates. A Graphical Processing Unit (GPU) could be beneficial over a CPU for some computer vision tasks.

Boards that did not meet the processor architecture and operating system requirements were discarded due to potential toolchain issues and lack of support for ROS. A GPU is a non-essential component, but could provide performance enhancements. A GPU needed to support CUDA or OpenCL otherwise it could not be used with OpenCV. Other key metrics such as processing power were used in sifting through boards. Several processing boards were shortlisted for possible selection as shown in Table 4.1. The shortlisted boards were the Commell LP-180 and LP-170 (COMMELL, 2013), SECOnITX-ION (Seltech International, 2013) and the VIA EPIA P910 (VIA Embedded, 2013). All of these boards had an x86 processer architecture and supported Linux. Some weight metrics were not obtainable and were estimated based on their size.

Table 4.1: Comparison of processing boards

PROCESSING BOARD	**COMMELL LP-180***	**COMMELL LP-170**	**SECONITX-ION**	**VIA EPIA P910**
Interfaces	USB 2.0, GB Lan, serial	USB 2.0, GB Lan, serial	USB 2.0, GB Lan, GPIO, serial	USB 3.0, GB Lan, GPIO
Size	100x72mm	100x72mm	120x120mm	100x72mm
Weight	150g	150g	250g	160g
Power	5V	12V	12V	12V
CPU	2x1.65GHz	2x1.8GHz	2x1.9GHz	4x1GHz
GPU	AMD	Intel	NVIDIA	VIA GPU
Availability	Ordered	In lab	Poor	Good

* Chosen processing board.

A notable range of boards that were not listed were those made by Gumstix (Gumstix, 2013). Gumstrix boards use an ARM architecture and are extremely small, low weight and have very low power consumption. They have been demonstated running unofficial ARM builds of Ubuntu and have managed to run ROS and OpenCV. Despite these apparent advantages, they were not used due to the ARM architecture had many toolchain issues and their performance was deemed not quite satisfactory for the required applications.

Shortlisted boards had mostly comparable metric performance. The only GPUs that were likely supported by OpenCV were the AMD and NVIDIA GPUs in the LP-180 and SECOnITX-ION. Although the GPU on the SECOnITX-ION would have likey been the easiest to develop on, it was discarded due to tis larger size and weight.

The remaining boards were the Commell LP-180, LP-170 and the VIA EPIA-P910. The LP-180 had an AMD GPU that supported OpenCL for OpenCV, whereas the EPIA-P910 had a poorly supported proprietary GPU. The low voltage requirerement for the LP-180 allowed for a smaller power supply and simpler voltage regulation. It was primarily for this reason that the LP-180 was chosen for use. The LP-180 was also the cheapest of all the boards at approximately $225, comparatively cheap compared to the EPIA-P910 at approximately $360 and the SECOnITX-ION at up to $800.

The LP-180 required a data storage device and a Random Access Memory (RAM) stick to function. A Hard Disk Drive (HDD) would not be suitable due to high weight and potential for damage from fast movements and impacts. A Solid State Drive (SSD) is more robust, weights less than a HDD, and was ideal for use. Also, SSDs have very fast read and write times and are well suited to real-time video logging. 60GB was suitable size for the expected applications of the vision system. Appropriate RAM was identified with high clock speed prioritised over latency. Additional components purchased were USB cables and a WiFi adapter. The final cost of all components was approximately $350.

ROS only supports Unix-like systems, and only officially supports Ubuntu Linux based distributions. For this reason, an Ubuntu based distribution was installed on the LP-180 to ensure robustness and minimise potential development issues. Several distributions were investigated and a lighweight variant of Ubuntu was selected called Xubuntu (Xubuntu, 2013). A 32-bit long term support release (Xubuntu 12.04.2) was used. The ROS Groovy Galapagos distribution was installed (targeted at Ubuntu 12.04), which was the latest version at the time of installation (Willow Garage, 2013).

4.3.2 Camera

The camera acquires vision information and sends it to the processing board. Metrics were identified taking into consideration the high level system requirements of the vision system.

1. Interface - The camera must have an interface suitable for communication with the LP-180.

2. Frame rate - A high frame rate is needed for future projects that will utilise this system.
3. Resolution - A high resolution is beneficial for robust feature detection.
4. Global shutter - A global shutter exposes the entire frame at once eliminating distortion from fast motions or vibrations.
5. Size, weight and ease of mounting - Low size and weight is desirable along with a simple method of mounting the module.
6. Chroma - A colour camera is beneficial over a monochromatic camera for feature identification.
7. Low light performance - This improves with a larger sensor size which enhances feature identification robustness.
8. Power - Low power draw is desirable, alongside the capability of being powered through the processing board.
9. Lens mount - Support for multiple lens mounting interfaces increases the range of lenses that can be used.

If cameras did not have USB or GB Lan interfaces, did not have global shutter or did not support RGB colour they were discarded immediately. Frame rate and resolution had to be balanced due to bandwidth constraints, particularly over the USB interface. The remaining metrics were used to help with final decisions. A shortlisted selection of cameras is shown in Table 4.2. The shortlisted cameras were the PointGrey FireFly and Flea3 (PointGrey, 2013), IDS UI-1221LE (IDS, 2013) and the mvBlueFox-MLC (MATRIX VISION GMBH, 2013). All of the shortlisted cameras were colour cameras. Power usage and supply requirements were generally negligible and are not shown in the table, however the PointGrey Flea3 did require an external power supply.

The main disadvantages of the PointGrey FireFly was its low resolution when running at a high frame rates, and it was comparitively large and heavy compared to other cameras. The PointGrey Flea3 had a high quality Sony sensor and could run at an impressive 120Hz at 646x488px. This camera was very expensive so it was excluded.

The IDS UI-1221LE-C-HQ and Matrix Vision mvBlueFOX-MLC200wC cameras were nearly identical and used very similar sensors. They both outperformed the PointGrey FireFly in frame rate at their maximum resolution. The mvBlueFox was selected primarily because of its substantially lower cost of $300 including accessories over the IDS camera at over $700. An additional benefit of selecting the mvBlue-Fox was that it offered several lens mount options which gave more flexibility in lens choices. Furthermore, the cameras had an impressively low weight and were relatively low-cost compared to other potential solutions.

Table 4.2: Comparison of cameras

CAMERA	POINTGREY FIREFLY	POINTGREY FLEA3	IDS UI-1221LE	MVBLUEFOX-MLC200WC[*]
Interface	USB/ Firewire	GB Lan	USB	USB
Frame rate min[a]	60	31		
Frame rate max[b]	122	120	87.2	90
Resolution min	320x240	648x488		
Resolution max	752x480	1032x776	752x480	752x480
Shutter	Global	Global	Global	Global
Weight	37g	38g	16g	10g
Sensor size	1/3”	1/3”	1/3”	1/3”
Lens mount	C/CS-mount	C/CS-mount	S-mount	C/CS/S-mount

[*] Chosen camera.

[a] Frame rate when operated at maximum resolution.

[b] Maximum frame rate achievable at minimum resolution without pixel binning.

Several mvBlueFox cameras were obtained along with attachments to support S mount and C mount lenses. Lenses were purchased with focal lengths of 2.8mm and 4mm to provide a reasonable level of versatility. Drivers for the mvBlueFox were installed on the LP-180 which functioned as expected. This camera met all of the key requirements with a suitable resolution and frame rate.

4.3.3 Omnidirectional Lens

The purpose of the omnidirectional lens is to allow the camera to capture images from a 360° panoramic view around the quadrotor. It was critical that the lens provided a good field of view above and below the plane of the vehicle. A typical low cost solution for an omnidirectional vision system is an ultra wide-angle fisheye lens. A more effective solution is a catadioptric lens system where a camera is directed towards an arrangement of mirrors.

The metrics for an omnidirectional lens were identified as:

1. Weight - The weight needs to be low to minimise the impact on vehicle dynamics.
2. Vertical field of view (FOV) - A larger vertical field of view minimises the risk of other vehicles moving out of view.
3. Cost - A low cost solution is desirable due to the high risk of damage.
4. Image quality - Good image quality with low distortion will improve the robustness of feature detection algorithms and give more accurate bearing measurements.

5. Ease of mounting - How easy is it to mount the lens to the camera.
6. Calibration - Calibration should be easy to do, accurate and easily implemented.

Several commercial solutions were investigated alongside a custom solution utilising a spherical mirror and their performance is shown in Table 4.3. A high end solution was a lens from Kumotek (KumoTek, 2013), and a comparatively cheaper solution was the Bloggie Panoramic Lens (Sony, 2013).

Table 4.3: Comparison of omnidirectional camera solutions

METHOD	**KUMOTEK VS-C450MR-TK**	**FISHEYE LENS**	**SPHERICAL MIRROR**[*]	**BLOGGIE PANORAMIC**
Mounting	C/CS mount	C/CS mount	Custom	Custom
Price	$1000	$20	$40	$83
Type	Central	Central	Non-central	Unknown
FOV up	15°	90°	45°	45°
FOV down	60°	2.5°	65°	15°
Image quality	High	Low at edges	Low at edges	Medium

[*] Chosen omnidirectional vision solution.

The Kumotek solution used a custom manufactured parabolic mirror which would provide the highest quality image. However, it was limited by a poor upwards field of view and an unreasonable price. A fisheye lens was not suitable as it would not provide a sufficient field of view below the quadrotor. The Sony Bloggie panoramic lens was relatively low cost and had impressive image quality. Unfortunately its downwards field of view was not ideal, although it was potentially usable. It was decided to design a low cost custom solution to compete with the Sony Bloggie Panoramic Lens.

An omnidirectional camera is classified as central (as opposed to non-central) if the optical rays coming from the camera that are reflected by the mirror intersect at a unique point. An omnidirectional camera can be calibrated to retrieve a projection function which maps image points to bearings on a unit sphere. The majority of calibration techniques rely on the assumption of a central system which is only achievable if using an elliptic, parabolic or hyperbolic mirror. No mirrors could be obtained at low cost and low weight that would have provided a central system so spherical mirrors were assessed.

Spherical mirrors of a wide variety of diameters and curvatures were benchmarked with lenses of varying focal lengths. The interest metrics were the vertical FOV obtainable, the mirror weight and the required distance between the mirror and the camera lens. A major challenge was identifying a mirror that provided a good field of view when placed beyond the minimum object distance from the lens. Fortunately, a configuration was identified which theoretically provided a better field of view than the Sony Bloggie lens. The identified configuration used a 25mm diameter mirror with a 15.5mm radius placed approximately 45mm from a 4mm focal length lens. A physical test was required to verify the FOV and focusing ability in this configuration as the optics could not be perfectly modelled.

A lens system to hold the mirror above the camera was designed as a computer aided design (CAD) model. A laser cut frame holds the camera and provides an attachment point for an acrylic tube which extends to the mirror. The CAD model and assembled omnidirectional lens can be seen in Figure 4.2. The lens had be displaced almost as far away from the sensor as was possible to focus on the mirror at such a short distance.

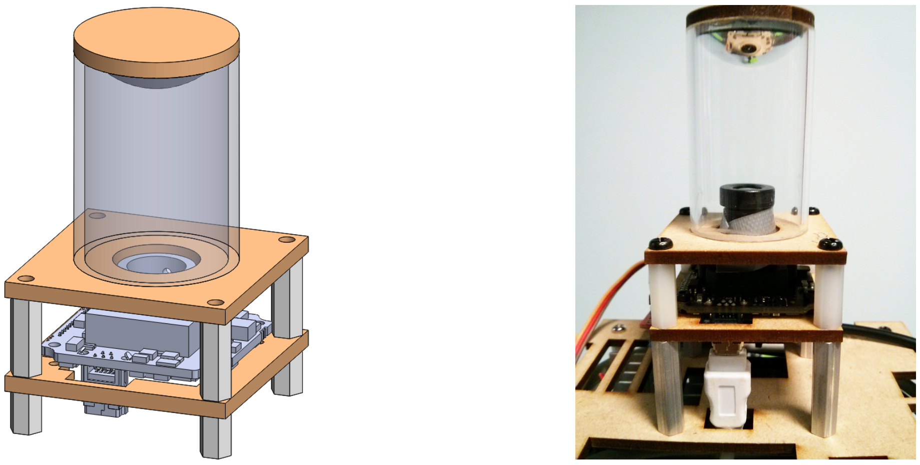
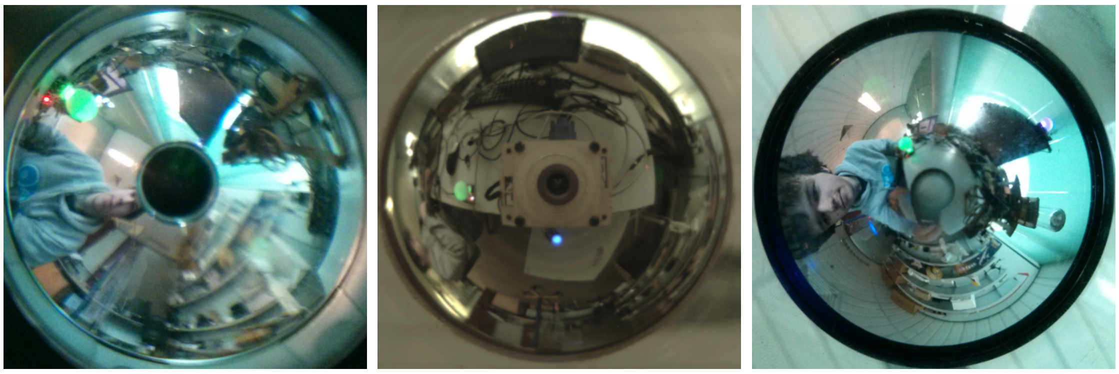


Figure 4.2: Omnidirectional lens CAD design (left) and assembled omnidirectional system with camera (right).

Images obtained with the Sony Bloggie lens and the custom system are shown in Figure 4.3. A large high end parabolic mirror system is also shown for comparison which could not be implemented on a quadrotor. The spherical mirror system image is taken using the mvBlueFox camera inside the custom mount. The Sony Bloggie Lens and parabolic mirror system images were captured with a phone camera. Note that the Sony Bloggie image does not have perfect focusing in this figure which is due to the use of a non ideal lens and misalignment. If used with a Sony Bloggie camera, the Bloggie panoramic lens gives a high quality image over its entire FOV.

The custom solution seen in subfigure (b) of Figure 4.3 was able to identify markers at up to 65° below the vehicle. This was far superior to the Sony Bloggie panoramic lens in subfigure (a) which could only identify markers at up to 15° below the vehicle. The upwards field of view of the two solutions was identical at approximately 45°. The custom solution experienced notable distortion at the edges



(a) Sony Bloggie Panoramic Lens **(b)** Custom catadioptric system with spherical lens **(c)** High end catadioptric system with parabolic mirror

Figure 4.3: Comparison of images from different omnidirectional vision systems

of the image which is a limitation of using a spherical mirror. Blurring at the edges of the image with the spherical mirror solution were due to the small depth of field of the camera lens. Neither of these solutions could compete with the image quality of the parabolic mirror system in subfigure (c).

The custom solution was the lowest cost and was easily assembled and repaired. Distortions and imperfect blurring were not a major issue since the detection was colour based rather than pattern based. For this reason the custom solution was selected for use.

4.4 Vehicle Markers

Vehicle markers are used for unique visual identification of other quadrotors in the environment. The metrics for vehicle markers were:

1. Unique identification - The markers must be able to uniquely identify vehicles.
2. Ease of detection - The markers need to be easy to detect using computer vision.
3. Performance over varying lighting conditions - The markers should be robustly detectable over a reasonable range of lighting conditions.
4. Ease of manufacture and mounting - The markers should be easily mounted and manufactured or configured as appropriate.
5. Weight - The markers should have a relatively low weight.

High level detection processes could not be implemented because of the limited computational power of the processing board. For this reason, simple and highly visible markers were essential. The three most applicable marker types identified were coloured lights, coloured markers and infrared (IR) markers. The performance metrics of the markers were difficult to quantify, so a qualitative analysis was performed.

Infrared markers are easily detectable with an IR camera but are primarily only suitable in indoor environments. There is no simple way to uniquely identify the vehicle corresponding to an IR marker. Detecting a specific marker shape of unique geometric arrangement of markers would be difficult, particularly from a single viewpoint. For this reason, infrared markers were not appropriate.

Coloured lights and coloured markers were the easiest to uniquely detect by assigning a unique colour to each marker. Coloured markers were a simple solution, however, coloured lights had the benefit of improved visibility and robustness over varying lighting conditions so were selected on this basis.

An I2C controlled RGB LED called the BlinkM was selected for use (ThingM, 2013). Its colour could be controlled directly by the PX4FMU or through a USB to I2C convertor with the LP-180. The quadrotor system has multiple 5V sources capable of powering the BlinkM and a 5V coin battery would also be suitable. A marker could be attached to the vehicle above the mirror of the omnidirectional lens for maximum visibility.

These markers were tested in an experiment which is detailed in Section 5.2.3. The markers were stationary and were powered by an Arduino, a low cost and easily programmable microcontroller board. A BlinkM was not implemented on an actual quadrotor as experimentation was only performed with a single quadrotor. Testing showed that dissipating the light with translucent balls was effective for detection from a wide range of angles. However, this was not effective at ranges over 2.5 metres due to the limited resolution of the camera.

4.5 Vicon Motion Capture System

The flying laboratory at ANU has a Vicon motion capture system which is used to measure the position and orientation of objects at 200Hz (Vicon Motion Systems Ltd, 2013). The Vicon motion capture system uses IR cameras to detect objects in its environment. Objects are fitted with IR markers in unique geometric arrangements which allows detection of multiple objects simultaneously. Only 3 markers are required to identify the pose of a vehicle, however more markers allows detection even under occlusion of some markers. Vicon has sub-mm accuracy, but error increases in situations where objects are viewable with a small number of cameras or markers are occluded.

Vicon is a popular tool in robotics as a real-time position sensor and also for logging vehicle positions for analysis. Data can be transmitted from the Vicon ground station over a WiFi or Ethernet connection. The Vicon system was used for obtaining velocity measurements of the quadrotor in real time for input into the algorithm. A number of community contributed ROS packages exist for real time data acquisition which wrap the Vicon Application Programming Interface (API). The Vicon ROS package used and modifications that were made to it are discussed in Section 5.3.

4.6 Inter-vehicle Communications System

The formation control algorithm can use an inter-vehicle communications for communicating velocities. The goals of an inter-vehicle communications system are to have a lightweight close range communication system with high reliability. A single quadrotor system was used for this project, and as such an inter-vehicle communications system was not implemented.

A suitable platform for inter-vehicle communications in future experiments was identified as the WiFly module by Roving Networks (Microchip, 2013). WiFly is a low cost and low profile module that allows wireless communication of serialised data using the Wi-Fi protocol at up to 464kbps. The device is driven through a serial interface which is on both the PX4FMU and the LP-180. Researchers at ANU already use this module to send IMU data from the PX4FMU to a base station.

4.7 Interfacing with Quadrotors

The formation control algorithm requires measurements of the attitude and velocity of the vehicle and implements a control force. The PX4FMU can transmit these measurements as well as take in external control commands using the MAVLink protocol. MAVLink is a protocol for transferring messages which was developed by the PIXHAWK team. The LP-180 connected to the PX4FMU over USB through a USB to UART converter. Alternatively, the serial ports on the LP-180 could have been level shifted and connected directly to the PX4FMU for a small weight saving. This would have required some custom wiring, but was not attempted as the USB solution simply worked. A ROS package is used to handle the MAVLink protocol and communicate over the serial port which is outlined in Section 5.4.

4.8 Power System

The quadrotors have a high power requirement and offer a flight time of less than 10 minutes. It was not reasonable to power the LP-180 with the same battery as the flight time would drop further and continually having to reboot the LP-180 would be problematic. Additionally, voltage fluctuations from varying motor power draw may have caused stability issues. To improve the usability of the system, the LP-180 was powered by a separate battery from the rest of the quadrotor.

A 1300mAh 7.2V Lithium-Polymer (LiPo) battery was used which was regulated to 5V for the LP-180 using a commercial switching regulator. The voltage regulator had major heat dissipation issues despite being rated to 5A (the rated peak draw of the LP-180) and supporting up to 7.5A. This was resolved by mounting the regulator underneath a propeller. The maximum battery life was not tested, however the LP-180 was used through a full length flight test with plenty of charge remaining. LiPo batteries are widely used on quadrotors due to their high energy density, but they can be combust if damaged or not used correctly. The batteries were mounted securely and safe from direct impacts which is described in the next section.

4.9 Mounting System

The mounting system houses the vision system in a module which can be easily attached or detached from the quadrotor. The quadrotor was going to be used in other research projects and so the vision system had to be easily removable. To align with the rest of the goals of this system, the mounting module had to be low cost and low weight. The vision system could not bring the centre of gravity of the vehicle too high as this would negatively affect the stability of the quadrotor. Lastly, the module needed to be easily repairable and replaceable due to the high risk of crashing on a quadrotor.

A custom module was designed to connect to the quadrotor frame and house the vision system. The entire system, including the quadrotor and all supporting components, was drafted using computer aided design (CAD) software. The vision system module mounted on the quadrotor can be seen in Figure 4.4. The design consisted of vertically stacked flat layers separated by spacers which housed the components. The module was laser cut out of Medium-Density fibreboard (MDF) and could be assembled primarily by just screwing spacers between layers and components. Legs were glued to the module which extended to the quadrotor frame. The legs were screwed on to the quadrotor frame which made the module easy to add or remove from the quadrotor.

It was desirable for the mounting system to fail first while protecting the expensive components housed inside. Plastic screws were used in several locations so that the module would simply break off rather than causing jarring impacts. The spacers provided some protection to the important components from the propellers in the

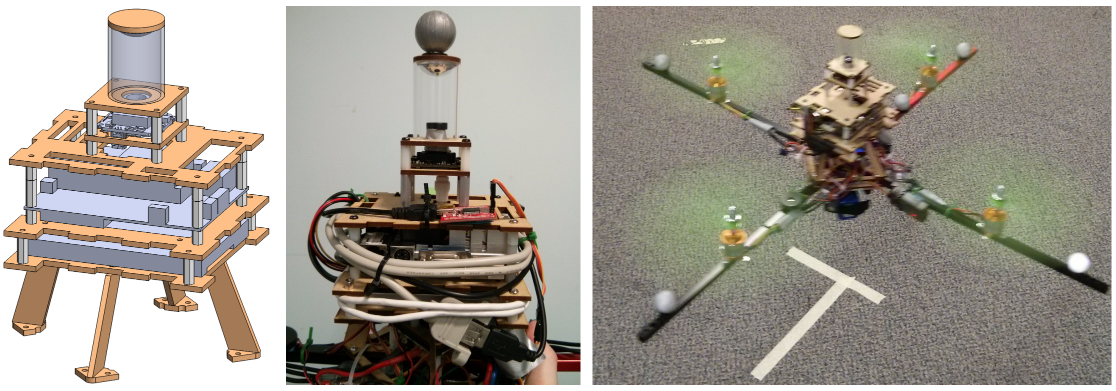


Figure 4.4: Vision system module CAD (left), assembled module (middle) and quadrotor with vision system mounted during flight test (right).

event of a failure.

A custom battery mount was designed to secure the two batteries and also to act as a landing frame. It was required to be relatively tough so it was made out of acrylic providing more strength over MDF while still being relatively cheap. As with the rest of the mounting hardware, it was laser cut providing low cost and rapid replacement.

The quadrotor was tuned for flight with and without the vision system mounted for stable, yet suitably agile flight. A quaternion based attitude controller developed at ANU was used on the PX4FMU as it required tuning of only three gains. Figure 4.4 shows the quadrotor design flying under manual RC with the vision system mounted. This test was just to verify the quadrotor was functioning and the vision system was not yet operational. A flight test in which the vision system is running and all onboard measurements are being obtained is discussed in Section 6.2. The next chapter highlights the software developments that were needed to acquire sensor measurements and interface the vision system with the quadrotor.

Chapter 5

Software Development

5.1 Software Systems Overview

This chapter discusses the software developments that were required for the implementation of the formation control algorithm on the quadrotor. The software developed is built upon an open-source software framework called Robot Operating System (ROS) designed for use with robots (Willow Garage, 2013). ROS abstracts applications into *nodes* which perform functions such as reading sensor data and passing messages between each other through *topics*. *Packages* of *nodes* are available developed by both Willow Garage and the community which provide an array of functionality to a wide variety of robot systems.

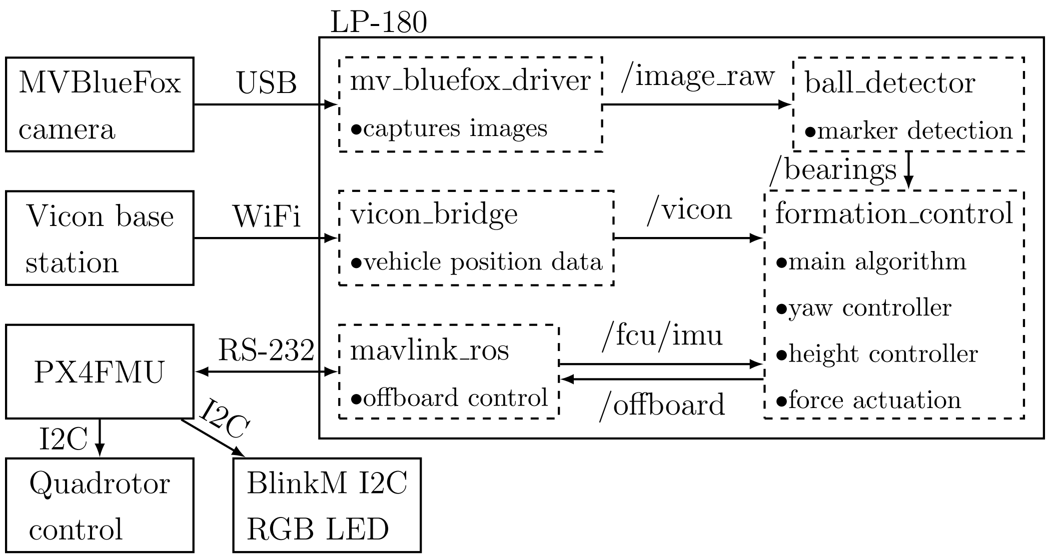


Figure 5.1: Overview of interaction between ROS nodes (dashed) and hardware (solid). Edge labels represent physical interfaces or ROS topics if prefixed with '/'.

An overview of the interactions between hardware and software systems is shown in Figure 5.1. Existing ROS nodes with only minor modifications were used for interfacing with the camera, Vicon and the PX4FMU flight controller. A custom ROS node 'ball_detector' was developed for detecting markers in an omnidirectional image and publishing their bearings. The formation control algorithm and implementation specific controllers are in a custom ROS node called 'formation_control'.

5.2 Bearing Measurements

This section outlines the new developments for obtaining bearing measurements using the camera. An existing camera driver was used, although modifications were made to increase its utility. The omnidirectional lens was calibrated in order to provide a transformation from pixel coordinates into 3D bearings. A simple, fast and robust detection algorithm was developed using OpenCV to detect the pixel coordinates of markers. The marker detection algorithm and bearing conversion code was implemented in a custom node called 'marker_detector'.

5.2.1 Camera Driver

The 'mv_bluefox_driver' package by Carlos Jaramillo was used for obtaining images from the camera (Jaramillo, 2013). The package provides conveniences such as automatic camera initialisation and simple parameter setting over directly using the mvIMPACT Acquire API. A downside of the package was that some important parameters were not exposed which limited the acquisition of images to 50Hz. Modifications were made to directly expose control of gain, exposure time, pixel clock and region of interest (ROI) to obtain images at 90Hz. The modifications can be seen in Appendix B.1.

5.2.2 Camera Calibration

The purpose of omnidirectional camera calibration is to map pixel coordinates to bearings. This mapping must take into account distortions such as a shift in the optical centre and radial distortion alongside standard camera distortions. There are several open-source toolboxes which perform calibration of omnidirectional cameras.

The Omnidirectional Calibration Toolbox from the from the Active Vision Group at the University of Oxford was tested as it indicated support for spherical mirror systems (Mei, 2013). Unfortunately, the automatic corner detection performed poorly and there was no user friendly way of manually picking corners with this toolbox. Furthermore, the provided C++ functions for converting pixel coordinates to bearings were poorly documented, so it was not used.

The omnidirectional camera was calibrated using the OCamCalib Omnidirectional Camera Calibration Toolbox for MATLAB (Scaramuzza, 2013). The toolbox effective calibrated the system and provided well documented C++ functions for pixel to bearing mapping that were easily implemented in the 'marker_detector' ROS node. The calibration should have been poor as it was designed for a central system and the custom omnidirectional system was not central. However, the accuracy was quite good as demonstrated in an experiment detailed in the next section.

5.2.3 Marker Detection

The requirements of the marker detection algorithm were to provide robust and fast detection of multiple coloured markers. The detection process for colour based tracking algorithms is typically done in a colour space where hue is independent of intensity and saturation, such as the HSV colour space. Several existing colour tracking algorithms were investigated, but were not implemented as they were too computationally expensive or not well suited to a catadioptric image.

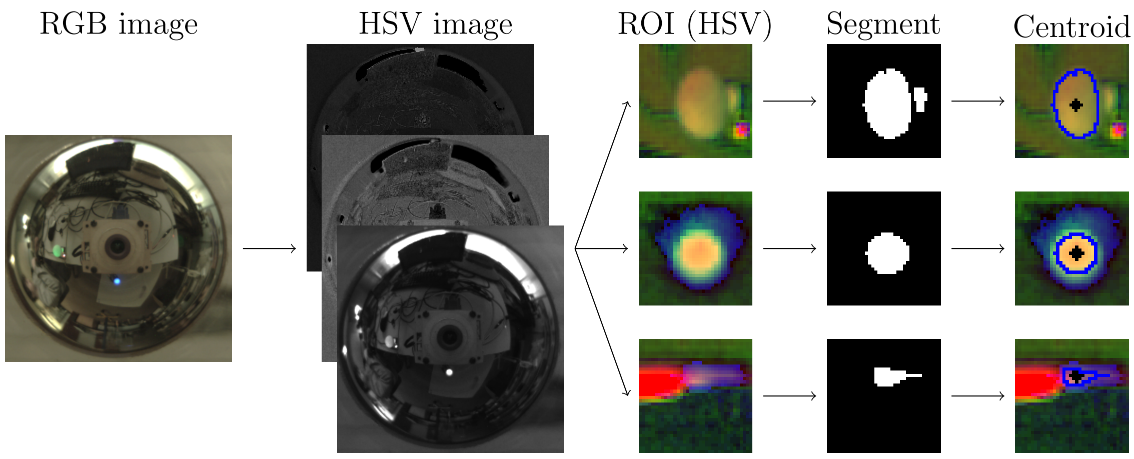


Figure 5.2: Simplified graphical representation of marker detection algorithm.

An algorithm was written in OpenCV to detect markers which is graphically detailed in Figure 5.2. The algorithm was fundamentally very simple in order to run efficiently on the LP-180. First, the tube and camera is masked, then the image is converted into the HSV colour space. A binary segmentation is applied to a Region Of Interest (ROI) around each marker which is chosen based on the location of the previous measurement. The centroid of the largest blob in each marker segmentation was then converted to a bearing and published as a ROS message.

If the segmentation was attempted on the whole image the frame rate was approximately 35Hz on the LP-180 with 3 markers. Carrying out the segmentation on a ROI around the previously detected position improved performance to approximately 55Hz. This was a suitable update rate for the implementation of the formation control algorithm. OpenCV did not yet officially support a GPU based transformation to HSV through OpenCL so all computation was done on the CPU. An output showing the computed centroids for each marker is shown in Figure 5.3. Note that the second marker is only just visible at 45° above the mirror.

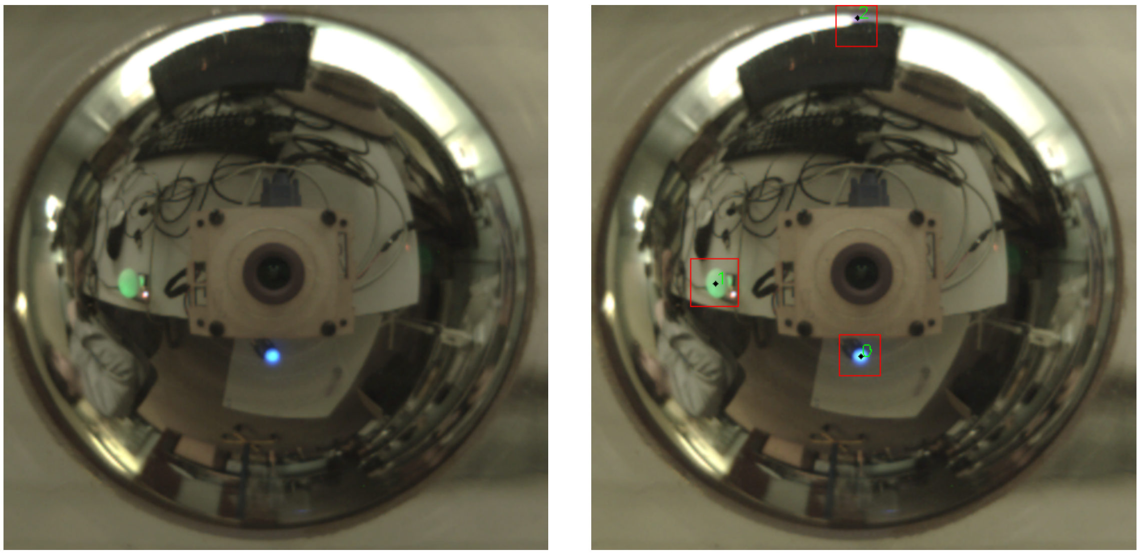


Figure 5.3: Image of marker detection with correct detection of 3 numbered markers placed 60 degrees below, 5 degrees below and 45 degrees above respectively.

The effectiveness of the solution was assessed by comparing measurements obtained with the vision system to ground truth measurements obtained using Vicon. In this experiment, three markers were placed around the Vicon flying space at varying heights and their positions were recorded. The quadrotor was manoeuvred around by hand in the flying and the pose from Vicon was logged. The position of the quadrotor was determined using Vicon and used to compute the relative bearings to the markers decomposed into azimuth and elevation. The bearing measurement from the vision system was transformed from the body fixed frame to the inertial reference frame of the Vicon system using roll and pitch from the PX4FMU and yaw from Vicon. The reason these measurements were used is discussed later in Section 5.4.

Figure 5.4 shows the error between bearing measurements obtained from the vision system and ground truth measurements obtained from Vicon data. Missing data points occur where a marker is not observable to the camera. The azimuth shows good accuracy throughout the experiment with a mean error of $-0.83°$ (SD $\pm 1.23°$). The largest inaccuracies observable in the figure occurred when there was a large delay in the yaw measurement from Vicon.

The elevation error is shown in Figure 5.5. The elevation shows good accuracy despite the calibration not being ideal for a spherical mirror system. The mean elevation error was $0.71°$ (SD $\pm 1.54°$). This experiment had several sources of error

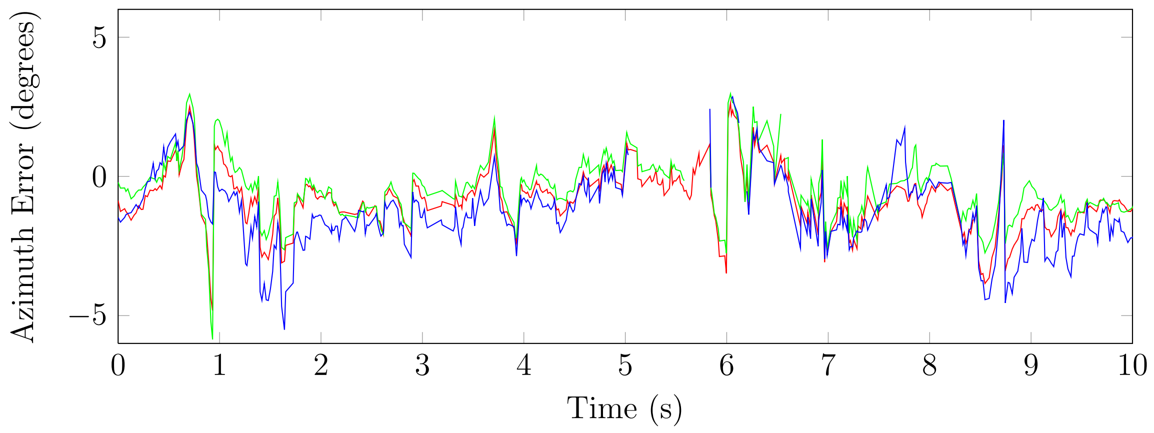


Figure 5.4: Error in azimuth bearing measurements between the vision system and Vicon.

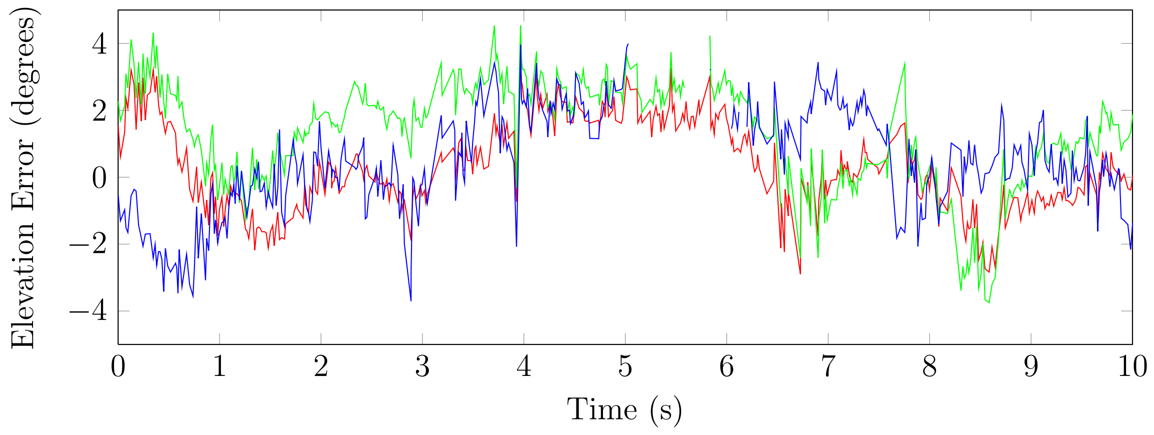


Figure 5.5: Error in elevation bearing measurements between the vision system and Vicon.

such as limited image quality and imprecisely measured marker positions.

There are occasional misdetections which lead to large deviations from the true value. These were relatively rare and the robustness during a long flight test can be observed in Section 6.2. Misdetections are not completely avoidable with such a low image resolution and simple computer vision technique. To counteract this issue, false bearings could be detected by thresholding based on the deviation from previous bearing measurements and discarded if appropriate. Misdetections were rare enough that this was not implemented, however it would be simple solution if misdetections occured over small periods. Synthetic measurements from Vicon could have been used in the event of an extended failure.

5.3 Vicon Velocity Measurements

Velocity measurements are needed to compute the $\dot{r}$ measurement between vehicles. This is a difficult measurement to obtain reliably on an aerial vehicle, so Vicon was used to retrieve these measurements. The 'vicon_bridge' ROS package by Markus Achtelik was used to interface with the Vicon system (Achtelik, 2013). The package receives pose measurements, but does not receive direct velocity measurements.

The difference in time between measurements must be known to compute velocity. Timestamps on received messages are unsuitable because measurements are received at time-varying varying rates over the channel. The latency compensated timestamps in 'vicon_bridge' showed similar levels of noise as just using arrival timestamps. ANU researchers previously assumed Δt to be constant which provided better results than using the timestamp methods.

This assumption was reasonable when using a wired link to Vicon. Unfortunately, this assumption lead to large spikes when using a wireless channel where measurements are often dropped and packets can even arrive in the wrong order. Additionally, measurements are also not published if a vehicle is occluded. The quadrotor implementation for this project was the first system in use at ANU which had to retrieve Vicon measurements over a wireless link without an intermediate ground station. A new method of computing velocity was investigated to provide more robust velocity measurements over the other methods.

Vicon publishes an incrementing frame number with each measurement. This number is retrieved by 'vicon_bridge', although it is not included in the message containing full pose measurements. Under the assumption that Vicon takes a measurement at a constant rate (every 0.005s), the change in time could be computed based on the change in the frame number. A simple modification to 'vicon_bridge' replaced the timestamp present in the header of the message with the Vicon frame number as shown in Listing 5.1.

Listing 5.1: Code appended after line 488 of vicon_bridge/src/vicon_bridge.cpp.

```
pose_msg->header.stamp=(ros::Time)lastFrameNumber;
```

Figure 5.6 shows a comparison between the new and old methods during a period of 8% packet loss. For comparison, there was a period of 50% packet loss over 10 seconds which produced spikes in the other methods of up to $110m/s$. It is clear that the new method is far superior to the previous method with comparatively negligible noise and the assumption that Vicon took measurements at a constant rate was valid. Despite the simplicity of this method, it can provide very robust measurements.

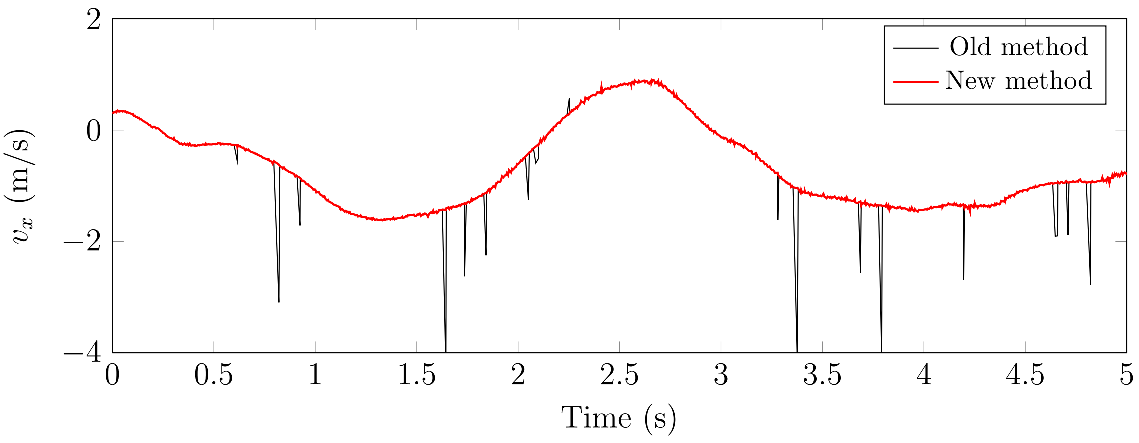


Figure 5.6: Comparison of velocity measurement methods during a period where 8% of measurements were dropped.

Vicon bridge offered two modes of operation, 'ClientPull' and 'ServerPush'. Measurements obtained using 'ServerPush' often arrived in bursts of 2-4 which gave an effective measurement rate of approximately 50Hz rather than 200Hz. When using 'ClientPull', the measurements arrived at a relatively steady rate although with a higher latency as mentioned by the vicon_bridge developer. The delay was not quantified, however in later experiments there was no observable delay when compared with attitude measurements from the PX4FMU.

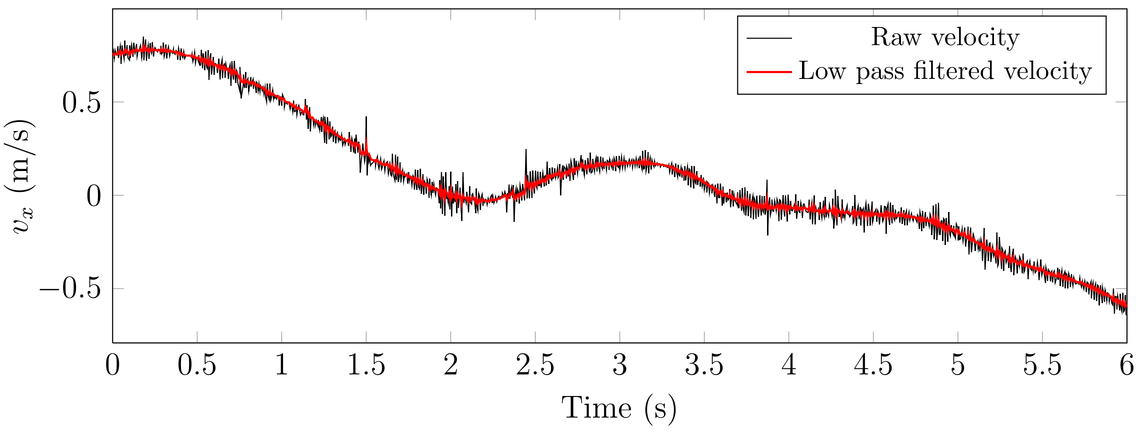


Figure 5.7: Comparison between raw velocity measurement based on frame number and low pass filtered measurement during actual flight test.

A key observation was that velocity measurements were relatively smooth if motors were off and a vehicle was moved manually which was the protocol of the experiment in Figure 5.6. The noise level increased in an actual flight test as can be seen in Figure 5.7, although it was still relatively small. This experiment was conducted on a quadrotor being used in other research. This noise was likely the result of vibrations induced by the motors and was reduced by applying a low pass filter to the incoming measurements (see Figure 5.7).

5.4 Attitude Measurements

Vehicle attitude measurements are required in computing the transformation between the body fixed frame of the vehicle and the inertial reference frame. IMU measurements from the PX4FMU were obtained using the mavlink_ros ROS node provided by the pixhawk team. The node publishes IMU information that is transferred from the PX4FMU over the MAVLink protocol using the ATTITUDE message type. Attitude measurements were obtainable at 20Hz at a baud rate over the serial interface of 115200Hz. This is an imposed rate in the PX4FMU associated with that baud rate, although higher baud rates were not tested.

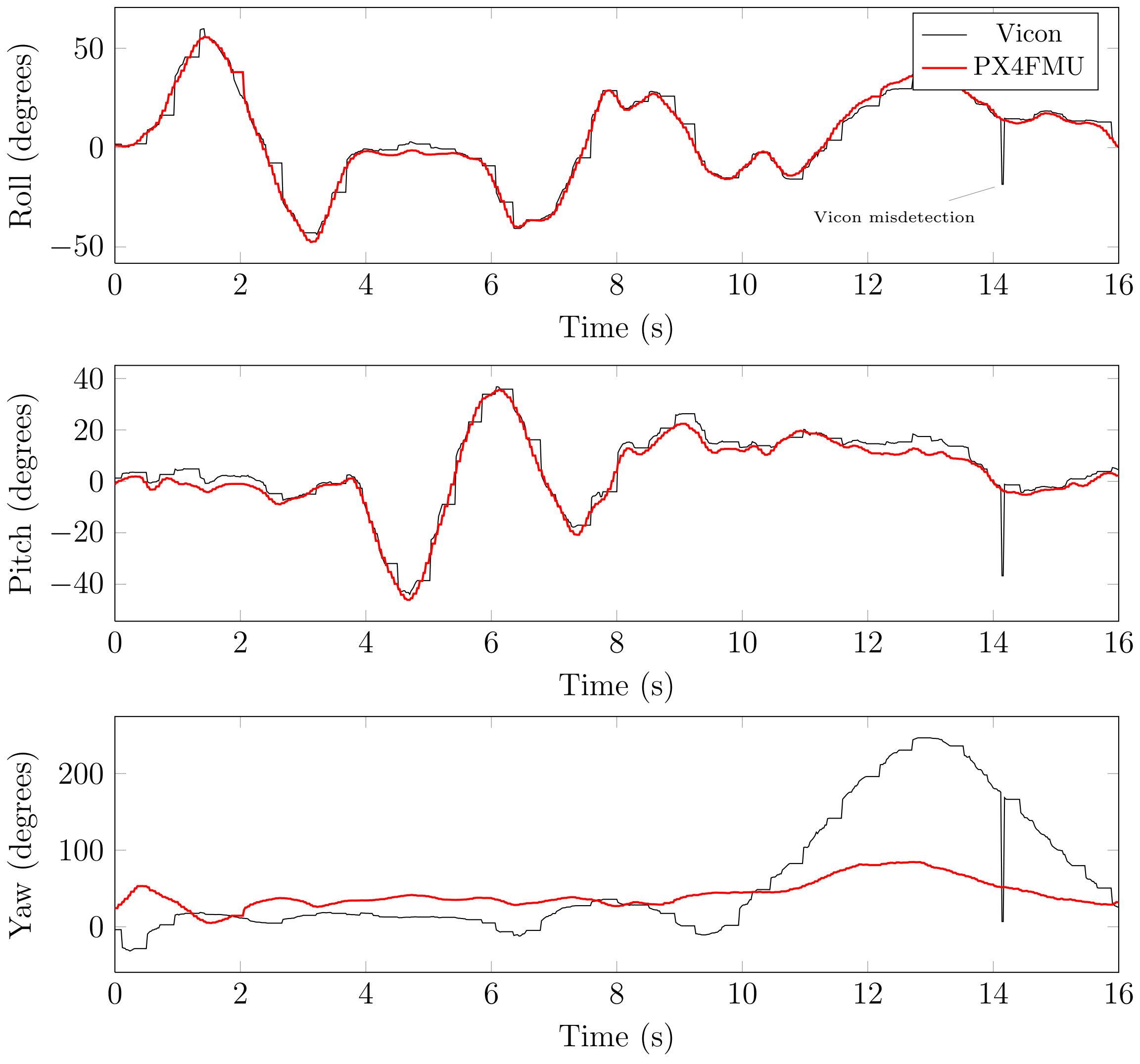


Figure 5.8: Attitude measurements obtained from Vicon and the PX4FMU.

The inertial reference frame differs between the IMU and Vicon such that they give different yaw measurements, however pitch and roll measurements are directly comparable. Figure 5.8 shows a comparison between roll, pitch and yaw measurements obtained from the PX4FMU and Vicon. The PX4FMU measurements do not experience communication dropouts or large noise spikes as was observed when Vicon mistakenly detected another object in the environment. The Vicon measurements indicated that the pitch and roll measurements from the PX4FMU were accurate. However, the yaw measurements from the PX4FMU did not match the shape of the real yaw of the system as indicated by Vicon (see Figure 5.8).

Yaw measurements on the PX4FMU are obtained from an extended Kalman filter which combines measurements from onboard gyroscopes, accelerometers and magnetometers. It is likely that the magnetometer is receiving inaccurate measurements because of its close proximity to the SSD. Since the yaw from the PX4FMU is unsuitable, the yaw from Vicon is used in computing the transformation from the body fixed frame of the camera to the Vicon frame. The PX4FMU pitch and roll measurements are also used to compute the transformation due to their consistent time of arrival and high accuracy.

5.5 Height and Yaw Controller

It was desirable to decouple height from the algorithm for early testing so that the algorithm was unaffected by oscillation in height and bias in bearing elevation measurements. A modified Proportional-Derivative (PD) controller to stabilise the quadrotor at a suitable height is suggested. The equation of the PD controller is

$$T = K_p e(t) + K_d \frac{d}{dt} e(t) + K_{grav}, \tag{5.1}$$

where T is the thrust, $e(t) = p_z^* - p_z$ is the error from the desired height, K_p is the proportional gain, K_d is the derivative gain and K_{grav} is a feed-forward term to counteract gravity.

This simple controller ignored effects including reduced motor power (as battery voltage dropped) and ground effect. This controller was not tested due to time constraints, but was verified in simulation. The gains would need to be determined experimentally for an implementation. In future, the gravity compensation component of the formation control algorithm could be used which will require testing to relate the thrust set point to the applied thrust.

Another controller was suggested to correct the vehicle yaw to a desired yaw in the Vicon frame defined as

$$\psi_{sp} = \psi_{IMU} + K \left(\psi_{Vicon}^* - \psi_{Vicon} \right), \tag{5.2}$$

where ψ_{sp} is the yaw setpoint, ψ_{IMU} is the yaw from the IMU, ψ_{Vicon} is the yaw measured in Vicon, ψ^*_{Vicon} is the desired yaw in Vicon and K is a proportional gain. This controller was also not tested because of time constraints.

5.6 External Control

Previously researchers implemented control algorithms by sending control commands through an RC controller connected to a base station. To implement the formation control algorithm, a method of sending control commands without a base station was required. The MAVLink protocol supported communication of external control measurements which could be done through a serial interface between the LP-180 and the PX4FMU. However, the PX4FMU state machine for handling flight modes did not handle offboard control messages, despite the onboard MAVLink and actuator modules supporting it.

A new state was implemented to enable offboard control. It took care of arming the vehicle, setting the flight mode, and enable control of the actuators with offboard signals. The state machine was designed so that an RC controller could instantly take control by toggling a switch. In the event of a dropout, the quadrotor would immediately stabilise and switch to RC until autonomous control was re-enabled with the RC controller.

The 'mavlink_ros' node was used for sending the MAVLink control messages. The roll, pitch, yaw and thrust setpoints of the vehicle could be set using the SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST MAVLink message type if offboard control is enabled. These setpoints are parsed into an attitude controller on the PX4FMU which applies a rate controller. A small modification to the 'mavlink_ros' package was applied to directly publish the control messages rather than utilise the generic MAVLink message publishing function. This modification was introduced because of a issue in producing the correct MAVLink packet structure in the formation_control node that was not identified.

5.7 Actuation of Control Force

The formation control algorithm was implemented in a custom node called 'formation_control'. This node read in the bearing, attitude and velocity measurements obtained from other nodes and computed the desired control force. The equations applied to compute the control force are are summarised in Figure 3.5 (velocity damping) and Figure 3.8 (image space damping).

The PX4FMU supported offboard control signals for setting attitude or rate setpoints. The control force cannot be directly actuated by the quadrotor and must be indirectly applied by manipulating the vehicle attitude. An advantage of quadrotors is that when they are close to level the thrust is decoupled from pitch and roll. With this assumption and assuming the height is essentially constant as imposed by the height controller, the thrust force was approximated as $F_T = mg$.

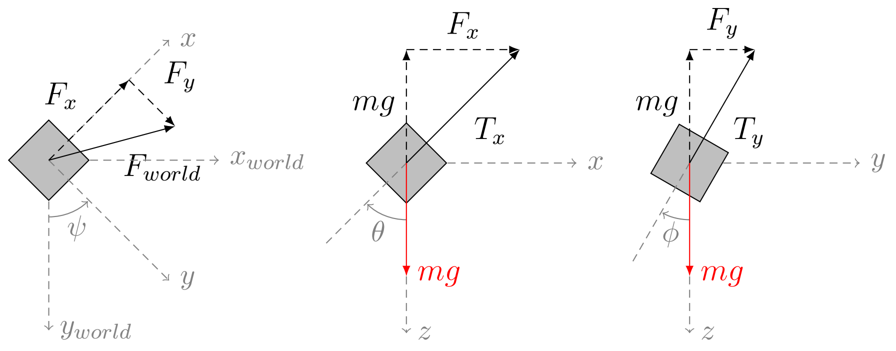


Figure 5.9: Free body diagram showing how control forces are applied depending on the yaw ψ, pitch θ and roll ϕ of the vehicle.

The free body diagram shown in Figure 5.9 indicates how directional forces are applied based on the roll, pitch, yaw and thrust of the vehicle. The equations for conversion from a control force vector to roll, pitch, yaw and thrust set points can be derived geometrically. The control force was transformed so that it is defined by the x and y axis of the body fixed frame based on the Vicon yaw measurement.

$$F_x = \cos(\theta)F_{world,x} - \sin(\theta)F_{world,y} \tag{5.3a}$$
$$F_y = \sin(\theta)F_{world,x} + \cos(\theta)F_{world,y} \tag{5.3b}$$

Then the required pitch and yaw is computed according to the relations

$$\sin(\theta) = \frac{F_x}{mg} \tag{5.4}$$
$$\sin(\phi) = \frac{F_y}{mg}. \tag{5.5}$$

For safety reasons, the maximum pitch and roll rate was limited to 10 degrees to reduce the maximum acceleration and ensure markers were always in view of the quadrotor. This method does not take into account under-actuation of the vehicle, but for the purposes of validating the formation control algorithm, it is a reasonable approximation. This simple method will need adaptation for an experiment where height is not held constant.

Chapter 6

Experiments and Simulations

6.1 Testing Methodology

The purpose of a physical implementation of the algorithm was to give an insight in to practical limitations such as sensor inaccuracy and imperfect actuation. Previous chapters have discussed the developments of the sensor systems individually and in controlled tests. An experiment to test all of the sensor systems during a flight test is discussed below. Simulations of the algorithm aimed at analysing the behaviour in scenarios not testable with only one quadrotor is described in Section 6.3. This chapter concludes with a discussion on what additional tests are needed prior to flying the quadrotor with the formation control algoithm.

6.2 Flight Test

The experimental configuration described in Section 5.2.3 was utilised in an actual flight test. Figure 6.1 shows the experiment environment. Data from the camera, Vicon, the PX4FMU and processed measurements were logged during this experiment. A video of the flight test which demonstrates the effectiveness of the real time marker detection can be viewed at `http://www.youtube.com/watch?v=ISheEfqbyxc`. Plots of the data obtained over the entire experiment are found in Appendix A.

In a 50 second flight test, only one marker was misdetected for less than 0.1 seconds which demonstrated the robustness of the computer vision algorithm. Markers were correctly reported as not visible rather than erroneously being detected elsewhere when viewed at extreme angles. Logging the video data reduced the marker detection rate by 6Hz to an average of 49Hz compared with performance when video data was not logged. This performance impact was relatively small which can be attributed to using an SSD for data storage.

The time of arrival of Vicon measurements was extremely sporadic which was the primary source of measurement error in this experiment. On one occasion, no measurements arrived from Vicon for around two seconds. The accuracy of bearing

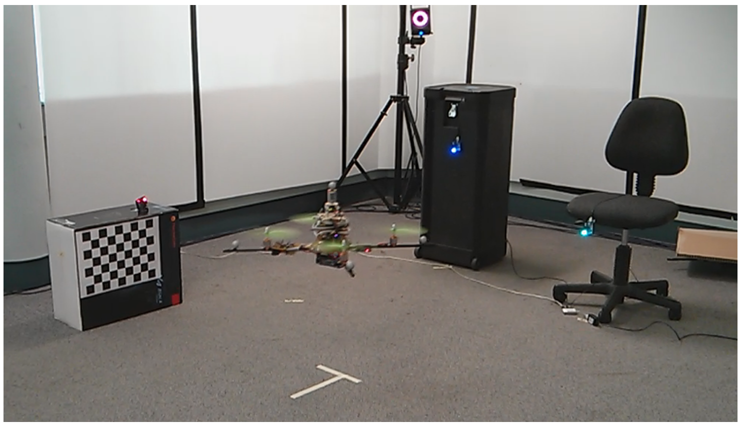


Figure 6.1: Sensor test experiment environment.

measurements was negatively impacted as they were dependent on yaw measurements from Vicon (see Section 5.4). It is suspected that the main cause of these delays was occlusion of markers by the propellers and objects in the environment rather than packet loss. Alternative positioning of Vicon markers should be investigated in the future to reduce the rate of occlusions.

Velocity measurements were also dependent on Vicon and the delay in measurements was clearly observable. Fortunately, the data did not experience large spikes and remained usable because of the modifications to the Vicon driver described in Section 5.3.

The roll and pitch measurements obtained from Vicon were far too sporadic to use for conversion of bearings to the Vicon reference frame. Roll and pitch measurements from the PX4FMU were accurate and arrived robustly as was previously demonstrated in Section 5.4. There was a bias of less than 1° in both measurements which may have been correctable with a recalibration of the accelerometers. Using these measurements when computing the transformation from the body fixed frame to the Vicon reference frame was effective as demonstrated by the accuracy of the bearing measurements.

Due to time constraints, the height and yaw controller was not tested and so the quadrotor was not yet ready to be flown under control of the algorithm. The following section simulates the formation control algorithm in a similar geometric configuration to what was shown in Figure 6.1. More complex experimental configurations with multiple vehicles are also discussed.

6.3 Algorithm Simulations

A simulation environment for the formation control algorithm was developed in MATLAB and the code is found in Appendix D. The simulation was not intended to model the dynamics of the quadrotor or performance of the sensors, but rather to investigate the global stability and performance of the formation control algorithm. Quadrotors were modelled as point masses which could fully actuate control forces imposed by the formation control algorithm.

The simulation provided an environment to test the interaction between a large number of vehicles which was not feasible to physically test. Appropriate gains for the formation control algorithm were safely determined using the simulation and variations to the algorithms were evaluated. A vehicle configuration was designed where vehicles were highly displaced from their goal bearings. The vehicles were initially arranged in a rectangle with no initial velocity. The range estimate at the start of the experiment was assigned to the correct range. The goal bearings form a right triangle with one vehicle sitting at the middle of the hypotenuse of the triangle.

Figure 6.2 shows the trajectory and bearing error over time in this configuration using the velocity damping algorithm. Correct convergence to the desired formation is demonstrated, confirming the validity of previous work by Stacey and Mahony (2013). The bearing error converges to a minimum at an exponential rate with no observable overshoot because of the velocity damping. Although the algorithm is invariant to scale, expansion and contraction of the formation will eventually stop due to the velocity damping.

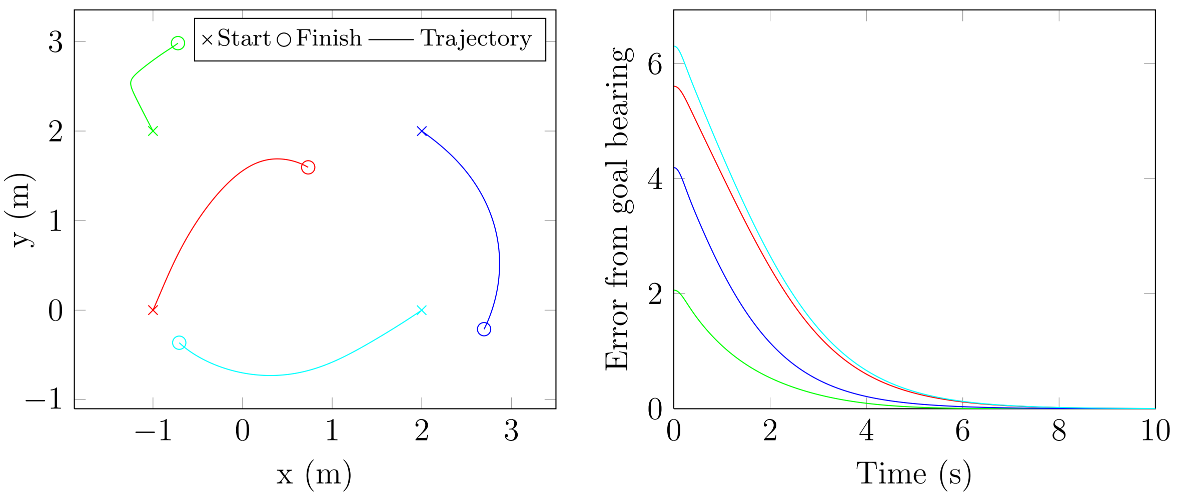


Figure 6.2: Velocity damping algorithm trajectories and bearing errors over time.

Figure 6.3 shows the same vehicle configuration with the image space damping algorithm applied. The gains were tuned for similar performance of bearing error over time to the velocity damping algorithm. The algorithm successfully converged to the goal formation which demonstrated that the algorithm performed to expectations and was a suitable alternative to the velocity damping algorithm. The formation expanded substantially leading to a large difference in the trajectory than

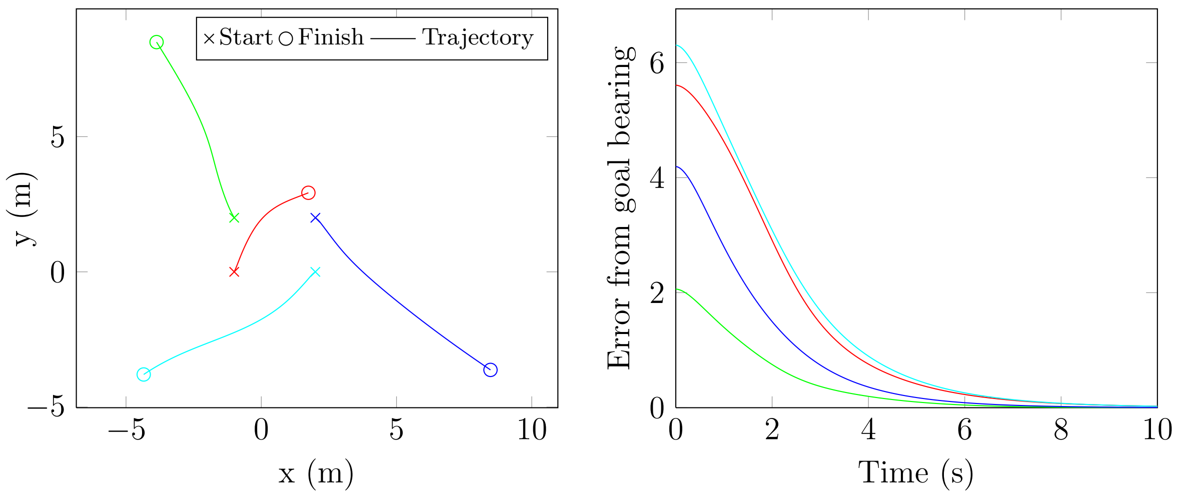
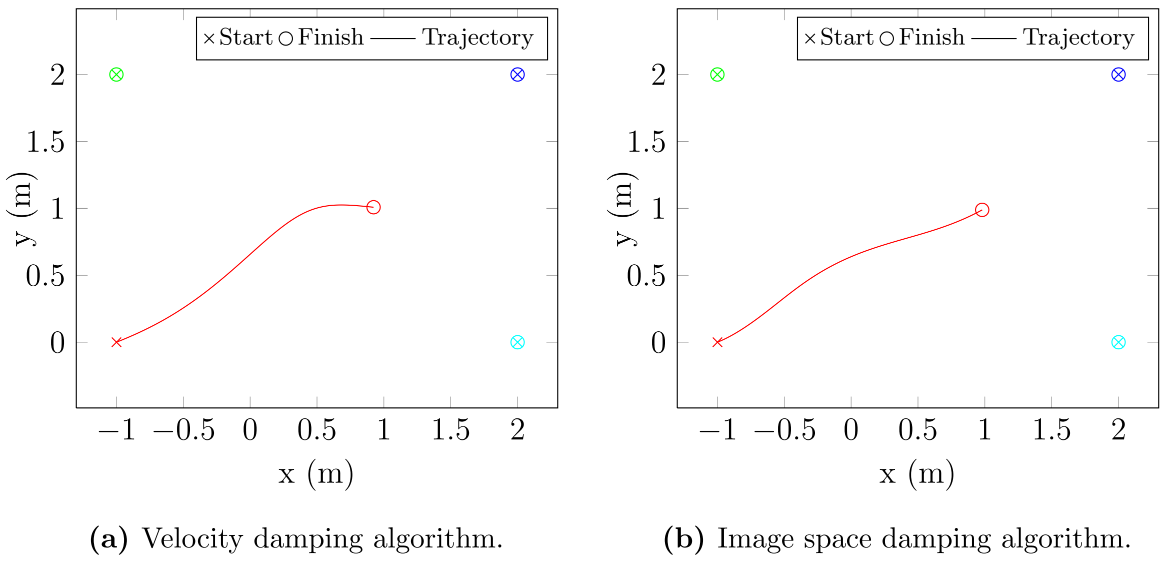


Figure 6.3: Image space damping trajectories and bearing errors over time.

what observed with the velocity damping algorithm. The expansion of the formation was uncontrolled without any velocity damping.

Stacey and Mahony (2013) suggest a vehicle topology which includes some range sensors to control the scale of the formation. In a practical implementation, a low accuracy position sensor could be used to estimate the growth rate of the formation and control it. If c_k was set too low in the image space damping algorithm the algorithm appeared to expand indefinitely without convergence as the control force became increasingly negligible. This issue was resolved by setting the gains to match the expected scale of the formation.



(a) Velocity damping algorithm. **(b)** Image space damping algorithm.

Figure 6.4: Simulated trajectories of the algorithms in the physical experiment configuration.

Figure 6.4 shows a simulation of a suggested experimental configuration with both algorithms. Four vehicles start in a rectangle where one vehicle is able to move and the other three are fixed in place. Using fixed vehicles breaks the scale invariance of the formation. The goal bearings are defined so that the goal position is (1,1) for the free vehicle. The algorithm gains were tuned to give appropriate vehicle velocities and safe trajectories. The vehicle correctly converged to the goal formation position in both algorithms. The performance was comparable, although, the trajectories were slightly different because of the change in damping method.

A few extreme conditions were tested in simulation. If $\hat{r}$ was initially set much smaller than the true range, then the effective gains in the system became very large and the system response was unpredictable. This behaviour is avoidable in a practical implementation by setting the range estimate larger than the expected initial range or by bounding the range estimate to sensible values, although breaks the passivity of the system. In certain configurations where vehicles would move close together, very large forces were observed. When this occurred, vehicles would fly off in unpredictable directions at high velocities. This indicates the necessity of a vehicle avoidance controller for this algorithm.

6.4 Algorithm Implementation

The quadrotor was not flown under the control of the formation control algorithm. The remaining tests and developments that were required prior to performing this experiment were to:

- Tune the height and yaw controllers and verify suitable performance.
- Verify that the control commands from the formation control algorithm were being actuated in the correct directions by the PX4FMU.
- Investigate alternative positioning of Vicon markers to minimise occlusions.
- Implement observers for velocity and yaw to handle long communication dropouts with Vicon.

The proposed experiment to verify the algorithm was fundamentally very simple because it would be applied with only one quadrotor rather than a formation. Flying the quadrotor with the formation control algorithm would have provided a test of the accuracy of the actuation of control forces. It would also have fully verified that the formation control algorithm was feasible to implement on an aerial vehicle. As detailed in this thesis, the performance of the sensor systems required for the formation control algorithm could be effectively assessed by flying the vehicle manually.

Chapter 7

Conclusions and Further Work

7.1 Conclusions

A high-end quadrotor was built based on a reference quadrotor design used at the ANU and the flight controller was modified to support offboard control. A custom high performance vision system was designed and appended to the quadrotor which utilised a custom omnidirectional lens with a better vertical field of view than similarly priced commercial solutions. A computer vision algorithm was developed which robustly detected vehicle markers at high speed and provided bearing measurements with suitable accuracy. A driver for the Vicon motion capture system was modified to give robust velocity measurements in the presence of dropped packets. A modification to the algorithm was proposed and tested in simulation which eliminated the direct dependence on velocity measurements. A flight experiment on the custom-built quadrotor demonstrated the suitability of all onboard sensor measurements for use with the formation control algorithm. Further experimentation is required to test proposed height and yaw controllers and to verify that control forces are correctly actuated before the quadrotor can be flown under the control of the algorithm.

7.2 Further Work

The sensor systems that were designed and tested on the custom-built quadrotor worked effectively. The remaining steps to be completed before the formation control algorithm can be used to control the quadrotor are discussed in Section 6.4. Once these are completed and the formation control algorithm works on a single vehicle, more quadrotors can be built which will allow a more comprehensive physical test of the algorithm.

A long-term goal is to extend the implementation outside the laboraty environment and into the field. Moving outdoors will require removing all dependence on Vicon, which means finding an alternative sensor to compute the range velocity $\dot{r}$.

The method of actuating control forces on the quadrotor was straightforward and utilised a simplified vehicle model which linked the actuation of control force to the attitude of the vehicle. The effectiveness of this method will need to be tested in the future. The actuation performance may be improved by designing the algorithm for an under-actuated model a quadrotor rather than a point mass model.

Several features may need to be added to the formation control algorithm for it to be competitive with other algorithms. These include object and vehicle avoidance, teleoperation and trajectory navigation. The effect of sensor measurement delays and loss of vision of other vehicles also needs to be considered. As previously mentioned, observers could be used to improve the robustness of measurements that arrive unpredictably, such as those from Vicon. Further analysis of the risk of potential stability and performance issues with the algorithm that have been brought up in this thesiere needs to be fs may also be warranted.

ANU is currently investigating high performance motor control and acrobatics. The current vision system weighs too much to be used on a vehicle for these applications. Also, the processing board used for the vision system in this project may not have enough power to use the camera at a high frame rate with more complex computer vision algorithms. Processing boards have been shrinking and becoming more powerful since this projects inception in early 2013 and may soon be able to support this kind of research.

References

Achtelik, M., 2013. vicon_bridge ROS. http://wiki.ros.org/vicon_bridge.

AutoQuad, 2013. AutoQuad ESC32. http://autoquad.org/esc32/.

Balch, T. and Arkin, R., 1998. Behavior-based formation control for multirobot teams. *IEEE International Conference on Robotics and Automation*, 14, 6 (1998), 926–939.

Beard, R. W.; Lawton, J.; and Hadaegh, F. Y., 2001. A coordination architecture for spacecraft formation control. *IEEE Transactions on Control Systems Technology*, 9, 6 (2001), 777–790.

Berghuis, H. and Nijmeijer, H., 1993. Global regulation of robots using only position measurements. *Systems & Control Letters*, 21 (1993), 289–293.

Borutzky, W., 2006. Bond graph modelling and simulation of mechatronic systems. An introduction into the methodology. *Proceeding 20th European Conference on Modeling and Simulation (ECMS)*, (May 2006), 17–28.

Chaumette, F. and Hutchinson, S., 2006. Visual servo control. I. Basic approaches. *IEEE International Conference on Robotics and Automation*, 13, 4 (Dec. 2006), 82–90.

Chaumette, F. and Hutchinson, S., 2007. Visual servo control. II. Advanced approaches. *Robotics and Automation*, 1, March (2007), 109–118.

Chen, Y. and Wang, Z., 2005. Formation control: a review and a new consideration. *IEEE/RSJ International Converence on Intelligent Robots and Systems*, , 435 (2005), 3181–86.

COMMELL, 2013. COMMELL LP-180. http://www.commell.com.tw/Product/SBC/LP-180.HTM.

Corke, P. I. and Hutchinson, S., 2001. A new partitioned approach to image-based visual servo control. *IEEE International Conference on Robotics and Automation*, 17, 4 (2001), 507–515.

DAS, A.; FIERRO, R.; AND KUMAR, V., 2002. A vision-based formation control framework. *IEEE Transactions on Robotics and Automation*, 18, 5 (2002), 813–825.

FRANCHI, A.; MASONE, C.; GRABE, V.; RYLL, M.; BULTHOFF, H. H.; AND GIORDANO, P. R., 2012. Modeling and Control of UAV Bearing Formations with Bilateral High-level Steering. *The International Journal of Robotics Research*, 31, 12 (Oct. 2012), 1504–1525.

GUMSTIX, 2013. Gumstix Open Source Products. `https://store.gumstix.com/index.php/category/27/`.

HAMEL, T. AND MAHONY, R., 2002. Visual servoing of an under-actuated dynamic rigid-body system: an image-based approach. *IEEE International Conference on Robotics and Automation*, 18, 2 (2002), 187–198.

HATANAKA, T. AND IGARASHI, Y., 2012. Passivity-based pose synchronization in three dimensions. *Automatic Control, IEEE ...*, 57, 2 (2012), 360–375.

HUTCHINSON, S.; HAGER, G.; AND CORKE, P. I., 1996. A tutorial on visual servo control. *IEEE International Conference on Robotics and Automation*, 12, 5 (1996), 651–670.

IDS, 2013. IDS. `http://en.ids-imaging.com/store/ui-1221le.html`.

JARAMILLO, C., 2013. mv_bluefox_driver ROS. `http://wiki.ros.org/mv_bluefox_driver`.

KUMOTEK, L., 2013. KumoTek Robotics. (2013). `http://kumotek.com/products/omni.htm`.

LEONARD, N. AND FIORELLI, E., 2001. Virtual leaders, artificial potentials and coordinated control of groups. *... and Control, 2001. Proceedings of the ...*, (2001).

MAHONY, R. AND STRAMIGIOLI, S., 2012. A port-Hamiltonian approach to image-based visual servo control for dynamic systems. *The International Journal of Robotics Research*, 31, 11 (Sep. 2012), 1303–1319.

MATRIX VISION GMBH, 2013. mvBlueFOX-MLC. `http://www.matrix-vision.com/USB2.0-single-board-camera-mvbluefox-mlc.html`.

MEI, C., 2013. Omnidirectional Calibration Toolbox. `http://www.robots.ox.ac.uk/~cmei/Toolbox.html`.

MICHAEL, N.; FINK, J.; AND KUMAR, V., 2011. Cooperative Manipulation and Transportation with Aerial Robots. *Autonomous Robots*, 30, 1 (2011), 73–86.

MICROCHIP, 2013. WiFly - Wi-Fi. `http://www.microchip.com/pagehandler/en-us/technology/wifi/software/wifly.html`.

PIXHAWK, 2013. PX4FMU Autopilot. `https://pixhawk.ethz.ch/px4/modules/px4fmu`.

POINTGREY, 2013. PointGrey. `http://www.ptgrey.com/`.

REN, W. AND BEARD, R., 2004. Decentralized Scheme for Spacecraft Formation Flying via the Virtual Structure Approach. *Journal of Guidance, Control, and Dynamics*, 27, 1 (Jan. 2004), 73–82.

SCARAMUZZA, D., 2013. OCamCalib. `https://sites.google.com/site/scarabotix/ocamcalib-toolbox`.

SCHAFT, A., 2006. Port-Hamiltonian systems: an introductory survey. *International Congress of Mathematicians*, (Aug. 2006). `http://doc.utwente.nl/66742/`.

SELTECH INTERNATIONAL, 2013. SECOnITX-ION. `http://www.seltech-international.com/en/Products/Embedded-Computers/Single-Board-Computers/215-SECOnITX-ION.html`.

SHIRAI, Y. AND INOUE, H., 1973. Guiding a robot by visual feedback in assembling tasks. *Pattern Recognition*, 5, 2 (Jun. 1973), 99–108.

SONY, 2013. Sony Bloggie. `http://www.sony.co.uk/hub/mobile-hd-snap-camera/bloggie-features/article/360-video-shooting`.

STACEY, G. AND MAHONY, R., 2013. A port-Hamiltonian approach to formation control using bearing measurements and range observers. In *52nd IEEE Conference on Decision and Control (Abstract)*.

STACEY, G.; MAHONY, R.; AND CORKE, P. I., 2013. A bondgraph approach to formation control using relative state measurements. *European Control Conference*, July (2013), 17–19.

THINGM, 2013. BlinkM. `http://thingm.com/products/blinkm`.

VIA EMBEDDED, 2013. VIA EPIA-P910. `http://www.viaembedded.com/en/products/boards/1950/1/EPIA-P910.html`.

Vicon Motion Systems Ltd, 2013. Vicon. http://www.vicon.com/.

Wang, P., 1991. Navigation strategies for multiple autonomous mobile robots moving in formation. *Journal of Robotic Systems*, 8, 2 (1991), 177–195.

Willow Garage, 2013. Robot Operating System. http://wiki.ros.org/.

Wilson, W.; Williams Hulls, C.; and Bell, G., 1996. Relative end-effector control using Cartesian position based visual servoing. *IEEE Transactions on Robotics and Automation*, 12, 5 (1996), 684–696.

Xubuntu, 2013. Xubuntu. http://xubuntu.org/.

Flight Test Data

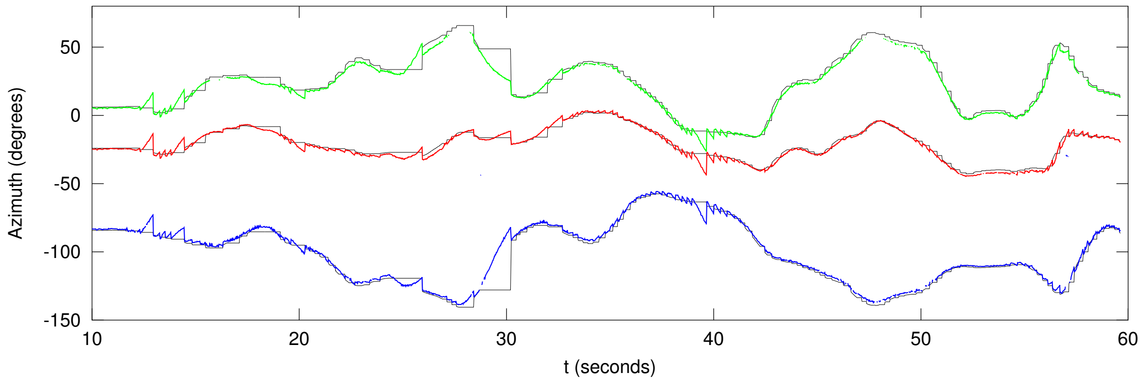


Figure A.1: Bearing azimuth measurements (Vicon reference black).

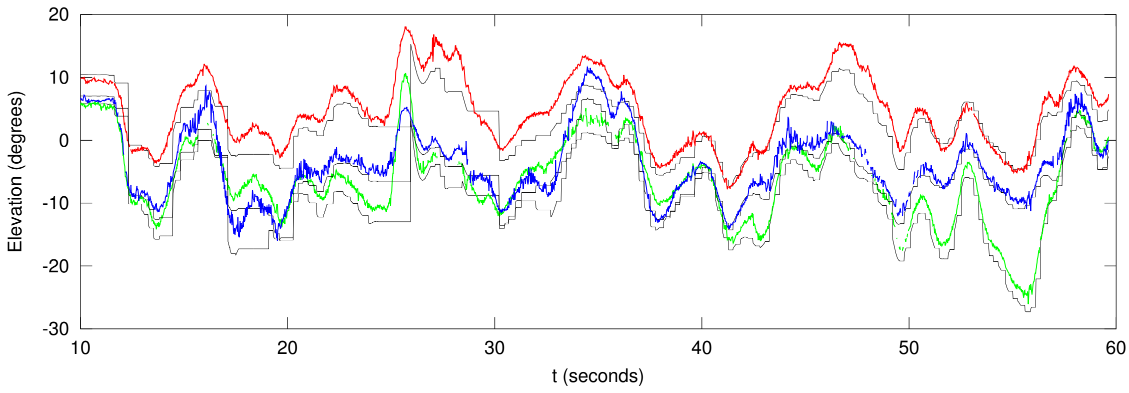


Figure A.2: Bearing elevation measurements (Vicon reference black).

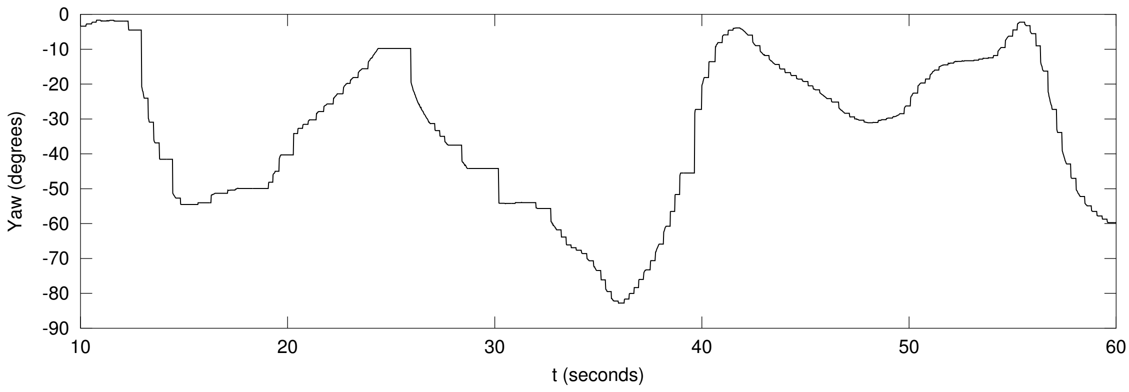


Figure A.3: Vicon yaw measurements.

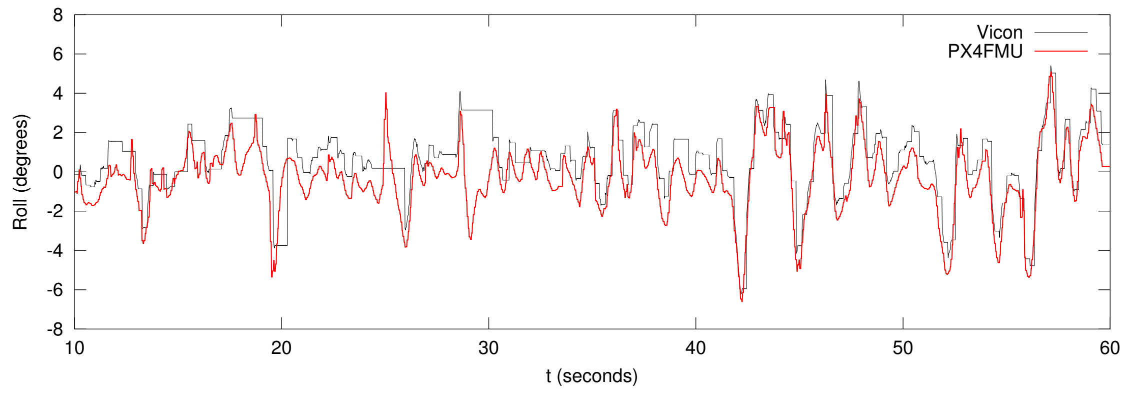


Figure A.4: PX4FMU and Vicon roll measurements.

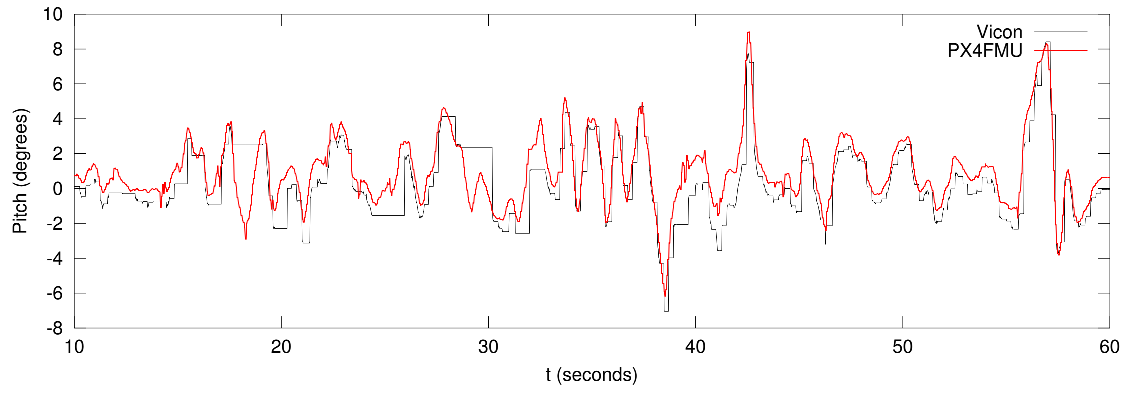


Figure A.5: PX4FMU and Vicon pitch measurements.

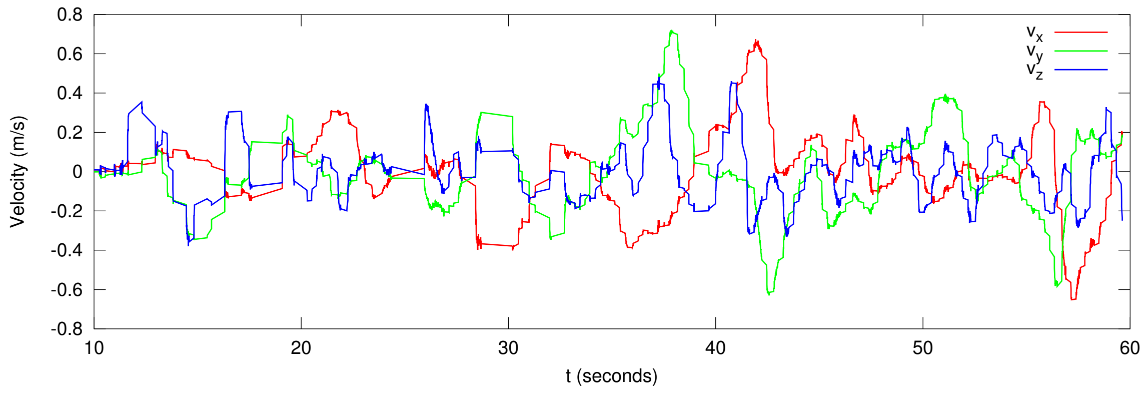


Figure A.6: Velocity measurements from Vicon

Appendix B

ROS Code

B.1 mv_bluefox_driver Node Modifications

This section outlines the modifications to the 'mv_bluefox_driver' package by Carlos Jaramillo which exposed control of gain, exposure time, pixel clock and region of interest.

Listing B.1: Modifications to /mv_bluefox_driver/include/camera.h.

```
class Camera {
<unmodified intermediate code not displayed for brevity>
private:
  int startX_, startY_,gain_dB_, expose_us_, pixelClock_KHz_;
<unmodified intermediate code not displayed for brevity>
```

Listing B.2: Modifications to /mv_bluefox_driver/src/camera.cpp.

```
Camera::Camera(ros::NodeHandle _comm_nh, ros::NodeHandle _param_nh) :
  node(_comm_nh), pnode(_param_nh) {
<unmodified intermediate code not displayed for brevity>
  pnode.param("startX", startX_, 0);
  pnode.param("startY", startY_, 0);
  pnode.param("gain_dB", gain_dB_, 0);
  pnode.param("expose_us", expose_us_, 20000);
  pnode.param("pixelClock_KHz", pixelClock_KHz_, 27000);
<unmodified intermediate code not displayed for brevity>

bool Camera::initSingleMVDevice() {
<unmodified intermediate code not displayed for brevity>
  settings.cameraSetting.gain_dB.write(gain_dB_);
  settings.cameraSetting.expose_us.write(expose_us_);
  settings.cameraSetting.pixelClock_KHz.write(mvIMPACT::acquire::TCameraPixelClock(
      pixelClock_KHz_));
  settings.cameraSetting.aoiStartX.write( startX_ );
  settings.cameraSetting.aoiStartY.write( startY_ );
<unmodified intermediate code not displayed for brevity>
```

B.2 marker_detector Node

This section covers the source code and custom ROS message types for the marker_detector node developed for detection of coloured markers.

Listing B.3: /formation_control/src/markerDetector.cpp.

```
/* This node detects vehicle markers and outputs their bearings */
/* General includes + ROS messages */
#include <ros/ros.h>
#include <math.h>
#include <limits>
#include <sstream>
#include "Eigen/Core" // Eigen vector c++ library
#include "formation_control/bearing.h"
/* Image retrieval and image processing */
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
/* Camera calibration */
#include "undistortFunctions/ocam_functions.cpp" // OCamCalib
/* Parameters */
#define calibLoc "/home/lachy/catkin_ws/src/formation_control/cfg/calib_results.txt"
#define output_ false
#define display_ false
#define publish_ true
#define minBounding_ 30
#define minDist_ 96
#define maxDist_ 230

namespace enc = sensor_msgs::image_encodings;
using namespace cv;

class ImageConverter {
```

```
public:
    /* ROS node + subscribers/publishers */
  ros::NodeHandle nh_;
  image_transport::ImageTransport it_;
  image_transport::Subscriber image_sub_;
  image_transport::Publisher image_pub_;
    ros::Publisher pub_bearings;
    formation_control::bearing msg;
    /* Marker detection variables and projection model parameters */
    XmlRpc::XmlRpcValue markers_;
    Eigen::Matrix<int, 6, Eigen::Dynamic> mrk_col;
    Eigen::Matrix<double, 1, Eigen::Dynamic> x,y, sz;
    int numMarkers;
    struct ocam_model cam;
    Mat ROImask;
public:
    ~ImageConverter() {}
  ImageConverter() : it_(nh_) {
        /* ROS node + subscribers/publishers initialisation */
    if (publish_) image_pub_ = it_.advertise("out", 1);
    image_sub_ = it_.subscribe("/mv_bluefox_camera_node/image_raw", 1, &ImageConverter
        ::imageCb, this); // Find names with rostopic list
        pub_bearings = nh_.advertise<formation_control::bearing>("bearings", 1000);
    nh_.param("markers_thresholds", markers_, markers_);
        /* Marker detection variables */
        get_ocam_model(&cam, calibLoc);
        numMarkers=markers_.size();
        mrk_col.resize(6,numMarkers);
        msg.bearing.resize(numMarkers);
        x.resize(numMarkers);
        y.resize(numMarkers);
        sz.resize(numMarkers);
        for(int i=0; i < numMarkers; i++) {
            x(i)=NAN;
            y(i)=NAN; }
        /* Evaluate marker thresholds from config.yaml */
    for(int i=0; i < numMarkers; i++) {
            std::stringstream stream(markers_[i]);
            int j=0;
            int n;
            while(stream >> n) {
                mrk_col(j,i)=n;
                j+=1; } }
        /* Define a mask for only searching on the the mirror */
        ROImask.create(480,480, CV_8U); // Set to image res
        ROImask.setTo(Scalar(0,0,0));
        circle(ROImask,Point2f(cam.yc, cam.xc),maxDist_,Scalar(255),−1,8,0);
        circle(ROImask,Point2f(cam.yc, cam.xc),minDist_,Scalar(0),−1,8,0); }

    /* Image callback */
  void imageCb(const sensor_msgs::ImageConstPtr& msg) {
    cv_bridge::CvImagePtr cv_ptr;
    try { cv_ptr = cv_bridge::toCvCopy(msg); }
    catch (cv_bridge::Exception& e) {
      ROS_ERROR("cv_bridge exception: %s", e.what());
      return; }
        markerDetector(cv_ptr); }

    /* Detects markers, publishes bearings */
    void markerDetector(cv_bridge::CvImagePtr cv_ptr) {
        Mat imgHSV, imgMask;
        cv_ptr−>image.copyTo(imgMask,ROImask); // Mask all but the mirror
    cvtColor(imgMask, imgHSV, CV_RGB2HSV); //Convert RGB to HSV
        for (int i=0; i<numMarkers; i++) {
            msg.bearing[i].x=NAN;
            msg.bearing[i].y=NAN;
            msg.bearing[i].z=NAN;
            /* Update ROI */
            Mat imgROI;
            int x_off,y_off, x_sz, y_sz;
            if (isnan(x(i))) {
                x_off=0;
                y_off=0;
                x_sz=imgHSV.cols;
                y_sz=imgHSV.rows;
                imgROI=imgHSV;
            } else {
                x_off=std::min(imgHSV.cols−2,(std::max(0,(int)(x(i)−sz(i)/2))));
                y_off=std::min(imgHSV.rows−2,(std::max(0,(int)(y(i)−sz(i)/2))));
                x_sz=std::min((int)sz(i),imgHSV.cols−x_off−1);
                y_sz=std::min((int)sz(i),imgHSV.rows−y_off−1);
                imgROI=imgHSV(Rect(x_off,y_off,x_sz,y_sz)); }
            /* Threshold images based on colour in HSV colour space */
            Mat binaryThreshold(imgROI.size(), CV_8U);
            if (mrk_col(0,i)>mrk_col(1,i)) { // Special case for colours that wrap around 0 hue (red)
                Mat binaryThreshold2(imgROI.size(), CV_8U);
                inRange(imgROI, Scalar(0,mrk_col(1,i),mrk_col(4,i)), Scalar(mrk_col(1,i), mrk_col(3,i),mrk_col(5,i)), binaryThreshold);
                inRange(imgROI, Scalar(mrk_col(0,i),mrk_col(2,i),mrk_col(4,i)), Scalar(255, mrk_col(3,i),mrk_col(5,i)), binaryThreshold2);
                binaryThreshold|=binaryThreshold2;
            } else {
                inRange(imgROI, Scalar(mrk_col(0,i),mrk_col(2,i),mrk_col(4,i)), Scalar( mrk_col(1,i),mrk_col(3,i),mrk_col(5,i)), binaryThreshold); }
            /* Find largest blob in image and compute centroid */
            vector<vector<Point> > contours;
            vector<Vec4i> hierarchy;
            findContours( binaryThreshold, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0));
            if (contours.size()==0) { x(i)=NAN; continue; }
            double maxArea=0;
            int maxC=0;
            for( int c = 0; c < contours.size(); c++ ) {
                double Area=contourArea(contours[c]);
                if (Area>maxArea) {
                    maxArea=Area;
                    maxC=c; } }
            if (maxArea==0) { x(i)=NAN; continue; }
            Moments mu = moments(contours[maxC], false);
            x(i)=mu.m10/mu.m00+x_off;
            y(i)=mu.m01/mu.m00+y_off;
            /* Compute area for next ROI based on size of marker */
            Point2f center;
            float radius;
            minEnclosingCircle(contours[maxC],center,radius);
            sz[i]=radius+minBounding_;
            /* Compute bearing */
            double point2D[2] = {y(i),x(i)};
            double point3D[3];
            cam2world(point3D, point2D, &cam);
            msg.bearing[i].x=point3D[0];
            msg.bearing[i].y=point3D[1];
```

```
            msg.bearing[i].z=point3D[2];
            /* Draw contours, centroids and labels of detected markers */
            if(publish_) {
                char buffer[3];
                sprintf(buffer,"%d",i);
                circle(cv_ptr->image, Point2f(x(i), y(i)), 2, Scalar( 0, 0, 0 ), -1, 8, 0 );
                rectangle(cv_ptr->image, Point2f(x_off,y_off),Point2f(x_off+x_sz,y_off+y_sz),Scalar( 255, 0, 0 ));
                putText(cv_ptr->image,buffer, Point2f( x(i), y(i)), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(0,255,0),1.5); }
            if(output_) ROS_INFO("Marker %d, IMG (%.1f,%.1f) -> CAM (%.3f,%.3f,%.3f), %.0f", i, x(i), y(i), msg.bearing[i].x, msg.bearing[i].y, msg.bearing[i].z, acos(msg.bearing[i].z)*180/3.141-90);
            circle(imgHSV,Point2f(x(i), y(i)),20,Scalar(0),-1,8,0); } // Mask region after something is detected there (not ideal)
        if(display_) {
            Mat imgRGB;
            cvtColor(cv_ptr->image, imgRGB, CV_RGB2BGR);
            imshow("Detected image", imgRGB);
            waitKey(1); }
        if (publish_) {
            //circle( cv_ptr->image, Point2f(cam.yc, cam.xc), minDist_, Scalar( 0, 0, 255 ), 1, 8, 0 );
            circle( cv_ptr->image, Point2f(cam.yc, cam.xc), maxDist_, Scalar( 0, 0, 255 ), 1, 8, 0 );
            image_pub_.publish(cv_ptr->toImageMsg()); }
        /* Publish bearings */
        pub_bearings.publish(msg); }
};

int main(int argc, char** argv) {
  ros::init(argc, argv, "marker_detector");
  ImageConverter ic;
  ros::spin();
  return 0;
}
```

Listing B.4: /formation_control/msg/bearing.msg.

```
# Contains an array of all of the detected bearings of markers
geometry_msgs/Vector3[] bearing
```

Listing B.5: /formation_control/cfg/config.yaml.

```
# Defines thresholds for markers in the HSV colour space
# Order is Hmin Hmax Smin Smax Vmin Vmax
markers_thresholds:
- 107 125 100 220 160 256
- 70 93 70 220 100 256
- 5 19 150 230 175 256
# Goal bearings (on unit sphere)
marker_goals:
- 1.588 -0.613 0.700
- 1.239 0.392 0.430
- -0.075 -1.216 0.559
# Location of fixed markers positions
use_fixed_markers: true
marker_fixed_pos:
- 1.588 -0.613 0.650
- 1.189 0.342 0.430
- -0.075 -1.196 0.459
```

B.3 formation_control Node

This section covers the source code and custom ROS message types for the formation_control node developed for applying the formation control algorithm. This node also uses *config.yaml* as was just listed.

Listing B.6: /formation_control/src/formationControl.src.

```
/* This node implements the formation control algorithm outputting a desired control force
 * This node takes in sensor measurements from the camera, Vicon and the IMU.
 * config.yaml defines many options such as vehicle marker colours and desired bearings. */

/* General includes + ROS messages + MAVLINK messages */
#include "ros/ros.h"
#include <fstream>
#include <iostream>
#include <math.h>
#include <limits>
#include "Eigen/Core" // Eigen vector c++ library
#include "formation_control/bearing.h"
#include "formation_control/state.h"
```

```
#include "geometry_msgs/Vector3.h"
#include "std_msgs/String.h"
//#include "mavlink.h"
//#include "mavlink_ros/Mavlink.h"
#include <std_msgs/UInt16MultiArray.h> // Output to mavlink_ros
/* Vicon and PX4 measurements/transforms */
#include "geometry_msgs/TransformStamped.h"
#include <tf/transform_datatypes.h>
#include <tf/transform_listener.h>
#include "sensor_msgs/Imu.h"
/* Display goal bearings and current bearings */
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
/* Standard includes */
#include <iostream>
#include <cstdlib>
#include <unistd.h>
#include <cmath>
#include <string.h>
#include <inttypes.h>
#include <fstream>
/* Latency Benchmarking */
#include <sys/time.h>
#include <time.h>
/* Parameters and numerical constants */
#define pi 3.1415926
#define g 9.81
#define viconTopic "vicon/PX4_FORMATION/PX4_FORMATION"
#define imuTopic "/fcu/imu"
#define bearingTopic "bearings"
#define modeTopic "/mode"
#define INT16_MAX 0x7fff
#define INT16_MIN (-INT16_MAX - 1)
#define UINT16_MAX 0xffff
#define vehicleMass 1.3
#define goalHeight 1
#define hoverThrust 0.2
#define Kp 0.0
```

```
#define Ki 0.0
#define Kd 0.0
#define c 3.0
#define c_obs_k 1.0
#define default_range 1.5
#define vel_obs_G 0.25
#define D 3
#define centreX 236.917082
#define centreY 238.194657
#define debug false
#define debugIMU false
#define debugVicon false
#define attHybrid true

// Logging - Saves to ~/.ros
#define logVicon false
#define logViconLoc "logVicon.txt"
#define logRPY false
#define logRPYLoc "logRPY.txt"
#define logBearing false
#define logBearingLoc "logBearing.txt"

#define SYSID 1
#define COMPID 0
#define MAVLINK_OFFBOARD_CONTROL_MODE_NONE 0
#define MAVLINK_OFFBOARD_CONTROL_MODE_RATES 1
#define MAVLINK_OFFBOARD_CONTROL_MODE_ATTITUDE 2
#define MAVLINK_OFFBOARD_CONTROL_MODE_VELOCITY 3
#define MAVLINK_OFFBOARD_CONTROL_MODE_POSITION 4
#define MAVLINK_OFFBOARD_CONTROL_FLAG_ARMED 0x10
#define MAV_MODE_PREFLIGHT 0
#define MAV_MODE_STABILIZE_ARMED 208
#define MAV_MODE_MANUAL_ARMED 192

namespace enc = sensor_msgs::image_encodings;
using namespace std;

geometry_msgs::Vector3 eigenToVector3(Eigen::Matrix<double, 3, 1> eig) {
    geometry_msgs::Vector3 vec;
    vec.x=eig(0);
    vec.y=eig(1);
    vec.z=eig(2);
    return vec;
}

Eigen::Matrix<double, 3, 1> Vector3ToEigen(geometry_msgs::Vector3 vec) {
    Eigen::Matrix<double, 3, 1> eig;
    eig(0)=vec.x;
    eig(1)=vec.y;
    eig(2)=vec.z;
    return eig;
}

class formationControl {
public:
    /* ROS node + subscribers/publishers + messages */
    ros::NodeHandle nh_;
  ros::Subscriber subBearing, subIMU, subVicon, subMode;
    ros::Publisher pub_force, pub_stateVicon, pub_stateIMU, pub_mavlinkMSG,
        pub_mavlinkMSGv2;
    geometry_msgs::Vector3 msgForce;
    formation_control::state msgViconState, msgIMUState;
    /* Formation control variables */
    double dt; // Time between bearing measurements
    int N; // Number of quads
    XmlRpc::XmlRpcValue mrk_goal_,marker_fixed_pos_;
  bool use_fixed_markers_;
    Eigen::Matrix<double, 3, Eigen::Dynamic> e, s, sTilde, sGoal, sPrev, sTildeDot, epsilon
        , sGoal_img, sCam, s_img, p_fixed;
    Eigen::Matrix<double, 1, Eigen::Dynamic> r, rHat, rDot;
    Eigen::Matrix<double, 3, 1> vel_obs;
    Eigen::Matrix<double, 3, 1> epsilon_total, epsilon_total_bff, epsilon_total_img;
    uint16_t rset,pset,yset,tset; // The implemented set points
    /* State information (velocity, roll/pitch/yaw), rotation matrices */
    string mode_;
    Eigen::Matrix<double, 3, 3> R_vicon2bff, R_IMU2bff, R_bff2cam, R_cam2img;
    tf::Matrix3x3 R_v;
    Eigen::Matrix<double, 3, 1> p, pDot;
    bool skipNext;
    int frame_number,frame_numberPrev;
    Eigen::Matrix<double, 1, 2> ePID;
    double ePIDDot;
    double ePIDInt;
    /* Visual debugging */
    image_transport::ImageTransport it_;
    image_transport::Subscriber image_sub_;
    /* Safety/logging/performance checking */
    ros::Time viconStamp, px4Stamp, bearingStamp;
    ros::Duration blindTime; // If PX4 measurements or Vicon measurements not received in this time, power down
    ofstream logFileVicon,logFileRPY,logFileBearing;
    int fpsCount;
    ros::Time fpsTime;
    int counter;
    ros::Timer timer;

public:
    formationControl() : it_(nh_) {
        /* ROS node + subscribers/publishers initialisation + others */
        counter=0;
        mode_ = "disarmed";
        subMode = nh_.subscribe(modeTopic, 1000, &formationControl::modeCallback,this);
        subBearing = nh_.subscribe(bearingTopic, 1000, &formationControl::bearingCallback,this);
        subIMU = nh_.subscribe(imuTopic, 1000, &formationControl::IMUCallback,this);
        subVicon=nh_.subscribe(viconTopic,1000, &formationControl::ViconCallback,this);
        //pub_mavlinkMSG = nh_.advertise<mavlink_ros::Mavlink>("/mavlink/to", 1000);
        pub_mavlinkMSGv2 = nh_.advertise<std_msgs::UInt16MultiArray>("/mavlink/offboard",1000);
        pub_force = nh_.advertise<geometry_msgs::Vector3>("force_xyz", 1000);
        image_sub_ = it_.subscribe("/out", 1, &formationControl::imageCb, this);
        pub_stateVicon = nh_.advertise<formation_control::state>("state_vicon", 1000);
        pub_stateIMU = nh_.advertise<formation_control::state>("state_IMU", 1000);
        timer = nh_.createTimer(ros::Duration(0.02), &formationControl::calculateForce, this); // Send control commands at 50Hz
        if (logVicon) {
            logFileVicon.open(logViconLoc);
            logFileVicon << "#time" << "\t" << "vx" << "\t" << "vx_raw" << "\t" << "vy" << "\t" << "vy_raw" << "\t" << "vz" << "\t" << "vz_raw" "\t" << "x" << "\t" << "y" << "\t" << "z" << endl;
        }
        if (logRPY) {
            logFileRPY.open(logRPYLoc);
            logFileRPY << "#time" << "\t" << "vR" << "\t" << "vP" << "\t" <<
                "vY" << "\t" << "pR" << "\t" << "pP" << "\t" << "pY" <<
                endl;
        }
        if (logBearing) logFileBearing.open(logBearingLoc);
        blindTime=ros::Duration(1.0);
        fpsTime=ros::Time::now();
        /* Formation control variables */
        nh_.param("marker_goals", mrk_goal_, mrk_goal_);
    nh_.param("marker_fixed_pos", marker_fixed_pos_, marker_fixed_pos_);
    nh_.param("use_fixed_markers", use_fixed_markers_, false);
        N=mrk_goal_.size(); // Number of quads
        dt=1;
    p_fixed.resize(3,N);
        e.resize(3,N);
        s.resize(3,N);
        sCam.resize(3,N);
        s_img.resize(3,N);
        sGoal_img.resize(3,N);
        sGoal.resize(3,N);
        sTilde.resize(3,N);
        sPrev.resize(3,N);
        sPrev=Eigen::MatrixXf::Constant(3,N,NAN).cast<double>(); // Eigen is
            AWQUAD
        sTildeDot.resize(3,N);
        r.resize(1,N);
        rHat.resize(1,N);
        rDot.resize(1,N);
        epsilon.resize(3,N);
        /* Evaluate goal bearings from config.yaml */
        for (int i=0; i < N; i++) {
            std::stringstream stream(mrk_goal_[i]);
            int j=0;
            double n;
            while(stream >> n) {
                sGoal(j,i)=n;
                j+=1; } }
        /* Evaluate locations of fixed markers from config.yaml */
        if (use_fixed_markers_) {
            for (int i=0; i < N; i++) {
```

```cpp
                std::stringstream stream(marker_fixed_pos_[i]);
                int j=0;
                double n;
                while(stream >> n) {
                    p_fixed(j,i)=n;
                    j+=1; } } }
        /* Define fixed rotation matrices */
        R_bff2cam << 0, -1, 0, 1, 0, 0, 0, 0, 1;
        R_cam2img << 0, 1, 0, 1, 0, 0, 0, 0, -1;

    }
    ~formationControl() {}

    void modeCallback(std_msgs::String msg) {
        mode_=msg.data;
    }

    /* Gets rotation matrix/rpy from IMU */
    void IMUCallback(sensor_msgs::Imu msg_imu) {
        double rollIMU, pitchIMU, yawIMU;
        tf::Quaternion qIMU(msg_imu.orientation.x, -msg_imu.orientation.y, -msg_imu.orientation.z, msg_imu.orientation.w);
        tf::Matrix3x3 R_IMU(qIMU);
        R_IMU.getEulerYPR(yawIMU, pitchIMU, rollIMU);
        R_IMU2bff << R_IMU[0][0], R_IMU[1][0], R_IMU[2][0], R_IMU[0][1], R_IMU[1][1], R_IMU[2][1], R_IMU[0][2], R_IMU[1][2], R_IMU[2][2];
        if (attHybrid) R_v.setEulerYPR(msgViconState.yaw*pi/180.0,-pitchIMU,-rollIMU);
        else R_v.setEulerYPR(msgViconState.yaw*pi/180.0,msgViconState.pitch*pi/180.0,msgViconState.roll*pi/180.0);
        R_vicon2bff << R_v[0][0], R_v[1][0], R_v[2][0], R_v[0][1], R_v[1][1], R_v[2][1], R_v[0][2], R_v[1][2], R_v[2][2];
        msgIMUState.roll=rollIMU*180.0/pi;
        msgIMUState.pitch=pitchIMU*180.0/pi;
        msgIMUState.yaw=yawIMU*180.0/pi;
        pub_stateIMU.publish(msgIMUState);
        if (debugIMU) ROS_INFO("IMU r,p,y=%2.2f,%2.2f,%2.2f",rollIMU*180/pi, pitchIMU*180/pi, yawIMU*180/pi);
    px4Stamp=ros::Time::now();
    }

    /* Gets rotation matrix/rpy and position/velocity from Vicon */
    void ViconCallback(const geometry_msgs::TransformStamped::ConstPtr& msg) {
        ros::Time viconStampPrev=viconStamp;
        viconStamp=ros::Time::now();
        /* Compute position and rotation matrix */
        Eigen::Matrix<double, 3, 1> pPrev=p;
        Eigen::Matrix<double, 3, 1> rpy;
        p << -msg->transform.translation.y, -msg->transform.translation.x, msg->transform.translation.z+0.15; // +~0.2 for cam origin
        tf::Quaternion q(-msg->transform.rotation.y, -msg->transform.rotation.x, -msg->transform.rotation.z, msg->transform.rotation.w);
        double roll, pitch, yaw;
        R_v.setRotation(q);
        R_v.getEulerYPR(yaw, pitch, roll);
        rpy << roll, pitch, yaw;
        if (attHybrid) R_v.setEulerYPR(yaw,-msgIMUState.pitch*pi/180.0,-msgIMUState.roll*pi/180.0);
        R_vicon2bff << R_v[0][0], R_v[1][0], R_v[2][0], R_v[0][1], R_v[1][1], R_v[2][1], R_v[0][2], R_v[1][2], R_v[2][2];
        /* Compute the change in time between received measurements */
        int frameDiff=msg->header.stamp.toSec() - frame_numberPrev;
        double dtt=0.005*frameDiff; // 200Hz, due to latency etc the timestamps cannot be used
        frame_numberPrev=msg->header.stamp.toSec();
        if (dtt==(0.005*frame_numberPrev) || dtt<0) return; // Startup condition
        Eigen::Matrix<double, 3, 1> pDot_raw=(p-pPrev)/dtt;
        pDot+=vel_obs_G*(pDot_raw-pDot);
        msgViconState.roll=rpy(0)*180/pi;
        msgViconState.pitch=rpy(1)*180/pi;
        msgViconState.yaw=rpy(2)*180/pi;
        msgViconState.pos=eigenToVector3(p);
        msgViconState.vel=eigenToVector3(pDot);
        msgViconState.vel_raw=eigenToVector3(pDot_raw);
        pub_stateVicon.publish(msgViconState);
        if (logVicon) logFileVicon << ros::Time::now() << "\t" << frame_numberPrev << "\t" << pDot(0) << "\t" << pDot_raw(0) << "\t" << pDot(1) << "\t" << pDot_raw(1) << "\t" << pDot(2) << "\t" << pDot_raw(2) << "\t" << p(0) << "\t" << p(1) << "\t" << p(2) << endl;
        if (debugVicon) {
            ROS_INFO("VICON Roll:%.2f\tPitch:%.2f\tYaw:%.2f",rpy(0)*180/pi,rpy(1)*180/pi,rpy(2)*180/pi);
            ROS_INFO("VICON x:%.2f\ty:%.2f\tz:%.2f",p(0),p(1),p(2));
            ROS_INFO("VICON vx:%.2f\tvy:%.2f\tvz:%.2f",pDot(0),pDot(1),pDot(2));
        }
    }

    /* Converts bearing measurements into a force */
    void bearingCallback(const formation_control::bearing &s_xyz) {
        dt=(ros::Time::now()-bearingStamp).toSec();
        bearingStamp=ros::Time::now();
        if (fpsCount++%100==0) {
            ROS_INFO("Marker detection at %2fhz",100/(ros::Time::now()-fpsTime).toSec());
            fpsTime=ros::Time::now();
        }
        /* Update range velocity measurements */
        Eigen::Matrix<double, 3, 1> p = Vector3ToEigen(msgViconState.pos);
    for (int k=0; k<N; k++) {
            double rPrev=r(k);
            r(k)=(p-p_fixed.col(k)).transpose()*(p-p_fixed.col(k));
            rDot(k)=(r(k)-rPrev)/dt;
        }
        /* Algorithm Implementation (old algorithm, but damping not applied) */
        for (int i=0; i<N; i++) sCam.col(i) = Vector3ToEigen(s_xyz.bearing[i]).transpose();
        sPrev=s;
        s=R_vicon2bff.transpose()*R_bff2cam.transpose()*sCam;
        sTildeDot=(s-sPrev)/dt;
        sTilde=s-sGoal;
        e=c*sTilde;
        for (int k=0; k<N; k++) {
            /* Handle missing markers */
            if (isnan(sTildeDot(0,k))) {
                e.col(k)=Eigen::MatrixXf::Zero(3,1).cast<double>();
                s.col(k)=Eigen::MatrixXf::Zero(3,1).cast<double>();
                sTilde.col(k)=Eigen::MatrixXf::Zero(3,1).cast<double>();
                sTildeDot.col(k)=Eigen::MatrixXf::Zero(3,1).cast<double>();
                rHat(k)=default_range; }
            /* Update range estimate (keep it positive just in case) */
            rHat(k)+=(rDot(k)-((e.col(k).dot(sTildeDot.col(k)))/(c_obs_k*rHat(k))))*dt;
            if (rHat(k)<0.1) rHat(k)=0.1;
            /* Get control force from vehicle link */
            epsilon.col(k)=(1/rHat(k))*(Eigen::MatrixXf::Identity(3,3).cast<double>()-s.
                col(k)*s.col(k).transpose())*e.col(k);
        }
        /* Sum forces, convert into bff, publish and then implement the force */
        epsilon_total=epsilon.rowwise().sum();
        Eigen::Matrix<double, 3, 1> pDot = Vector3ToEigen(msgViconState.vel); //
            Damping not tested
        epsilon_total_bff=R_vicon2bff*epsilon_total-R_vicon2bff*D*pDot;
        msgForce=eigenToVector3(epsilon_total_bff);
        pub_force.publish(msgForce);
        if(debug) ROS_INFO("Output control force (BFF) = (%.3f,%.3f,%.3f)", msgForce.
            x, msgForce.y, msgForce.z);
        /* Log Azimuth/Bearing data */
        if(logBearing) {
            logFileBearing << ros::Time::now() << "\t";
            for (int i=0; i<N; i++) {
                Eigen::Matrix<double, 3, 1> qi = p_fixed.col(i)-p;
                Eigen::Matrix<double, 3, 1> si = qi/sqrt(qi.transpose()*qi);
                float viconAzimuth = atan2(si(1),si(0))*180/pi;
                float viconElevation = asin(si(2))*180/pi;
                float bearingAzimuth = atan2(s(1,i),s(0,i))*180/pi;
                float bearingElevation = asin(s(2,i))*180/pi;
                if(isnan(s_xyz.bearing[i].x)) logFileBearing << viconAzimuth << "\t" <<
                    "NaN" << "\t" << viconElevation << "\t" << "NaN" << "\t";
                else logFileBearing << viconAzimuth << "\t" << bearingAzimuth << "\
                    t" << viconElevation << "\t" << bearingElevation << "\t";
            }
            logFileBearing << endl;
        }
    }

    /* Calculate force */
    void calculateForce(const ros::TimerEvent&) {
        //https://pixhawk.ethz.ch/mavlink/#
            SET_QUAD_SWARM_ROLL_PITCH_YAW_THRUST
        rset=INT16_MAX*atan(msgForce.x/(vehicleMass*g))/pi; //+-
        pset=INT16_MAX*-atan(msgForce.y/(vehicleMass*g))/pi; //+-
        /* Limit roll and pitch angles to 10 degrees */
        if (((float)rset/INT16_MAX*180)>10.0) rset=INT16_MAX*10/180;
        if (((float)rset/INT16_MAX*180)<-10.0) rset=-INT16_MAX*10/180;
        if (((float)pset/INT16_MAX*180)>10.0) pset=INT16_MAX*10/180;
        if (((float)pset/INT16_MAX*180)<-10.0) pset=-INT16_MAX*10/180;
        /* Just keep a steady yaw (unreliable) */
        //yset=INT16_MAX * 0.00f;
        /* Yaw controller (needs checking) */
        yset=INT16_MAX*((msgIMUState.yaw+0.1*msgViconState.yaw)/180);
        /* Height controller (needs checking) */
        if (mode_=="height") {
            rset=INT16_MAX*0.0;
            pset=INT16_MAX*0.0;
        }
    ePID(0)=ePID(1);
    ePID(1)=goalHeight-msgViconState.pos.z;
    //ePIDInt=ePIDInt+((ePID(0)+ePID(1))/2)*dt;
    ePIDDot=(ePID(1)-ePID(0))/dt;
        double thrust=Kp*ePID(1)+Kd*ePIDDot+hoverThrust;//+Ki*ePIDInt;
        if (thrust<0) { tset = 0; }
        else if (thrust>1) { tset = UINT16_MAX; }
        else { tset=UINT16_MAX*thrust; }
        /* Safety check */
    ros::Time now = ros::Time::now();
        if ((now-viconStamp)>ros::Duration(2.0) || (now-px4Stamp)>ros::Duration(2.0))
            {
            tset=UINT16_MAX * 0.0f;
            if (debug) ROS_WARN("No vicon or px4 messages for 2.0s, thrust zero"); }
    else if ((now-viconStamp)>ros::Duration(0.5) || (now-px4Stamp)>ros::Duration(0.5))
        {
            tset=UINT16_MAX * 0.2f;
            if (debug) ROS_WARN("No vicon or px4 messages for 0.5s, thrusting down");
                }
        /* Implement the force */
        implementForce();
    }

    /* Implement the control force, with independent height controller */
    void implementForce() {
        /* Send command to customised mavlink_ros node */
        std_msgs::UInt16MultiArray msg;
        msg.data.resize(4);
        msg.data[0]=rset;
        msg.data[1]=pset;
        msg.data[2]=yset;
        if (mode_=="formation") msg.data[3]=tset;
        else msg.data[3]=UINT16_MAX*0.01f;
        if (mode_!="stop") pub_mavlinkMSGv2.publish(msg);
        /* Log rpy/setpoint */
        if (logRPY) logFileRPY << ros::Time::now() << "\t" << msgViconState.roll <<
             "\t" << msgViconState.pitch << "\t" << msgViconState.yaw << "\t" <<
             msgIMUState.roll << "\t" << msgIMUState.pitch << "\t" <<
            msgIMUState.yaw << endl;
        /* Mavlink packet just seems incorrect, similar code seems to work elsewhere? who
            knows */
        /*
        mavlink_message_t msg_control;
        mavlink_set_quad_swarm_roll_pitch_yaw_thrust_t sp;
        sp.group = 0;
        sp.mode = MAVLINK_OFFBOARD_CONTROL_MODE_ATTITUDE;
        sp.roll[0] = INT16_MAX * 0.00f;
        sp.pitch[0] = INT16_MAX * 0.00f;
        sp.yaw[0] = INT16_MAX * 0.00f;
        sp.thrust[0] = UINT16_MAX*0.01f;
        mavlink_msg_set_quad_swarm_roll_pitch_yaw_thrust_encode(200, 0, &msg_control, &
            sp); // This line fails to give correct MAVLINK message
        mavlink_ros::Mavlink rosmavlink_msg;
    rosmavlink_msg.len = msg_control.len;
    rosmavlink_msg.seq = msg_control.seq;
    rosmavlink_msg.sysid = msg_control.sysid;
    rosmavlink_msg.compid = msg_control.compid;
    rosmavlink_msg.msgid = msg_control.msgid;
        for (int i = 0; i < msg_control.len / 8; i++)
      {
        (rosmavlink_msg.payload64).push_back(msg_control.payload64[i]);
      }
        pub_mavlinkMSG.publish(rosmavlink_msg);
        if (debug) { ROS_INFO("MAVLINK RPYT SETPOINT: %d (%0.2f) %d (%0.2f) %
            d %u",sp.roll[0], (float)sp.roll[0]*180/INT16_MAX, sp.pitch[0], (float)sp.pitch
            [0]*180/INT16_MAX, sp.yaw[0], sp.thrust[0]); }
        */
    }

    /* Displays goal bearings and measured bearings for debugging */
    void imageCb(const sensor_msgs::ImageConstPtr& msg_image) {
    cv_bridge::CvImagePtr cv_ptr;
    try { cv_ptr = cv_bridge::toCvCopy(msg_image, enc::BGR8); }
    catch (cv_bridge::Exception& e) {
      ROS_ERROR("cv_bridge exception: %s", e.what());
      return; }
        /* Compute bearings in image frame and display on image */
        s_img=R_cam2img*sCam;
        sGoal_img=R_cam2img*R_bff2cam*R_vicon2bff*sGoal;
        epsilon_total_img=R_cam2img*R_bff2cam*R_vicon2bff*epsilon_total;
        int scale=100;
        char buffer[3];
        for (int k=0; k<N; k++) {
            sprintf(buffer,"%d",k);
            // Goal bearings (green)
            cv::line(cv_ptr->image, cv::Point2f(centreX,centreY), cv::Point2f(centreX+
                sGoal_img(0,k)*scale*2,centreY+sGoal_img(1,k)*scale*2),cv::Scalar
                (0,255,0));
            cv::putText(cv_ptr->image,buffer, cv::Point2f(centreX+sGoal_img(0,k)*scale,
                centreY+sGoal_img(1,k)*scale), cv::FONT_HERSHEY_SIMPLEX, 0.5, cv::
                Scalar(0,255,0),1.5);
            if (!isnan(s_img(0,k))) {
                // Measured bearings (blue)
                cv::line(cv_ptr->image, cv::Point2f(centreX,centreY), cv::Point2f(centreX
                    +s_img(0,k)*scale*2,centreY+s_img(1,k)*scale*2),cv::Scalar(255,0,0));
                cv::putText(cv_ptr->image,buffer, cv::Point2f(centreX+s_img(0,k)*scale,
                    centreY+s_img(1,k)*scale), cv::FONT_HERSHEY_SIMPLEX, 0.5, cv::
                    Scalar(255,0,0),1.5);
            }
        }
        // Control force (red)
        //cv::line(cv_ptr->image, cv::Point2f(centreX,centreY), cv::Point2f(centreX+
            epsilon_total_img[0]*scale*2,centreY+epsilon_total_img[1]*scale*2),cv::Scalar
            (0,0,255));
        //cv::putText(cv_ptr->image,"F", cv::Point2f(centreX+epsilon_total_img[0]*scale,
            centreY+epsilon_total_img[1]*scale), cv::FONT_HERSHEY_SIMPLEX, 0.5, cv::
            Scalar(0,0,255),1.5);
        cv::imshow("Detected image", cv_ptr->image);
        cv::waitKey(1);
    }
};

int main(int argc, char **argv) {
  ros::init(argc, argv, "formation_control_algorithm");
    formationControl ic;
  ros::spin();
  return 0;
}
```

Listing B.7: /formation_control/msg/quadCtrl.msg.

```
# The message published to the mavlink_ros node for external control
Header header
int16 roll
int16 pitch
int16 yaw
uint16 thrust
```

Listing B.8: /formation_control/msg/state.msg.

```
# Holds the position, attitude and velocity measurements
float64 roll
float64 pitch
float64 yaw
geometry_msgs/Vector3 pos
geometry_msgs/Vector3 vel
geometry_msgs/Vector3 vel_raw
```

B.4 mavlink_ros Node modifications

This section outlines the modifications made to the mavlink_ros_serial node by the Pixhawk team to retrieve external control commands and publish them to the PX4FMU. This modification should not have been necessary, but MAVLINK packets made in the formation_control node with the same code seemed to fail.

Listing B.9: /mavlink_ros/src/mavlink_ros_serial.src.

```
<unmodified intermediate code not displayed for brevity>
#include<std_msgs/UInt16MultiArray.h>
<unmodified intermediate code not displayed for brevity>
  offboard_sub = mavlink_nh.subscribe("offboard", 1000, offboardCallback);
<unmodified intermediate code not displayed for brevity>
```

```
void offboardCallback(const std_msgs::UInt16MultiArray msg) {
    mavlink_message_t message;
    mavlink_set_quad_swarm_roll_pitch_yaw_thrust_t sp;
    sp.group = 0;
    sp.mode =2; // MAVLINK_OFFBOARD_CONTROL_MODE_ATTITUDE;
    sp.roll[0] = msg.data[0];
    sp.pitch[0] = msg.data[1];
    sp.yaw[0] = msg.data[2];
    sp.thrust[0] = msg.data[3];
    mavlink_msg_set_quad_swarm_roll_pitch_yaw_thrust_encode(200, 0, &message, &sp);
    static uint8_t buf[MAVLINK_MAX_PACKET_LEN];
    unsigned len = mavlink_msg_to_send_buffer((uint8_t*)buf, &message);
    write(fd, buf, len);
    tcdrain(fd);
}
```

Appendix C

PX4FMU Code

Offboard control was added by appending the required states to the commander module.

Listing C.1: Modifications to commander of the PX4MU for offboard control at /src/modules/commander/commander.cpp.

```
<unmodified intermediate code not displayed for brevity>
/* Check if offboard control signal is ever found */
if (sp_offboard.timestamp!=0&&!status.offboard_control_signal_found_once) {
    status.offboard_control_signal_found_once = true;
    warnx("Found offboard signal for first time");
}

/* Update the mode switches, switch to offboard control if recent signal is obtained and
    switch for offboard control is enabled. Arm if appropriate */
check_mode_switches(&sp_man, &status);
if (status.offboard_control_signal_found_once&&status.assisted_switch==1) {
    if (hrt_absolute_time() < sp_offboard.timestamp + RC_TIMEOUT) {
        if (status.offboard_control_signal_lost) {
            control_mode.flag_control_offboard_enabled = true;
            status.navigation_state = NAVIGATION_STATE_STABILIZE;
            status.offboard_control_signal_lost = false;
            status_changed = true;
            warnx("Enabled offboard control");
            mavlink_log_critical(mavlink_fd, "[cmd] Enabled offboard control");
            transition_result_t resoffb;
            resoffb = TRANSITION_NOT_CHANGED;
            resoffb = arming_state_transition(&status, &safety, ARMING_STATE_ARMED,
                &armed);
            if (resoffb == TRANSITION_DENIED) {
                warnx("ERROR: main denied: arm %d main %d mode_sw %d", status.
                    arming_state, status.main_state, status.mode_switch);
                mavlink_log_critical(mavlink_fd, "[cmd] ERROR: main denied: arm %d main %d mode_sw %d", status.arming_state, status.main_state, status.mode_switch);
            }
            if (resoffb == TRANSITION_CHANGED) {
                    if (status.arming_state == ARMING_STATE_ARMED) {
                        warnx("[cmd] ARMED by offboard control");
                    } else {
                        warnx("[cmd] FAILED TO ARM by offboard control");
                    }
            }
        }
    } else if (control_mode.flag_control_offboard_enabled) {
        control_mode.flag_control_offboard_enabled = false;
        status.offboard_control_signal_lost = true;
        warnx("Disabled offboard control (signal lost)");
        mavlink_log_critical(mavlink_fd, "[cmd] Disabled offboard control (signal lost)");
    }
} else if (control_mode.flag_control_offboard_enabled) {
    control_mode.flag_control_offboard_enabled = false;
    status.offboard_control_signal_lost = true;
    warnx("Disabled offboard control (by switch)");
    mavlink_log_critical(mavlink_fd, "[cmd] Disabled offboard control (by switch)");
}

/* ignore RC signals if in offboard control mode */
if (!control_mode.flag_control_offboard_enabled && sp_man.timestamp != 0) {
<unmodified intermediate code not displayed for brevity>
```

Appendix D

MATLAB Simulation Code

This MATLAB code was used for simulating the formation control algorithm.

Listing D.1: MATLAB script for simulation of formation control algorithm

```
clear all, close all, clc
% Time stuff
tmax=10; % seconds
dt=1/50;% Timestep for simulation
% Constants
g=9.81;
% Algorithm Parameters
system=1;
cobs=1;
switch system
    case 0 % Old system
        c_k = 3;
        D_i=3;
    case 1 % New system
        d_k=0.2*50;
        c_k = 10*50;
        G=25*50*1.3;
        D_i=0;
    case 2 % New system, fixed range independent
        d_k=0.2;
        c_k = 10;
        G=50;
        D_i=0;
end
%% Height controller gains
Kp=0.5;
Ki=0;
Kd=3;
```

```
%% Define goals/Initial positions
numQuads=4;
mass=[1 1 1 1];
colours=['r','g','b','c'];
% FIXED MARKERS
% fixed=[0 1 1 1];
% p=[-1 0 0; -1 2 1; 2 2 1; 2 0 1]';
% pDot=[0 0 0; 0 0 0; 0 0 0; 0 0 0]';
% pGoal=[1 1 1; -1 2 1; 2 2 1; 2 0 1]';
% COMPLEX ARRANGEMENT
fixed=[0 0 0 0];
p=[-1 0 0; -1 2 1; 2 2 1; 2 0 1]';
pDot=[0 0 0; 0 0 0; 0 0 0; 0 0 0]';
pGoal=[1 1 1; 0 2 1; 2 0 1; 0 0 1]';
%% Logging to file
log = 0;
if (log==1)
    fileID = fopen('D:\???.txt','w');
    fprintf(fileID,'t p1x p1y p1z p2x p2y p2z p3x p3y p3z p4x p4y p4z\r\n');
    fileID2 = fopen('D:\???.txt','w');
    fprintf(fileID2,'t e1 e2 e3 e4\r\n');
end
%% Initialise simulation variables etc
for i=1:numQuads
    for j=1:numQuads
        qGoal(:,i,j)=pGoal(1:2,j)-pGoal(1:2,i);
        rGoal(i,j)=sqrt(qGoal(:,i,j)'*qGoal(:,i,j));
        sGoal(:,i,j)=qGoal(:,i,j)/(sqrt(qGoal(:,i,j)'*qGoal(:,i,j)));
        q(:,i,j)=p(1:2,j)-p(1:2,i);
        s(:,i,j)=q(:,i,j)/(sqrt(q(:,i,j)'*q(:,i,j)));
        r(i,j)=sqrt(q(:,i,j)'*q(:,i,j));
        e(:,i,j)=c_k*(s(:,i,j)-sGoal(:,i,j));
        rHat(i,j)=sqrt(q(:,i,j)'*q(:,i,j)); % Initial range estimates exact
        sigma(:,i,j)=zeros(2,1);
    end
    fc(:,i)=[0;0;0];
end
ePID=zeros(2,numQuads);
ePIDInt=zeros(1,numQuads);
vec_p=zeros(3,numQuads,2);
bearingError=zeros(1,numQuads);
```

```
vec_p(:,:,2)=p(:,1:numQuads);
%% Initialise plot
figure(1) % Plots goal relative positions
hold on
for i=1:numQuads
    posH_Goal(i)=scatter3(pGoal(1,i),pGoal(2,i),pGoal(3,i),50,colours(i));
    set(posH_Goal(i),'MarkerEdgeColor',colours(i),'MarkerFaceColor',colours(i))
end
axis equal
axis([-2 4 -2 4])
figure(2) % Plots vehicle position over time
set(gcf,'units','normalized','outerposition',[0.2 0.2 0.6 0.6])
subplot(3,1,2), xlabel('t (s)'), ylabel('sum((s-s_{goal})^2)')
subplot(3,1,3), xlabel('t (s)'), ylabel('height (m)')
% subplot(3,1,1), xlabel('x (m)'), ylabel('y (m)'), zlabel('z (m)')
figure(3), xlabel('x (m)'), ylabel('y (m)'), zlabel('z (m)')
hold on
for i=1:numQuads
    posH(i)=scatter3(p(1,i),p(2,i),p(3,i),50,colours(i),'XDataSource',sprintf('p(1,%d)',i),'
        YDataSource',sprintf('p(2,%d)',i),'ZDataSource',sprintf('p(3,%d)',i));
    set(posH(i),'MarkerEdgeColor',colours(i),'MarkerFaceColor',colours(i))
    %force(i)=quiver3(p(1,i),p(2,i),p(3,i),0,0,0);
end
axis equal
%% Run simulation
t=0;
it=1;
p1=zeros(numQuads,300);
p2=zeros(numQuads,300);
p3=zeros(numQuads,300);
bearingErrorAll=zeros(numQuads,300);
while t<tmax
    for i=1:numQuads
        % Compute relative positions/ranges/bearings
        for j=1:numQuads
            q(:,i,j)=p(1:2,j)-p(1:2,i);
            rOld(i,j)=r(i,j);
            r(i,j)=sqrt(q(:,i,j)'*q(:,i,j));
            rDot(i,j)=(r(i,j)-rOld(i,j))/dt;
            sOld(:,i,j)=s(:,i,j); % Update bearing measurements
            s(:,i,j)=q(:,i,j)/(sqrt(q(:,i,j)'*q(:,i,j)));
```

```matlab
            sDot(:,i,j)=(s(:,i,j)-sOld(:,i,j))/dt;
        end
        % Height controller
        ePID(1,i)=ePID(2,i);
        ePID(2,i)=1-p(3,i);
        ePIDInt(i)=ePIDInt(i)+((ePID(1,i)+ePID(2,i))/2)*dt;
        ePIDDot=(ePID(2,i)-ePID(1,i))/dt;
        grav(i)=Kp*ePID(2,i)+Ki*ePIDInt(i)+Kd*ePIDDot+mass(i)*g; % @@@@ Integral = trapezoidal rule
        for j=1:numQuads
            switch system
                case 0 % Old system
                    e(:,i,j)=c_k*(s(:,i,j)-sGoal(:,i,j));
                    rHat(i,j)=rHat(i,j)+(rDot(i,j)-(1/cobs)*(e(:,i,j))'/rHat(i,j)*sDot(:,i,j))*dt;
                    epsilon(:,i,j)=(1/rHat(i,j))*(eye(2)-s(:,i,j)*s(:,i,j)')*e(:,i,j);
                    delta(:,i)=-pDot(1:2,i)'*D_i;
                case 1 % New system
                    sigma(:,i,j)=sigma(:,i,j)+(1/G)*(c_k*(s(:,i,j)-sigma(:,i,j))-d_k*(sigma(:,i,j)-sGoal(:,i,j)));
                    rHat(i,j)=rHat(i,j)+(rDot(i,j)-(c_k/cobs)*(s(:,i,j)-sigma(:,i,j))'/rHat(i,j)*sDot(:,i,j))*dt;
                    epsilon(:,i,j)=(c_k/rHat(i,j))*(eye(2)-s(:,i,j)*s(:,i,j)')*(s(:,i,j)-sigma(:,i,j));
                    delta(:,i)=-pDot(1:2,i)'*D_i;
                case 2 % New system, fixed range independent
                    sigma(:,i,j)=sigma(:,i,j)+(1/G)*(c_k*(s(:,i,j)-sigma(:,i,j))-d_k*(sigma(:,i,j)-sGoal(:,i,j)));
                    rHat(i,j)=rHat(i,j)+(rDot(i,j)-(c_k/cobs)*(s(:,i,j)-sigma(:,i,j))'/rHat(i,j)*sDot(:,i,j))*dt;
                    epsilon(:,i,j)=(c_k*rHat(i,j))*(eye(2)-s(:,i,j)*s(:,i,j)')*(s(:,i,j)-sigma(:,i,j));
                    delta(:,i)=-pDot(1:2,i)'*D_i;
            end
        end
        % Total control force
        fc(:,i)=[nansum(epsilon(:,i,:),3);grav(:,i)]+[delta(:,i);0];
    end
    %% Update positions/velocities + plot data
    for i=1:numQuads
        pDotDot(:,i)=(fc(:,i)-mass(i)*g*[0;0;1])/mass(i); % Update Acceleration
        pDot(:,i)=pDot(:,i)+pDotDot(:,i)*dt; % Update velocity
        if(~fixed(i))
            p(:,i)=p(:,i)+pDot(:,i)*dt; % Update position
        end
        bearingError(i)=sum(nansum((s(1:2,i,:)-sGoal(1:2,i,:)).^2,3));
        bearingErrorAll(i,it)=sum(nansum((s(1:2,i,:)-sGoal(1:2,i,:)).^2,3));
        p1(i,it)=p(1,i);
        p2(i,it)=p(2,i);
        p3(i,it)=p(3,i);
    end
    %% Log data for LaTeX plots
    if (log==1)
        A=[t p(:,1)' p(:,2)' p(:,3)' p(:,4)'];
        fprintf(fileID,'%.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f
            \r\n',A);
        B=[t bearingError];
        fprintf(fileID2,'%.4f %.4f %.4f %.4f %.4f\r\n',B);
    end
    t=t+dt;
    it=it+1;
end
%% Plot all data at once
figure(4), xlabel('x (m)'), ylabel('y (m)'), zlabel('z (m)')
hold on
axis equal
for i=1:numQuads
    plot3(p1(i,:),p2(i,:),p3(i,:),colours(i));
end
figure(2)
subplot(3,1,2)
hold on
for i=1:numQuads
    plot((1:size(bearingErrorAll,2))*dt,bearingErrorAll(i,:),colours(i))
end
fclose('all')
```