Multi-agent Payload Manipulation and Transportation using Small Scale Quadrotor Platforms

Thomas J Miesen

A thesis
submitted in partial fulfillment
of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

University of Washington

2019

Committee:

Mehran Mesbahi, Chair

Behçet Açikmeşe

Program Authorized to Offer Degree:
Aeronautics and Astronautics

University of Washington

Abstract

Multi-agent Payload Manipulation and Transportation
using Small Scale Quadrotor Platforms

Thomas J Miesen

Chair of the Supervisory Committee:
Dr. Mehran Mesbahi
Aeronautics and Astronautics

We have demonstrated the application of a 2-dimensional formation protocol used for the manipulation of an unknown object supported by $n = 3$ agents. The protocol leverages a proportional-derivative consensus function which serves to maintain a rigid virtual linkage between agents while supporting a point-load. Additionally, a overlaying unit-vector consensus protocol was implemented as a simple means for agent distribution around a destination and formation orientation. All demonstrations were carried out in both simulation and on our custom quadrotor platforms. Within this thesis, we discuss both the experimental results, as well as the developmental process used to manufacture the tools required for application.

TABLE OF CONTENTS

LIST OF FIGURES

COMMON NOTATION

ξ: 3-dimensional inertial state vector in Cartesian coordinates $\left(\xi = \begin{bmatrix} x & y & z \end{bmatrix}^T \in \mathbb{R}^3\right)$

ϕ, θ, ψ: Roll, pitch, and yaw (respectively) Euler angles defined by a R_{zyx} rotation matrix.

φ: 3-dimensional rotational state vector, where: $\varphi = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T \in \mathbb{R}^3$

K_p, K_d, K_i: Proportional, derivative, and integral (respectively) control gains.

e_3: Z-component unit vector, $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T$

d_i : Unit-vector pointing from the i-th quadrotor to its anchor on a supported payload, defined in the inertial coordinate frame.

q_i : Vector pointing from the center of mass of the payload to the i-th quadrotor anchor, defined in the inertial coordinate frame.

$R_{(...)}$: Body-frame to inertial frame rotation matrix for a given body of interest.

$m_{(...)}$: Mass for a given body of interest in kg.

$I_{(...)}$: Inertial tensor for a given body of interest.

$u_{(...)}$: Control input for a given controller or body of interest.

$A_{(...)}$: State-space representation of the natural system dynamics of a given body.

$B_{(...)}$: State-space representation of the control mapping matrix relating a control input to the evolution of the state for a given body.

ACKNOWLEDGMENTS

I would first like to acknowledge the RAIN lab members, for their patience and support throughout this project. I also wish to acknowledge the creators of Arduino, Python, and Blender and their respective open-source communities, as well as the University of Washington Co-Motion Maker-space. Each of these groups helped provide the tools and resources that made this project – and many others – possible.

DEDICATION

I dedicate this thesis to those whom supported me in my time at the University of Washington. To my family, who have given me nothing by love and support at every junction. To my loving partner, Amanda, who has helped inspire me to persevere and go beyond. To my amazing Adviser, Mehran, for giving me the opportunity and resources that made this thesis possible.

Chapter 1
INTRODUCTION

Over the past decade, the commercial availability of quadrotors and multi-rotors has grown exponentially. This increase has inspired countless consumer and research endeavors leveraging this versatile tool for a multitude of applications. Initially much of the research focus was centered around deepening our understanding of the fundamental dynamics and control techniques available for the platform. Now that our understanding of the fundamentals has matured, the focus of research has shifted more towards what applications and challenges these platforms can undertake [24, 25]. With the high degree of spatial control offered by quadrotors, one of the natural potential applications is object manipulation and transportation [13, 16, 18, 21, 23].

Practical object transportation presents a number of lateral considerations and challenges that need to be addressed; in particular, how the payload is supported and secured during transport. Each support method offers a range of advantages and disadvantages. For example, cable-suspended loads rely purely on tension for support and often serve as the simplest support scheme available. The chief draw-back to cable-suspension is payload swing, which often must be accounted for by maneuvering the supporting agent [11, 14]. As an alternative to cable suspension, the quadrotor can directly attach to the payload itself using a rigid linkage or anchor point. This mode of attachment addresses payload swing, but comes at the cost of high coupling between support agent states and alteration of control response characteristics [16, 30].

As with most single agent applications, tasks like payload manipulation can be naturally extended to distributed systems. Distribution across multiple agents not only reduces the individual workload, but also affords certain payload manipulation schemes a higher degree

of control authority over the payload state. This increased influence can then be levereaged to address limitations presented by certain support methods. For example, the post-maneuver swing encounterd in a cable-supsended system is mitigated when supported by three or more agents [23]. A key consideration when dealing with any coordinated multi-agent system is formation control.

In the context of coordinated tasks like payload manipulation, there are two primary considerations that must be weighed against the system setup. First, is inter-agent formation maintenance. For systems where the attachment anchors are physically connected to one another, the inter-agent forcing is naturally induced by the physical forces being transferred through the payload support frame. However, in systems lacking such physical connections between agents, inter-agent forcing must be actively generated by a formation controller to maintain formation cohesion. Second, is the formation control approach demanded by eccentric or asymmetric payloads, where the center of mass or shape of the payload requires a change in formation positioning or thrust compensation depending on if the agents are free to change their position relative to the payload.

The primary focus of this thesis will be the 2D manipulation of a cable-suspended payload, supported by a formation of quadrotors. The network formation control techniques developed by the Robotics Aerospace and Information Networks (RAIN) lab provide a very attractive and elegant means of distributed actuation, often with a very low computational overhead [5, 17, 22]. As such, this thesis will discuss the process of development and practical implementation of these control algorithms for payload manipulation using quadrotors.

1.1 Quadrotor Development for the RAIN Lab

Historically, the RAIN lab has primarily dealt with demonstration of its network algorithms on ground-based robotic platforms and simulations of various physical systems [5, 12, 22]. In order to develop, implement, and validate the control algorithms developed throughout this thesis, the existing hardware infrastructure of the RAIN lab was

updated to support areal drone operation. This section describes the various avenues considered for both the final quadrotor design, and payload transportation techniques.

The available flight space within the RAIN lab is relatively small (around $2 \times 2 \times 2$ m). As such, the size of the selected quadrotor was restricted to examining the small-scale (< 300 mm max diameter) range. The following section discusses a variety of quadrotors currently available for research applications, with any larger quadrotors being referenced for features that may be desired in a final platform.

1.1.1 Existing Designs and Approaches

As quadrotors have shifted gradually from the research to the industrial and consumer spheres, there are a variety of commercially produced quadrotors available to the general public. Each option can come in a in a wide range of sizes, price points, and levels of functionality.

Figure 1.1: Product photograph of the Ascending Technologies Hummingbird [26]

.

AscTec Hummingbird: The AscTec Hummingbird was developed by Ascending Technologies specifically for research applications. It houses a low-level processor that handles flight stability maintenance at a rate of 1 kHz, guided by a higher-level processor

specifically designed to run custom user code using an Intel Atom processor. Communication is handled by a low-power Xbee 2.4 GHz wireless card, with a range of around 1000m. While this platform boasts a number of perks (brushless DC motors, emergency modes, etc), the profile (540×540 mm) is much too large for the flight space available in the lab, however it serves as a good reference for the higher performance spectrum.

Figure 1.2: Product Photograph of Crazieflie 2.1 small-scale quadrotor [1]

.

CrazyFlie: This is one of the few commercially available, small-scale quadrotors ($92 \times 92 \times 29$ mm) that is also specifically designed for research applications, and has been used in a variety of research laboratories for single agent and swarm demonstrations. The on-board state estimation uses a 10-Dof sensor array (3-axis gyro, accelerometer, compass, and pressure transducer). The coding back-end is completely open-source and community maintained, allowing for custom packages to be developed for research applications. The Crazy-PA system uses a 2.4GHz NRF24L01+ network card to connect to up to 4-5 agents

simultaneously. The system can be further expanded in functionality through purchase and installation of expansion decks sold by Bitcraze. The two processors on the crazyflie are the STM32F405 (Cortex-M4, 168 MHz) processor tasked with operating the main applications, and the nRF51822 (Cortex-M0, 32 MHz) tasked with radio and power management. The performance of this platform coupled with the technical support available, makes this the most attractive commercial option available, despite the higher cost (around $200 /unit) [1].

Figure 1.3: Product Photograph Cazepony small-scale quadrotor [4].

CazePonie: The Crazeponie is another commercially available small-scale quadrotor (100×100 mm) with comparable specifications to the Crazyflie 2.1 at a lower price point. The on-board processing is handled by a single STM32F103T8U6 processor (72 MHz), and state measurements handled by a MPU6050 IMU (3-axis gyro and accelerometer) and altimeter. The Crazepony uses a 2.4 GHz NRF24L01+ network card for wireless communication. At around $80/unit, this quadrotor best represents the lower-end of the commerically avialable quadrotors with a high degree of functionality [4].

Figure 1.4: Product photograph of the PepperF15h open-source small-scale quadrotor platform [7].

PepperFish by Fishpepper: The home-brew RC-drone community has a number of very unique custom-builds that produce high-fidelity quadrotors for a very low price point, and is also one of the smallest (66×66 mm). Most of the groups working on such projects tend to focus on mid-size to larger quadrotors [3]. One community member by the handle Fishpepper, developed their own custom PCB and control boards, complete with schematics and coding available open-source to the general online community. They also out-sourced some of their hardware to be fabricated and sold online at a fair price point for the compact size of the parts. This is the smallest quadrotor on this list with complete modularity, and allows for the addition of an on-board camera, all at a price point around $100/unit. The primary drawback of this platform is that it is designed to be a human-piloted, meaning controlling algorithms will need to be developed to interface with the joystick inputs used by the lower-level flight controller, and thus may restrict some potential application methods later on [7].

Figure 1.5: Product photograph of an open-source Arduino-based quadrotor developed by Joop Brokking [3].

YMFC Arduino Auto-leveling Quadrotor: Another example of the RC-drone community that features the usage of the Arduino open-source IDE for code development and update of the quadrotor platform. While this particular platform is in the larger range of quads, the underlying control and code infrastructure can easily be adapted to a smaller configuration. In addition, the nature of Arduino micro-controllers allows for the most flexibility in terms of addition of sensors, multi-level controllers, and other platform modifications. While the project presented by this particular example is designed to produce another human-piloted quadrotor, the inputs to the control interface are fully able to be designed to allow for any desired inputs deemed necessary by the development team [3].

1.2 Payload Transportation

This section discusses a selection of attachment and transportation methods currently used in literature for quadrotor platforms known to the author. Each of these methods presented will have a single and multi-agent counterpart, and will express the various challenges they present.

1.2.1 Existing Methods for Object Manipulation using Quadrotors

Cable-suspension: Cable suspension is one of the simplest methods of attachment available for a quadrotor system. At minimum this scheme requires a single point of attachment on both the payload and supporting agent, each connected by a stiff cable rated for the desired support load. The placement of the anchors, movement of the payload, and number of supporting agents is often where much of the complexity begins to show.

First, we need to consider the limitations of cables as a force transfer mechanism. As cables can only transfer tensile force, they are unable to provide any forcing in the direction of the payload. For a single agent operating under a gravitational field, the inability of the cable to provide a pushing force is offset by the pull of gravity on a payload. A simple example is a payload at rest supported by a quadrotor, which then reduces its altitude and results in the cable losing tension. We will denote losses in tension as *slacking*. Figure 1.6 shows how gravity aids in tension recovery after the cable has been slacked. Depending on the application, the model can be assumed to be quasi-static, and thereby

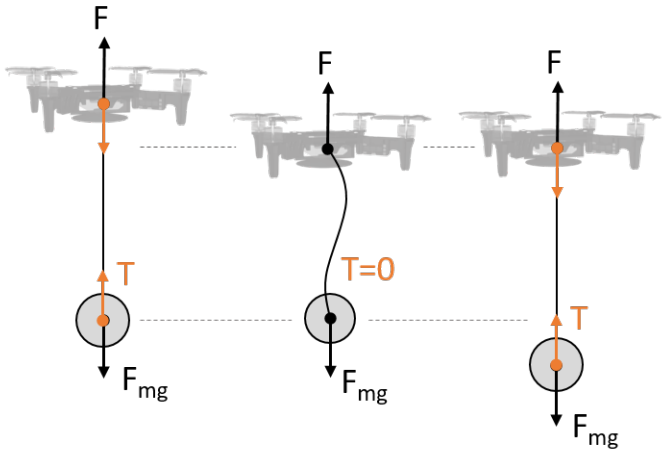

Figure 1.6: Graphical example of a cable-suspended load slacking and losing tension after the quadrotor drops in altitude. Due to the pull of gravity on the payload, the cable is able to regain tension, after the mass drops and matches the altitude reduction of the quadrotor.

mathematically ignore the slacking effect, however this may result in poor overall performance [23]. Additionally, the cables are often assumed to be mass-less (relative to the system) and inelastic.

A purely tension-driven mechanism also means that translation of the payload can only be achieved by leading the payload motion. In essence, both moving and stopping the motion of the payload requires the quadrotor to be positioned ahead of the payload in the direction of desired force. The naiive approach to translation would be to simply move slowly in the direction of choice, thereby reducing the lateral forces required to start and stop the motion of the payload. Althernatively, if the dynamics of the payload are relatively well understood, then the control response of the quadrotor itself can be updated by some control or trajectory response relative to the motion of the payload [14].

Many of the above challenges can be addressed by the addition of multiple supporting agents. For istance, a minimum of three supporting agents is sufficient to address the payload swing, and maintain full translational control over a point-load [23]. The new challenges presented by introduction of multiple agents fall on to agent distribution, anchor placement, and formation control, which will be discussed in further detail in the payload dynamics section.

Fixed-linkage: For the scope of this thesis, any system using a ridged connection to the payload is grouped into the category of a fixed-linkage system. Often these systems are designed to address the limitation presented by cable suspension. As the payload is directly attached to the quadrotor itself, the dynamics of each merge to form a new dynamical system. For a single agent, this approach offers a number of advantages for control, as the forces applied by the quadrotor are inherently applied directly to the payload.

In the case of multi-agent systems that are physically attached to a common payload, the dynamics of the system as a whole become significantly more complex [16], where the control actuation of each quadrotor now directly influences one another through the common system. This interdependence can be reduced by addition of mechanical degrees of

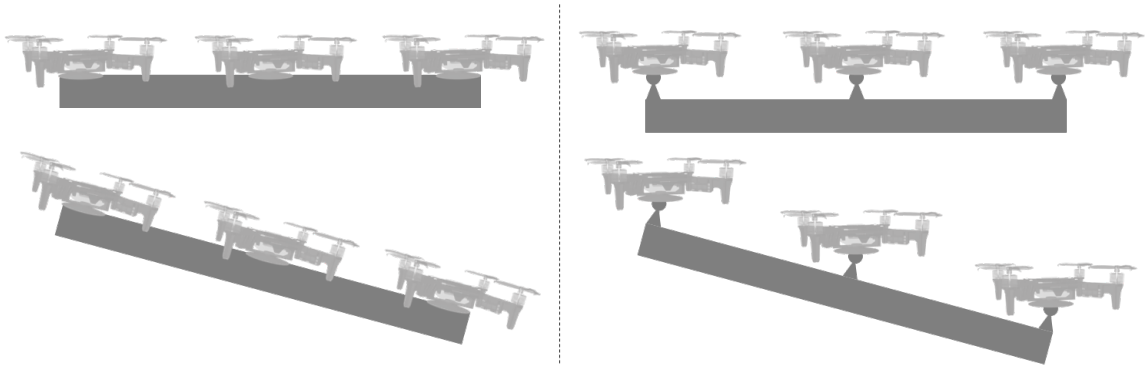

Figure 1.7: An illustrative comparison between the grasping and ball-joint attachment methods with multiple agents. (Left) For a grasping system, an altitude change made by one agent affects the attitude of both the payload and all other attached agents. (Right) For a ball-joint anchored system, an altitude change made by one agent will only predominantly influence the attitude of the payload itself and the altitude of any agents attached along the linkage.

freedom to the anchor mechanism. For example use of a ball-joint anchor [13] in place of a grasping mechanism [16, 30], allows for the quadrotors to maintain relatively independent control over their attitude dynamics, with the inertial distance between agents held constant by the linkage. Figure 1.7 shows an example of how the ball-joint attachment mechanism can influence payload control and the attitude of each quadrotor. Reducing the influence each agent has over another agents state helps reduce the amount of state information a given agent needs to posses of neighboring agents. In the case of a ball-jointed system, the information required to inform an individual control response is reduced to the inertial states.

Chapter 2

SYSTEM DYNAMICS AND CONTROL DESIGN

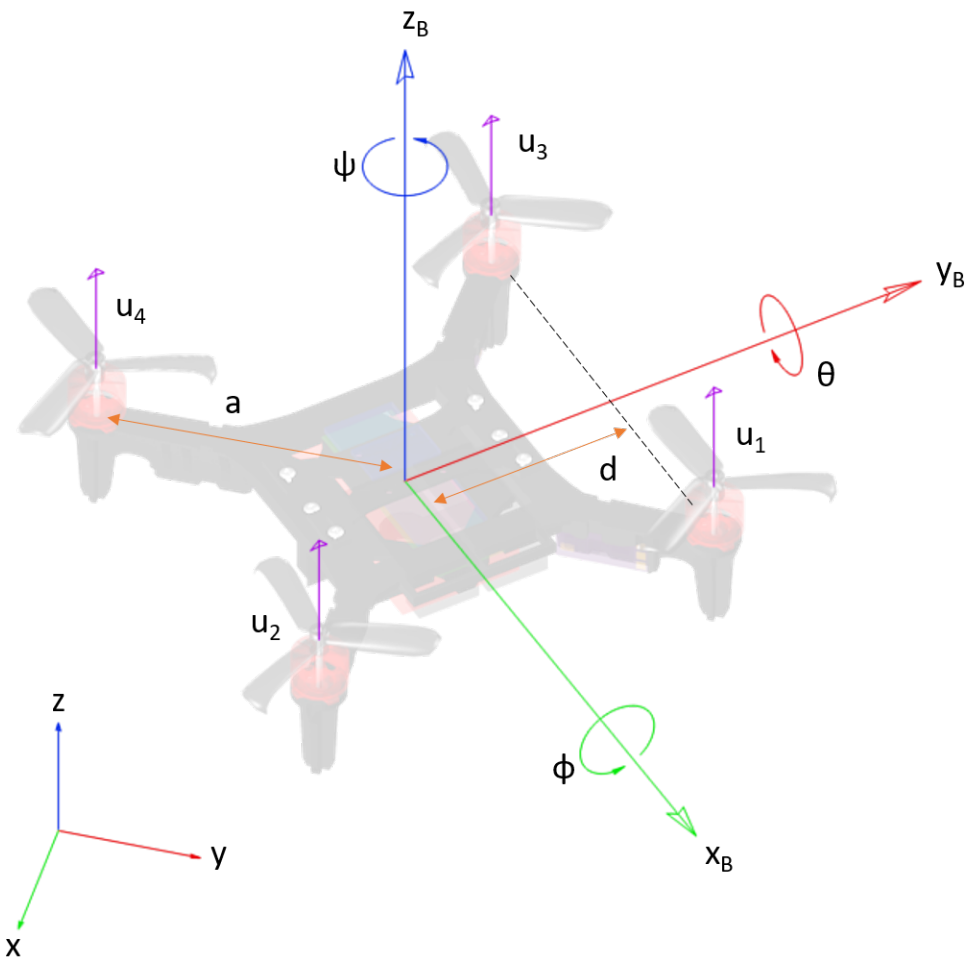

Figure 2.1: Diagram of the RAIN-Drop quadrotor global and body-frame definitions.

2.1 Quadrotor

Discussed in this section are the governing dynamics and kinematics for a general quadrotor platform, followed by several approaches for platform control. These control functions will included both the local control protocol, as well as any relevant global trajectory generation algorithms used to ensure convergence of the controller to an arbitrary point in space. The payload dynamics covered will be specifically related to the cable-suspension method discussed in Chapter 1, applied to both a single-anchor point-mass, and a multi-anchor rigid-body payload.

2.1.1 Quadrotor Dynamics and Linearization

Dynamics and Kinematics: For a general quadrotor, the inertial dynamics can be derieved and expressed as a set of Newton-Euler equations [2, 20, 23],

$$m\ddot{\xi} = FRe_3 - mge_3 \tag{2.1}$$

where $\xi = (x, y, z)^T$, F is the magnitude of the thrust applied to the body frame, e_3 is the unit vector in the global z direction $e_3 = (0, 0, 1)^T$, m is the mass of the quadrotor, g is gravity, and R is the rotation matrix.

Assumption: *Non-linear effects such as: drag, hub-monents, and blade-flapping [2], are assumed to be small, or within the band-width of control response for the final control protocol.*

As the constructed quadrotor has four fixed actuators that rotate with the body-frame during operation, it is necessary to account for such rotations in the dynamical equations relating the thrust output vector of the quadrotor, to the global inertial accelerations acting on the platform. Additionally as with any rotation, the order of applied rotations dramatically affects the final value, thus a R_{zyx} Euler rotation matrix defined and be used

for all future calculations,

$$R_{zyx} = R_x R_y R_z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

Which would map from the inertial frame to the body frame, meaning our map for the forces from the body to the inertial is simply,

$$R = R_{zyx}^T$$

The force acting on the body frame is defined by the summation of each of the motor inputs,

$$F = u_{tot} = u^T \mathbf{1}$$

Where $u = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^T$ and $\mathbf{1} = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T$. For the purposes of explicitly showing the linearization process used, these equations can be expanded from their compact matrix form,

$$F_x = u_{tot}(\sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi) \tag{2.3}$$

$$F_y = -u_{tot}(\sin\phi\cos\psi - \cos\phi\sin\theta\sin\psi) \tag{2.4}$$

$$F_z = u_{tot}\cos\theta\cos\phi - mg \tag{2.5}$$

Resulting in a system of ODEs,

$$\begin{bmatrix} \dot{\xi} \\ \ddot{\xi} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \frac{u_{tot}}{m}(\sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi) \\ \frac{u_{tot}}{m}(-\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi)) \\ \frac{u_{tot}}{m}(\cos\theta\cos\phi) - g \end{bmatrix} \tag{2.6}$$

Similarly for the Newton-Euler rotational dynamics [2, 20, 27],

$$I\dot{\omega} = -\omega \times I\omega + T \tag{2.7}$$

Where I is the principal moment of inertia matrix,

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$$

ω is the angular velocity matrix,

$$\omega = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

and T is the stacked vector of the torques acting on the system, where each element is a torque acting on a specific axis of interest,

$$T = \begin{bmatrix} T_\phi \\ T_\theta \\ T_\psi \end{bmatrix}$$

Note that as the configuration is an 'X' format, the torques are applied differently than the '+' configureation presented in [2], where the moment-inducing thrust elements are now the combination of the two motors acting along the line connecting each motor as presented in [27]. We can account for this configuration change by defining a mapping matrix, M, which maps the motor inputs to the principal torques for a given configuration. Using the 'X' configuration yields the following,

$$T = \begin{bmatrix} d & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & a \end{bmatrix} Mu, \qquad M = \begin{bmatrix} 1 & -1 & 1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 \end{bmatrix}$$

Where the definitions for d and a are shown in fig. 2.1. Expanding T into three independent equations,

$$T_\phi = d(u_1 - u_2 + u_3 - u_4)$$

$$T_\theta = d(-u_1 - u_2 + u_3 + u_4)$$

$$T_\psi = d(-u_1 + u_2 + u_3 - u_4)$$

Solving for $\dot{\omega}$ in (2.7) yields the following,

$$\ddot{\phi} = \frac{1}{I_{xx}}(d(u_1 - u_2 + u_3 - u_4) - \dot{\theta}\dot{\psi}(I_{yy} + I_{zz})) \tag{2.8}$$

$$\ddot{\theta} = \frac{1}{I_{yy}}(d(-u_1 - u_2 + u_3 + u_4) - \dot{\phi}\dot{\psi}(I_{xx} - I_{zz})) \tag{2.9}$$

$$\ddot{\psi} = \frac{1}{I_{zz}}(C_\alpha(a(-u_1 + u_2 + u_3 - u_4) - \dot{\theta}\dot{\phi}(I_{xx} + I_{yy})) \tag{2.10}$$

Where C_α is some coefficient relating the portion of vertical thrust that is redirected into the xy-plane, acting on the z-axis (and consequently acting on ψ).
Condensing these equations into the vector form, where $\varphi = (\phi, \theta, \psi)^T$,

$$\begin{bmatrix} \dot{\varphi} \\ \ddot{\varphi} \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \frac{1}{I_{xx}}(d(u_1 - u_2 + u_3 - u_4) - \dot{\theta}\dot{\psi}(I_{yy} + I_{zz})) \\ \frac{1}{I_{yy}}(d(-u_1 - u_2 + u_3 + u_4) - \dot{\phi}\dot{\psi}(I_{xx} - I_{zz})) \\ \frac{1}{I_{zz}}(C_\alpha(a(-u_1 + u_2 + u_3 - u_4) - \dot{\theta}\dot{\phi}(I_{xx} + I_{yy})) \end{bmatrix} \tag{2.11}$$

Dynamics and Kinematics Linearization: There are a number of popular control approaches that use a system of the form $\dot{x} = Ax + Bu$. In addition, these matrices can serve a number of other beneficial pruposes from derivation of the full system transfer-function [19, 20] and Kalman-filtering [28]. As such, for the purposes of this thesis, and those seeking to utilize these matrices for their own research applications, the linearization procedure will be outlined as follows:

In general a system can be linearized about some point, and written in a linear form,

$$X = \begin{bmatrix} \xi \\ \varphi \\ \dot{\xi} \\ \dot{\varphi} \end{bmatrix} \implies \dot{X} = AX + Bu = \begin{bmatrix} \dot{\xi} \\ \omega \\ \ddot{\xi} \\ \dot{\omega} \end{bmatrix}$$

For the quadrotor, the equations (2.6) and (2.11) are linearized by determining the Jacobian (derivative with respect to each element). The linearization process results in the following A matrix,

$$A = \begin{bmatrix} 0_{[6x6]} & I_{[6x6]} \\ G_{[6x6]} & 0_{[6x6]} \end{bmatrix} \tag{2.12}$$

Where I_{6x6} is a 6×6 identity matrix, and G is the lower matrix block containing the remaining state information for A,

$$G = \begin{bmatrix} 0 & 0 & 0 & (\frac{u_{tot}}{m})(c\phi s\psi - s\phi s\theta c\psi) & (\frac{u_{tot}}{m})(c\phi c\theta c\psi) & \frac{u_{tot}}{m}(s\phi c\psi - c\phi s\theta s\psi) \\ 0 & 0 & 0 & -(\frac{u_{tot}}{m})(c\phi c\psi + s\phi s\theta s\psi) & (\frac{u_{tot}}{m})(c\phi c\theta s\psi) & \frac{u_{tot}}{m}(s\phi s\psi + c\phi s\theta c\psi) \\ 0 & 0 & 0 & -(\frac{u_{tot}}{m})(c\theta s\phi) & -(\frac{u_{tot}}{m})(s\theta c\phi) & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The primary linearization point of interest for the quadrotor control protocol, is a resting hover state. This state implies that the x_B, y_B is level with respect to the global x, y plane, such that $z \approx z_B$. At a resting hover state, the quadrotor thrust vector operates in the same axial direction as gravity, with minimal lateral forces. In order to achieve this, the attitude must be level ($\phi \approx 0$ and $\theta \approx 0$). To maintain a resting position, the thrust must exactly cancel the gravitational force acting on the system ($F = mg$), and the remaining system rates must be zero, resulting in the linearization point,

$$(\xi, \varphi, \dot{\xi}, \dot{\varphi}) = (0, 0, 0, 0, 0, \psi, 0, 0, 0, 0, 0, 0), \qquad u_{tot} = mg \tag{2.13}$$

Where ψ is left as a free variable for orientation of the A matrix.

Substituting (2.13) into (2.12) results in the following simplified G,

$$G = \begin{bmatrix} 0 & 0 & 0 & gs\psi & gc\psi & 0 \\ 0 & 0 & 0 & -gc\psi & gs\psi & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Similarly linearizing (2.6) and (2.11) with respect to the control inputs to determine the control-input Jacobian,

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{s\phi s\psi + c\phi s\theta c\psi}{m} & \frac{s\phi s\psi + c\phi s\theta c\psi}{m} & \frac{s\phi s\psi + c\phi s\theta c\psi}{m} & \frac{s\phi s\psi + c\phi s\theta c\psi}{m} \\ \frac{s\phi c\psi - c\phi s\theta s\psi}{m} & \frac{s\phi c\psi - c\phi s\theta s\psi}{m} & \frac{s\phi c\psi - c\phi s\theta s\psi}{m} & \frac{s\phi c\psi - c\phi s\theta s\psi}{m} \\ \frac{c\phi c\theta}{m} & \frac{c\phi c\theta}{m} & \frac{c\phi c\theta}{m} & \frac{c\phi c\theta}{m} \\ (\frac{d}{I_{yy}}) & (-\frac{d}{I_{yy}}) & (\frac{d}{I_{yy}}) & (-\frac{d}{I_{yy}}) \\ (-\frac{d}{I_{xx}}) & (-\frac{d}{I_{xx}}) & (\frac{d}{I_{xx}}) & (\frac{d}{I_{xx}}) \\ (-C_\alpha \frac{a}{I_{zz}}) & (C_\alpha \frac{a}{I_{zz}}) & (C_\alpha \frac{a}{I_{zz}}) & (-C_\alpha \frac{a}{I_{zz}}) \end{bmatrix} \tag{2.14}$$

Linearizing (2.14) about (2.13),

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & \frac{1}{m} & \frac{1}{m} & \frac{1}{m} \\ (\frac{d}{I_{yy}}) & (-\frac{d}{I_{yy}}) & (\frac{d}{I_{yy}}) & (-\frac{d}{I_{yy}}) \\ (-\frac{d}{I_{xx}}) & (-\frac{d}{I_{xx}}) & (\frac{d}{I_{xx}}) & (\frac{d}{I_{xx}}) \\ (-C_\alpha \frac{a}{I_{zz}}) & (C_\alpha \frac{a}{I_{zz}}) & (C_\alpha \frac{a}{I_{zz}}) & (-C_\alpha \frac{a}{I_{zz}}) \end{bmatrix}$$

General Position Control at a Non-zero Yaw: Another way to account for the quadrotor yaw rotation in a general control policy $u = -K(x - x_d)$, is to rotate the error being input into that matrix. Around hover, quadrotor's body-frame z-axis corresponds to the global z-axis, and as such we can apply a simple rotation to the global xy elements, and thereby allowing for the correct application of the control response. Let's consider the rotation about z, ignoring the z-axis elements,

$$\begin{bmatrix} x_B \\ y_B \end{bmatrix} = \begin{bmatrix} \cos(\psi) & \sin(\psi) \\ -\sin(\psi) & \cos(\psi) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.15}$$

The control policy can now be simply applied to the rotated state vector to achieve the desired response,

$$u = -K \begin{bmatrix} x_B \\ y_B \end{bmatrix} = -K \begin{bmatrix} x\cos(\psi) + y\sin(\psi) \\ y\cos(\psi) - x\sin(\psi) \end{bmatrix} \tag{2.16}$$

If the state matrix (2.12) were to be used to generate an LQR control policy about $\psi = 0$, then that same control policy could then be used in general, by rotating the error.

2.1.2 Estimation Methods

Realistically, it is not always possible to measure the states of a given system. As such there are a number of estimation approaches available to infer the desired state information. Data filtration is another form of estimation, where the true measurement of the system veiled behind signal noise introduced either via the sensor measurement or the system dynamics. This section outlines the approaches considered by this thesis for use on the live platform.

Low-Pass Filter: The bulk majority of measurement noise encountered in most systems is high frequency. In particular on quadrotors, vibration is the chief culprit to noise intrusion on attitude measurements, along with the electrical noise introduced by the motor operation. One of the simplest methods available to address this is using a low-pass filter of the form,

$$y(t) = (1 - k)y(t - 1) + kz(t)$$

where $y(t)$ is the estimate of the noisy measurement $z(t)$. It is often convenient to define k in term so of a cut-off frequency f_c by,

$$k = \frac{2\pi\Delta t f_c}{2\pi\Delta t f_c + 1}$$

This cut-off frequency defines the frequency at which perturbations of a higher frequency will decay at a rate of $20\frac{dB}{dec}$ from the cut-off.

Complementary Filter: The complimentary filter, merges a weighted average of two estimated or measured signals of a common state, resulting in a complementary state estimate,

$$y_c(t) = (1 - k_c)y_1(t) + ky_2(t)$$

Where the value of k_c defines which of measurements $y_1(t)$ and $y_2(t)$ is trusted more, and thereby comprises the bulk of the final measurement.

Finite-difference Approximation: The simplest approximation for a derivative is the finite difference approximation. In particular for real-time operating platforms, we're interested in estimating the current derivative of a given state, based off of that states historical data. Often this is referred to as a Backwards-difference approximation,

$$\frac{d}{dt}y(t) \approx \frac{y(t) - y(t - dt)}{dt}$$

While this approach is quite simple to implement, it is important to note that the primary drawback of this approach is the inherent discretization error that's incurred as the value of dt increases. Often this error is referred to a truncation error. This is made more apparent when considering the Taylor expansion of a previous time-step point $y(t - dt)$ from $y(t)$,

$$y(t - dt) = y(t) - dt\frac{d}{dt}y(t) + \frac{dt^2}{2!}\frac{d^2}{dt^2}y(t) - \frac{dt^3}{3!}\frac{d^3}{dt^3}y(t) + \mathcal{O}(dt^4)$$

If we define the truncation error as,

$$TE = \frac{dt^2}{2!}\frac{d^2}{dt^2}y(t) - \frac{dt^3}{3!}\frac{d^3}{dt^3}y(t) + \mathcal{O}(dt^4) = \mathcal{O}(dt^2)$$

Substituting back into our Taylor series,

$$y(t - dt) = y(t) - dt\frac{d}{dt}y(t) + \mathcal{O}(dt^2)$$

$$\frac{d}{dt}y(t) = \frac{y(t) - y(t - dt)}{dt} + \mathcal{O}(dt)$$

Where we can see that our error is of order dt, and thereby increases as dt increases.

Forward Euler approximation: One of the simplest means of approximating an integral available represented by,

$$y(t) \approx y(t - dt) + dt(\dot{y}(t))$$

As this approximation is derived directly from the finite-difference approximation, and thereby shares the same inherent estimation error incurred at higher values of dt.

Kalman Filtering: A recursive filter that compares an estimate of the expected state output from a given control input, and the measured state. For discrete time, this filter consists of two primary steps [28],

Update/Correction:
$K(k) = P^{-}(k)H^{T}(HP^{-}(k)H^{T} + R)^{-1}$
$\hat{x}(k) = \hat{x}^{-}(k) + K(k)\left(z(k) - H\hat{x}^{-}(k)\right)$
$P(k) = \left(I - K(k)H\right)P^{-}(k)$
Prediction:
$\hat{x}^{-}(k+1) = A\hat{x}(k) + Bu(k)$
$P^{-}(k+1) = AP(k)A^{T} + Q$

Where $K(k)$ is the Kalman gain, $P(k)$ is the updated error covariance, $\hat{x}(k)$ is the state estimate, and $z(k)$ is the state measurement all at time step k. H is the noise-less mapping matrix from the state-space to the observation space, and A and B represent the linearized state and control matrices, defined for quadrotors as equations (2.12) and (2.14). Variables denoted with a superscript '-' (e.g. $x^{-}(k)$) represent variables that have not yet been updated with current measurement data (often referred to as *a priori*).

Kalman filtering represents a powerful method of estimation and noise-rejection based off the known model characteristics of the system. One challenge that needs to be taken into consideration is how the estimation fidelity is affected by large alterations to the dynamical system. In the case of object transportation, as discussed in chapter 1, attachment to a payload and introduction of multiple agents influencing the dynamics of that payload and thereby that of the quadrotor, harbors the potential to influence estimation fidelity of a model-based approach.

2.1.3 Quadrotor Controls

Attitude Control

PD and PID: There are a number of control design techniques used to produce a stable attitude controller for a quadrotor. A simple practical approach used in academic papers and the general quadrotor flying community is gain tuning [3]. This is accomplished by setting up a rocking beam, with a freely rotating joint at the center, and mounting a motor at each end of the beam. Each PID gain is tuned individually until near instability (unstable oscillations from disturbances), and then each gain is set to below that value. Once all gains are enabled, the user then makes minor adjustments based off the combined performance. Indeed this approach works quite well for setting up a base-line attitude controller quickly, however due to the non-analytic approach to determining these gains, there is no information on how stable these gains are, or if other stabilizing gains exist. Additionally, if any modification is made to the control code that may impact the rate of controller update, we have no information to determine whether the new update rate will be stable with the previous control gains. Instead we can make an effort to examine the frequency-domain response of our linearized system dynamics, by determining the attitude transfer function.

As the system kinematics are relatively well understood, we can define an A matrix for our attitude dynamics,

$$A_r = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\dot{\psi}(I_{xx} - I_{zz}) & 0 & -\dot{\theta}(I_{xx} + I_{yy}) \\ 0 & 0 & 0 & 0 & -\dot{\psi}(I_{yy} + I_{zz}) & -\dot{\phi}(I_{xx} + I_{yy}) \\ 0 & 0 & 0 & -\dot{\phi}(I_{xx} - I_{zz}) & -\dot{\theta}(I_{yy} + I_{zz}) & 0 \end{bmatrix}$$

If we consider the linearization point (2.13), results in only a simple upper block identity

matrix.

$$A_r = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Similarly we can derive our rotational control matrix,

$$B_r = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ (\frac{d}{I_{yy}}) & (-\frac{d}{I_{yy}}) & (\frac{d}{I_{yy}}) & (-\frac{d}{I_{yy}}) \\ (-\frac{d}{I_{xx}}) & (-\frac{d}{I_{xx}}) & (\frac{d}{I_{xx}}) & (\frac{d}{I_{xx}}) \\ (-C_\alpha \frac{a}{I_{zz}}) & (C_\alpha \frac{a}{I_{zz}}) & (C_\alpha \frac{a}{I_{zz}}) & (-C_\alpha \frac{a}{I_{zz}}) \end{bmatrix}$$

If we consider a PD control protocol for the attitude dynamics (against some reference point $\varphi_d, \dot{\varphi}_d$) which is mapped to the 'X' configuration,

$$u_r(t) = -K_r \begin{bmatrix} \varphi - \varphi_d \\ \dot{\varphi} - \dot{\varphi}_d \end{bmatrix} = -M^T \begin{bmatrix} K_{\phi,p} & 0 & 0 & K_{\phi,d} & 0 & 0 \\ 0 & K_{\theta,p} & 0 & 0 & K_{\theta,d} & 0 \\ 0 & 0 & K_{\psi,p} & 0 & 0 & K_{\psi,d} \end{bmatrix} \begin{bmatrix} \varphi - \varphi_d \\ \dot{\varphi} - \dot{\varphi}_d \end{bmatrix} \quad (2.17)$$

Incorporating K_r into the linearized dynamics results in the linearized closed-loop system,

$$A_{r,cl} = (A - BK_r)$$

The rocking-beam stabilizing procedure will provide the stabilizing gains for the ϕ and θ axes individual, which will then be adjusted to stabilize the entire platform. With these gains, we can analyze the real eigenvalues of the closed-loop system. It is important to note that this analysis will only capture the linearized system eigenvalues, and will ignore the

higher-order effects hidden through the linearization process, in particular the influence of the cross-coupled body rates. In Matlab this is accomplished by fixing all but one of the control gains to their empirically determined values, and iterating the non-fixed gain through a defined range. As the platform is a discrete-time system, we use the Matlab c2d() function to determine the zero-order-hold discrete-time system matrices A_{rd}, B_{rd}. The pseudo code of this analysis for a PD control law is shown in figure 2.2.

$A_r = [...]$, $B_r = [...]$, $M = [...]$
$A_{rd} = c2d\,(A_r)$, $B_{rd} = c2d\,(B_r)$
$k_p =$ actual value
for $k_d = k_{min} : dk : k_{max}$
  $K = [k_p * eye(3), k_d * eye(3)]$
  $E = eig(A_d - B_d M' K)$
  plot(k_p,real(E))
end
$k_d =$ actual value
for $k_p = k_{min} : dk : k_{max}$
  $K = [k_p * eye(3), k_d * eye(3)]$
  $E = eig(A_d - B_d M' K)$
  plot(k_d,real(E))
end

Figure 2.2: Psuedo code used to coarsely analyze the real eigenvalues around the empirically determined stabilizing gains for a PD attitude controller.

Position Control

LQR A popular and well-documented approach to optimal control is a Linear Quadratic Regulator for an infinite time-horizon in discrete time [10], satisfies the following cost,

$$V(x,u) = \frac{1}{2}\sum_{k=0}^{\infty}\left(x(k)^T Q x(k) + u(k) R u(k)\right)$$

where the goal is to determine the optimal control with respect to the cost function,

$$\mathbf{u} = \min_{u} V(x,u)$$

subject to the conditions,

$$x(k+1) = Ax(t) + Bu(t)$$

$$x(0) = x_0$$

The solution for an optimal control policy $\mathbf{u} = -Kx$ can be determined from the solution to the Discrete-time Algebraic Riccati Equation (DARE),

$$P = A_d^T P A_d - A_d^T P B_d \left(R + B_d^T P B_d\right)^{-1} B_d^T P A_d + Q$$

where A_d and B_d are the zero-order-hold discrete-time versions of equations (2.12) and (2.14). Generally this recursion is performed until the solution to P converges to a given value, and then can be used to produce an optimal static gain,

$$K = -\left(B_d^T P B_d + R\right)^{-1} B_d^T P A_d$$

The advantage of this control design approach, is the ability to both produce a stabilizing control gain with respect to your system matrices, that affords the potential to update as the platform sate evolves.

The primary drawbacks to using the LQR approach, is that the control response of your system is optimal for the linearized system $Ax + Bu$, and not necessarily the true non-linear system. In addition, the control response of K is restricted to the states defined in A. If for example a higher-order derivative or integrating term were desired to be added

to the control policy, the A and B matrices would need to be redefined, and will increase in size by the number of states added to consideration. As the general process for determining a satisfactory LQR controller results in tuning the parameters for Q and R, for a non-linear platform like a quadrotor, this control strategy may produces non-ideal results.

In the case of a cascaded control architecture, another consideration must be made for application of an LQR. The linearized platform dynamics (2.12) show a natural partitioning between the inertial and rotational states about the linearization point. Therefore the LQR control gain K can be split into it's attitude and inertial components and applied on each relevant processor, with each computed result superimposed to produce the final thrust output. However, in the event that the attitude controller is updating at a faster rate than the inertial controller, then the optimality and stability of the LQR policy may no longer be valid. Alternatively, the state and control Jacobians can be computed to produce a purely inertial policy which generates the reference states for a fixed gain-tuned attitude control policy. As this thesis is not focused on control optimality, we will move forward to sub-optimal control techniques.

Velocity PID: Conceptually we can think of an LQR policy derived from the A matrix (2.12) as a PI control on velocity and rotational rate. Therefore it's intuitive that the addition of an inertial acceleration term then extends the controller into a PID on velocity. Addition of the acceleration term allows for the use of more aggressive gains for the LQR control policy, due to the introduction damping on the velocity. Figure 2.3 shows the introduction of an acceleration term onto an unstable LQR control policy.
If we consider setting up an independent inertial control policy,

$$u = -H \begin{bmatrix} K_{\dot{x},i} & 0 & 0 & K_{\dot{x},p} & 0 & 0 & K_{\dot{x},d} & 0 & 0 \\ 0 & K_{\dot{y},i} & 0 & 0 & K_{\dot{y},p} & 0 & 0 & K_{\dot{y},d} & 0 \\ 0 & 0 & K_{\dot{z},i} & 0 & 0 & K_{\dot{z},p} & 0 & 0 & K_{\dot{z},d} \end{bmatrix} \begin{bmatrix} \xi - \xi_d \\ \dot{\xi} - \dot{\xi}_d \\ \ddot{\xi} - \ddot{\xi}_d \end{bmatrix} \tag{2.18}$$

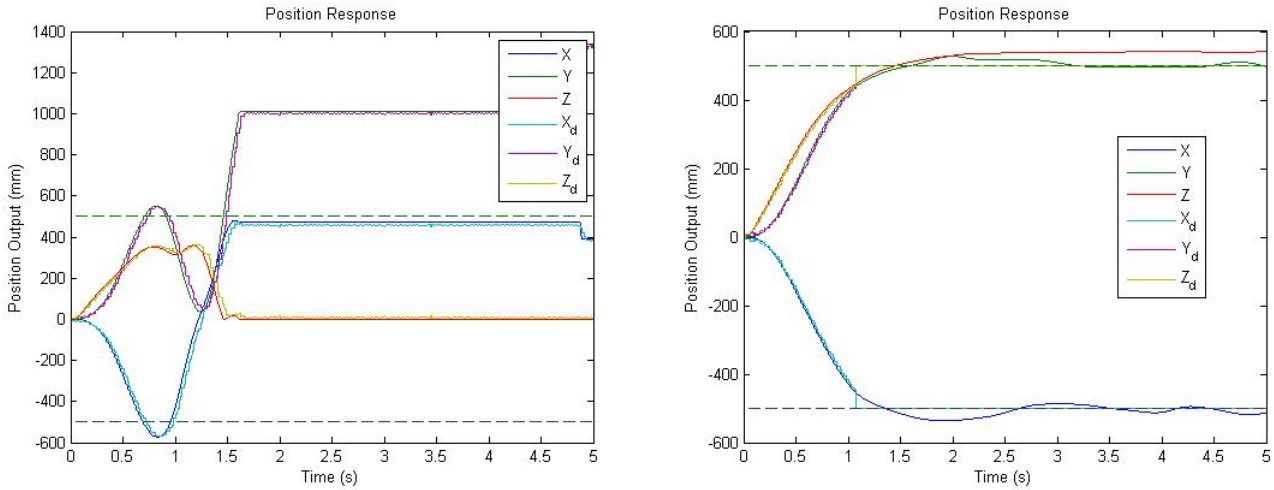

Figure 2.3: Position response of a simulated quadrotor using an LQR control policy, and a velocity trajectory generation scheme. The LQR cost parameters $R = I_{4\times4}$, $Q_{1,2} = 0.01$, and $Q_{7,8} = 0.01$ (all other remaining entries = 1) (Left) Response without acceleration control. System becomes unstable and flips around 1.5 seconds. (Right) LQR response control policy with an acceleration control gain $k_a = 0.01$. Desired acceleration for both policies was zero.

Where H is a matrix that maps the inertial error to the motor thrust when $\psi = 0$,

$$H = \begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

With the acceleration being held near zero during translation, the quadrotor translates at a relatively constant rate from point to point. The primary challenge that this approach introduces is the control policy fighting both increases and decreases in platform velocity, which can manifest as the oscillations in the trajectory.

Z Integral: For the purposes of object pick-up of a payload of unknown mass, the supporting quadrotors will subsequently have their effective mass increased, due to the additional weight. While the proportional control acting on the z-axis will account for some

of this, the error from the desired z set-point will increase as the payload mass increases. To illustrate this, let's consider the simple double-integrator dynamics in the Laplace domain, covered by a velocity PID,

$$G_p = \frac{X(s)}{F(s)} = \frac{1}{ms^2}$$

$$G_c = \frac{F(s)}{E(s)} = k_a s^2 + k_d s + k_p$$

Considering a reference point of $R(s) = 0$, where $E(s) = R(s) - X(s)$, and where a disturbance force is now being added onto our control force,

$$X(s) = G_p(F(s) + F_D(s))$$

How does this influence our error dynamics? We can expand our control transfer function,

$$F(s) = G_c E(s)$$

Where recalling our $E(s) = R(s) - X(s) \implies E(s) = -X(s)$ for $R(s) = 0$,

$$\frac{E(s)}{F_d(s)} = \frac{-G_p}{1 + G_c G_p} = \frac{-1}{(m + k_a)s^2 + k_d s + k_p}$$

Where our steady-state error can be determined through the final-value theorem approach,

$$e_{ss} = \lim_{s \to 0} s \left(\frac{-1}{(m + k_a)s^2 + k_d s + k_p} \right) F_d(s)$$

If we assume a step-input disturbance $F_d(s) = \frac{F_d}{s}$,

$$e_{ss} = \lim_{s \to 0} \left(\frac{-F_d}{(m + k_a)s^2 + k_d s + k_p} \right)$$

Resulting in our steady-state error,

$$e_{ss} = \frac{-F_d}{k_p}$$

which will clearly grow as F_d increases and k_p remains fixed. One simple means to address this is the introduction of a integrator term into the control policy,

$$e_{ss} = \lim_{s \to 0} \left(\frac{-F_d}{(m + k_a)s^2 + k_d s + k_p + \frac{k_i}{s}} \right)$$

Which drives the error for a step disturbance to 0,

$$e_{ss} = \frac{-F_d}{k_p + \infty} = 0$$

Disturbance forces can come from a multitude of other sources, as well, from the reduced thrust due to a decline in total battery voltage as it depletes [3], to the forces neglected by using the idealized model [2]. As such, introduction of an integrator is capable of readily addressing these step disturbances, in the steady-state.

2.1.4 Inertial Trajectory Generation

For the purposes of this thesis, the trajectory generation policies have been broken into three categories: (i) transit, (ii) wind-down, and (iii) position-hold. Table 2.1 tabulates the final trajectory generation policy.

The key concept here is to ensure that the inertial control gains are being used effectively to produce the best response for a given need, with the ultimate goal of taking in a desired final destination, and guiding the platform to that destination both quickly and smoothly. First let's denote the radial distance to some final destination $\xi_f = (x_f, y_f, z_f)^T$ as,

$$r_t = \| \xi_f - \xi(t) \|$$

Transit: For translation, we will define some maximum desired velocity we want the quadrotor to travel at, v_{max}, in the direction of the final destination. The maximum velocity can simply be rotated into the unit-direction of the destination relative to the current position to produce our reference velocity. The desired position reference can then be integrated from the previous set-point using the reference velocity. Together these produce a simple transit policy,

$$\dot{\xi}_d(t) = v_{max} \frac{\xi_f - \xi(t)}{r_t}$$

$$\xi_d(t) = \xi_d(t-1) + dt\dot{\xi}_d(t)$$

These equations represent the ideal velocity and position set-points. In reality, disturbances and model uncertainty will cause the actual platform dynamics to deviate from the idealized system. We can instead leverage the integrating capabilities of the position control policy by using the relative velocity to combat these disturbances. To better understand how this approach works, we take our ideal trajectory and solve for the case where $\xi(t-1) = \xi_d(t-1)$. Taking the backwards approximation of our true inertial dynamics,

$$\xi(t-1) = \xi(t) - dt\dot{\xi}(t)$$

Which can readily be substituted into our ideal trajectory policy to produce our state-dependent position tracking policy,

$$\xi_d(t) = \xi(t) + dt(\dot{\xi}(t) - \dot{\xi}_d(t))$$

The core concept here is that if the quadrotor is already tracking at the desired velocity, then the current desired position will exactly match the current quadrotor position. The advantage of using the current position information is it allows us to restrict the trajectory policy from calling a value beyond a certain range from the current quadrotors state, in particular after a disturbance, which may push it into an unstable response set. Therefore our running transit policy becomes,

$$(r_t > r_{wd}) :$$
$$\dot{\xi}_d(t) = v_{max}\frac{\xi_f - \xi(t)}{r_t}$$
$$\xi_d(t) = \xi(t) + dt(\dot{\xi}_d(t) - \dot{\xi}(t))$$

Where r_{wd} is the start of the wind-down region. This is graphically depicted in figure 2.4 as (i).

Wind-down: As the quadrotor approaches the final destination, the velocity of the platform needs to come to rest. However, if this is done by simply setting the reference velocity to 0, then the resulting discontinuity of the velocity response can result in a abrupt

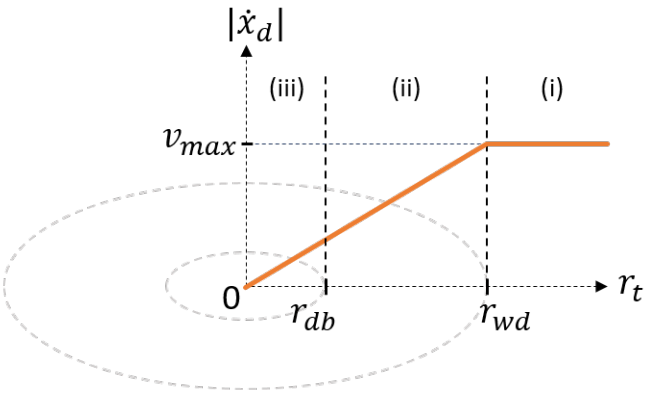

Figure 2.4: Graphical representation of the desired velocity set-point relative to a final destination. It is broken into three phases: (i) transit, (ii) wind-down, and (iii) position-hold.

stop, overshoot, and/or oscillation. Additionally, if a dead-band is used and the quadrotor is perturbed from the destination, it falls upon the position controller alone to combat disturbances. Instead we can define a region around the destination where the reference velocity linearly decreases to 0 as a function of the norm distance to the destination. We can define this wind-down region can simply a norm band $0 \leq r_t \leq r_{wd}$, where the function is active. For this policy, we can design a simple linear function who's origin is placed at the destination,

$$\dot{r}_d(t) = \frac{v_{max}}{r_{wd}} r_t$$

Which can be re-directed into our 3D coordinates as our wind-down trajectory policy,

$$(0 \leq r_t \leq r_{wd}) :$$
$$\dot{\xi}_d(t) = \frac{v_{max}}{r_{wd}} \frac{\xi_f - \xi(t)}{r_t}$$
$$\xi_d(t) = \xi(t) + dt(\dot{\xi}_d(t) - \dot{\xi}(t))$$

This is graphically depicted in figure 2.4 as (ii).

Position-hold: Finally when the quadrotor is within norm distance, $r_t < r_{db}$ from the final destination, then the position is no longer integrated, and is simply set as the final

destination, in order to re-prioritize the position controller for disturbance rejection from the destination as opposed to the desired velocity,

$$(0 \leq r_t < r_{db}) :$$
$$\dot{\xi}_d(t) = \frac{v_{max}}{r_{wd}} \frac{\xi_f - \xi(t)}{r_t}$$
$$\xi_d(t) = \xi_f$$

This is graphically depicted in figure 2.4 as (iii).

Acceleration: Depending on the application, the acceleration controller can be leveraged to dampen the system response, or help drive the velocity from one set-point to another. For the purposes of this thesis, we will default to the damping functionality by setting a persistent zero reference acceleration,

$$\ddot{\xi}_d(t) = 0$$

Final Velocity and Position Protocol

Transit	**for:** $(r_t > r_{wd})$	$\dot{\xi}_d(t) = v_{max} \frac{\xi_f - \xi(t)}{r_t}$ $\xi_d(t) = \xi(t) + dt(\dot{\xi}_d(t) - \dot{\xi}(t))$
Wind-down	**for:** $(r_{db} \leq r_t \leq r_{wd})$	$\dot{\xi}_d(t) = \frac{v_{max}}{r_{wd}} \frac{\xi_f - \xi(t)}{r_t}$ $\xi_d(t) = \xi(t) + dt(\dot{\xi}_d(t) - \dot{\xi}(t))$
Position Hold	**for:** $(0 \leq r_t < r_{db})$	$\dot{\xi}_d(t) = \frac{v_{max}}{r_{wd}} \frac{\xi_f - \xi(t)}{r_t}$ $\xi_d(t) = \xi_f$

Final Acceleration Protocol

for: $(\forall r_t)$	$\ddot{\xi}_d(t) = 0$

Table 2.1: Tabulated position, velocity, and acceleration trajectory generation policy for various ranges of $r_t = \| \xi_f - \xi \|$, where $\xi = (x, y, z)^T$. The fundamental goal is to generate a continuously viable trajectory towards a global destination $\xi_f = (x_f, y_f, z_f)^T$, from the current location ξ.

2.1.5 Attitude Trajectory Generation

Many of the commercially available quadrotors described in chapter 1 utilize a cascade control architecture. This architecture helps to ensure that the faster attitude dynamics remain able to quickly react to disturbances while executing computationally intensive tasks on an a higher-level processor. Regardless of how the position control response is calculated, the control input inevitably needs to be converted to attitude reference set-points to be enforced by the attitude controller. We acknowledge it is possible to use a zero-reference attitude controller, which essentially seeks to maintain a hover state that is disturbed by the inertial control response. However, such disturbances would need to be sufficiently large to over-power the attitude stabilization in order to achieve a desired response, and could instead be accomplished through leveraging the attitude controller itself. This section outlines a few approaches available to convert the inertial control inputs into the relevant attitude reference values.

Integration of Torque-induced Angular Acceleration Approach: If an LQR or similar inertial control policy is used to generate motor thrust inputs, the inertial policy would be of the form,

$$u = -K(X - X_d) = \{u_1, u_2, u_3, u_4\} \in \mathbb{R}^4$$

These motor values can be used to determine their induced torque on the body-frame. As the motors are affixed to the body-frame itself, the motor-thrust relationship to the induced torque can readily be determined. An approximation of the rotational inertial properties can then be used to determine the angular acceleration of the body-frame. The value of this approach is that the LQR policy developed using (2.12) and (2.14) can be used by simply setting the attitude entries of the control gain or error vector to zero.

To determine our conversion, recall our rotational kinematics (2.8) (2.9) and consider the case where the yaw is under control. If the yaw rate $\dot{\psi} \approx 0$, the dynamical relationships for

the pitch and roll axes simplify to,

$$\ddot{\phi}_d = \frac{1}{I_{xx}}(d(u_1 - u_2 + u_3 - u_4)) \tag{2.19}$$

$$\ddot{\theta}_d = \frac{1}{I_{yy}}(d(-u_1 - u_2 + u_3 + u_4)) \tag{2.20}$$

Which is simply the mapping of the control thrusts generated from the inertial control policy to the thrust and angular acceleration,

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ F \end{bmatrix} = \begin{bmatrix} \frac{d}{I_{xx}} & 0 & 0 \\ 0 & \frac{d}{I_{yy}} & 0 \\ 0 & 0 & 1 \end{bmatrix} H^T u \tag{2.21}$$

For the sake of simplicity, let's consider a simple PD controller for attitude stabilization, which also happens to mimic the LQR control policy for the current A and B matrices. If $e(t) = (\varphi(t) - \varphi_d(t))$ and $\dot{e}(t) = (\dot{\varphi}(t) - \dot{\varphi}_d(t))$, the protocol can be written in the form,

$$u_{rot} = -(K_p e(t) + K_d \dot{e}(t))$$

Since we know a stable PD protocol will drive the error term towards zero, introducing attitude trajectory set-points will simply shift the origin to the reference.

We can now examine how introducing a rate set-point affects the control. Since (2.19) and (2.20) provide the angular acceleration, an instinctive approach would be to integrate an approximate set-point using a forward euler approximation,

$$\dot{\varphi}_d(t) = \dot{\varphi}_d(t-1) + dt(\ddot{\varphi}_d(t))$$

$$\varphi_d(t) = \varphi_d(t-1) + dt(\dot{\varphi}_d(t))$$

However this results in an unstable response, particularly the attitude rate control. Instead using the desired change in angular rate from the angular acceleration over the descretization interval as the desired set-points, produces stable tracking,

$$\dot{\varphi}_d(t) = dt(\ddot{\varphi}_d(t)) \tag{2.22}$$

$$\varphi_d(t) = \varphi_d(t-1) + dt(\dot{\varphi}_d(t)) \tag{2.23}$$

The impact of the reference becomes more apparent if we break the control response into separate hover-maintenance and trajectory tracking components,

$$u_{rot} = -K_p(\varphi(t) - \varphi_d(t)) - K_d(\dot{\varphi}(t) - \dot{\varphi}_d(t))$$

Substituting in our reference trajectories,

$$u_{rot} = -K_p(\varphi(t) - (\varphi_d(t-1) + dt\dot{\varphi}_d(t))) - K_d(\dot{\varphi}(t) - dt(\ddot{\varphi}_d(t))$$

$$u_{rot} = -K_p(\varphi(t) - \varphi_d(t-1) - dt\dot{\varphi}_d(t)) - K_d(\dot{\varphi}(t) - dt(\ddot{\varphi}_d(t))$$

If we assume that the proportional regulation has been tracking the reference angle then we know that the current angle will be approximately equal to the previous set-point ($\varphi(t) \approx \varphi_d(t-1)$),

$$u_{rot} = -K_p(-dt\dot{\varphi}_d(t)) - K_d(\dot{\varphi}(t) - dt(\ddot{\varphi}_d(t))$$

$$u_{rot} = -K_d\dot{\varphi}(t) + dt(K_p\dot{\varphi}_d(t) + K_d\ddot{\varphi}_d(t))$$

Which essentially results in a damping response on the rotational rate, with an applied disturbance torque that moves the platform from it's hover equilibrium towards a desired inertial set-point.

Alternatively the desired angle can be integrated from the current angle using the desired reference rate,

$$\varphi_d(t) = \varphi(t) + dt(\dot{\varphi}_d(t))$$

The advantage of this approach is that noise corrupting the attitude angle estimation has a reduced effect on the tracked trajectory, as the angle controller is only reacting to the error from the desired rate being applied over a time step. The challenge with a response driven by a reference angular rate, would be a potential bias in the final angle. However, as we derived for quadrotors in equation (2.6), any angular bias would result in a portion of the thrust vector being diverted into the x-y plane. Any undesired movement in the x-y plane

would then result in control response, and thereby a corrective rate set-point. As the inertial controller is stabilizing about a stationary hover about some destination, then the system will naturally correct any angular bias in order to reach equilibrium.

Small-angle Approximation Approach: Another approach to determining the attitude references from desired inertial acceleration values, is using the small angle approximation. If we take our acceleration dynamics from equation (2.6), and apply the small angle approximation ($\cos(\varphi) \approx 1$ and $\sin(\varphi) \approx \varphi$),

$$\ddot{x} = \frac{u_{tot}}{m}(\sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi) \approx \frac{u_{tot}}{m}(\phi\sin\psi + \theta\cos\psi)$$
$$\ddot{y} = \frac{u_{tot}}{m}(-\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi)) \approx \frac{u_{tot}}{m}(\theta\sin\psi - \phi\cos\psi)$$
$$\ddot{z} = \frac{u_{tot}}{m}(\cos\theta\cos\phi) - g \approx \frac{u_{tot}}{m} - g$$

Where the yaw angle, ψ is not considered under the small angle approximation, as it is not necessarily being held near zero by the attitude control while hovering. It is also expected that the z acceleration would be independent of the yaw angle, as rotation about the z-axis would not impact the thrust in that direction under the small angle approximation for θ and ϕ. Simplifying and introducing some placeholder variables for notational simplicity, $a = \frac{m\ddot{x}}{u_{tot}}$ and $b = \frac{m\ddot{y}}{u_{tot}}$,

$$a = \phi\sin\psi + \theta\cos\psi \tag{2.24}$$

$$b = \theta\sin\psi - \phi\cos\psi \tag{2.25}$$

Solving 2.25 for θ,

$$\theta = \frac{b + \phi\cos\psi}{\sin\psi} \tag{2.26}$$

Substituting 2.26 into 2.24 and solving for ϕ,

$$a = \phi\sin\psi + \frac{b + \phi\cos\psi}{\sin\psi}\cos\psi$$

$$a\sin\psi = \phi\sin^2\psi + b\cos\psi + \phi\cos^2\psi$$

$$a\sin\psi = \phi + b\cos\psi$$

Resulting in a solution,

$$\phi = a\sin\psi - b\cos\psi \tag{2.27}$$

Substituting 2.27 into 2.26,

$$\theta = \frac{b + (a\sin\psi - b\cos\psi)\cos\psi}{\sin\psi}$$

$$\theta = \frac{b(1-\cos^2\psi) + a\sin\psi\cos\psi)}{\sin\psi}$$

$$\theta = \frac{b\sin^2\psi + a\sin\psi\cos\psi)}{\sin\psi}$$

Resulting in the solution,

$$\theta = a\cos\psi + b\sin\psi \tag{2.28}$$

Replacing our placeholder variables a and b yields our final solutions for the angular reference values,

$$\theta = \frac{m}{u_{tot}}(\ddot{x}\cos\psi + \ddot{y}\sin\psi) \tag{2.29}$$

$$\phi = \frac{m}{u_{tot}}(\ddot{x}\sin\psi - \ddot{y}\cos\psi) \tag{2.30}$$

This can also be condensed into matrix form,

$$\begin{bmatrix}\theta\\ \phi\end{bmatrix} = \frac{m}{u_{tot}}\begin{bmatrix}\cos(\psi) & \sin(\psi)\\ \sin(\psi) & -\cos(\psi)\end{bmatrix}\begin{bmatrix}\ddot{x}\\ \ddot{y}\end{bmatrix} \tag{2.31}$$

Where the thrust is easily determined by,

$$u_{tot} = m(\ddot{z} + g) \tag{2.32}$$

Motor Thrust Bias: Along with the reference attitude trajectory, a motor thrust bias must be provided to set the common operating point for the motor thrust. Recalling that $F = u^T\mathbf{1}$, the desired thrust bias for each motor simply becomes the average of the computed inertial thrust values,

$$u_{th} = \frac{1}{4}\sum_{i=0}^{4} u_i \tag{2.33}$$

This biasing term is added on top of the rotational control values for each motor, allowing for the system to provide inertial thrust while providing the imbalance necessary to apply corrective torques.

2.2 Payload

2.2.1 Eccentric Loading on Quadrotor

A common simplifying assumption made when modeling a payload supported by a quadrotor, is that the payload is directly attached to the quadrotor's center of mass [15, 23]. For larger quadrotors this assumption is appropriate as the high thrust and moment advantage offered by the larger arm span allows for the attitude controller to account for much of torque induced by the eccentric loading. However, as the platform being developed for the RAIN lab is intended to be much smaller, the eccentric loading becomes much more relevant to platform stability. Figure 2.5 illustrates the eccentric loading condition, which can be modeled as an induced disturbance torque acting on the body frame. This torque then is introduced into the quadrotor rotational dynamics (2.7),

$$I\dot{\omega} = -\omega \times I\omega + T + T_{ecc} \tag{2.34}$$

The expression for T_{ecc} is defined in the following sections, as it varies based off the loading configuration assumptions.

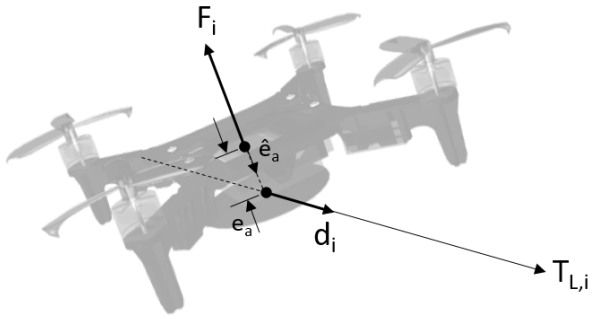

Figure 2.5: Diagram of the quadrotor supporting a tensile load mounted eccentrically from the platform's center of mass at some distance e_a.

2.2.2 Cable-suspended Single-anchor System

The dynamics of the point-mass system depicted in figure 2.6 for n agents can be described by a simple Newtonian system [15, 23],

$$m_L \ddot{\xi}_L = -\sum^{n} (T_{L,i} d_i) - m_L g e_3 \tag{2.35}$$

Where $T_{L,i}$ is the tension in the i-th cable, d_i is the unit vector pointing from the i-th quadrotor anchor to the common anchor point on the supported payload, and m_L is the payload mass. For the sake of this thesis, the drag forces acting on the payload are considered negligible, cables are attached to the payload center of mass, and the rotational dynamics are negligible. As the supporting cables are only able to provide a tensile support we will need to enforce the condition,

$$T_{L,i} = \begin{Bmatrix} T_{L,i}, & if: \ T_{L,i} > 0 \\ 0, & if: \ T_{L,i} \leq 0 \end{Bmatrix}$$

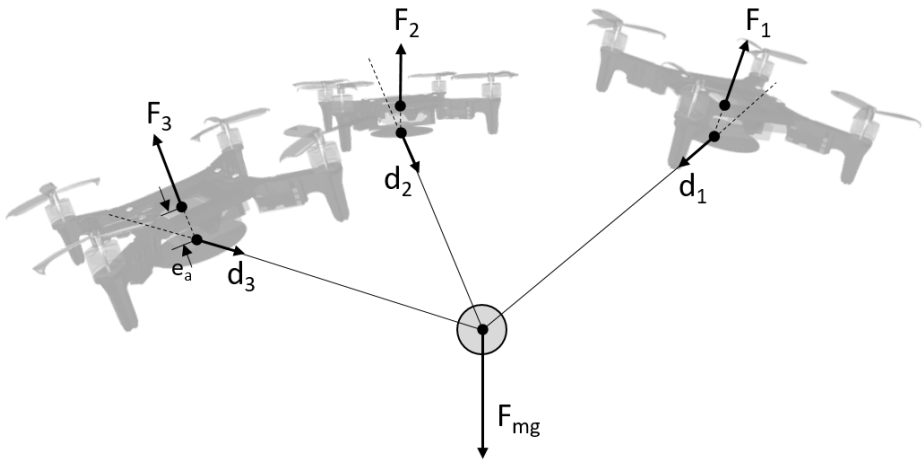

Figure 2.6: Cable-suspended payload transportation using a single point of attachment. This diagram assumes that the payload is attached by an inelastic cable fixed to the center of mass of a given payload.

The eccentric torque induced by the point-mass is directly resulting from the tension acting on the payload,

$$T_{ecc} = e_a \hat{e}_a \times \left(R^T T_{L,i} d_i \right) \tag{2.36}$$

Where e_a is the magnitude of the mounting offset, $\hat{e}_a$ is the unit vector pointing from the quadrotor center of mass to the anchor point relative to the body frame, d_i is the vector pointing from the i-th quadrotor anchor to the corresponding payload anchor point in inertial-frame coordinates, and $T_{L,i}$ is the magnitude of the cable tension transferred from the supported payload. If we assume that the payload is mounded directly below the quadrotor center of mass, $\hat{e}_a = [0, 0, -1]^T$.

2.2.3 Cable-suspended Rigid Body Multi-anchor System

The dynamics of a multi-anchor payload are an extension of the point-mass payload dynamics (2.35), with an array of anchors distributed accross a rigid-body frame. Distributed support points requires an understanding of the orientation of the rigid body

they're attached to, and thereby requires the introduction of the payload rotational dynamics [23],

$$I_L\dot{\omega}_L = -\omega_L \times I_L\omega_L + R_L^T \sum^{n} (q_i \times -T_{L,i}d_i) \tag{2.37}$$

where I_L is the rotational inertial tensor, ω_L is the rotational rate about the principal axes of the body frame, and q_i is the vector pointing from the rigid body's center of mass to the anchor for the i-th quadrotor,

$$q_i = R_L q_{i,LB}$$

Where R_L is the rotation matrix from the body frame of the payload into the inertial frame shared by the quadrotors and the payload dynamics, and will be of the same R_{zyx} form as (2.2) used in the quadrotor rotational dynamics derivation. $q_{i,LB}$ denotes the vector from the center of mass of the payload to the i-the anchor with respect to the payload body-frame.

The inertial tensor will be assumed to be a principal inertia matrix,

$$I_L = \begin{bmatrix} I_{L,xx} & 0 & 0 \\ 0 & I_{L,yy} & 0 \\ 0 & 0 & I_{L,zz} \end{bmatrix}$$

Additionally as the orientation now influences the tension vector direction, the orientation of the payload also needs to be accounted for in the payload's inertial dynamics,

$$m_L\ddot{\xi}_L = -\sum^{n} (T_{L,i}d_i) - m_L g e_3 \tag{2.38}$$

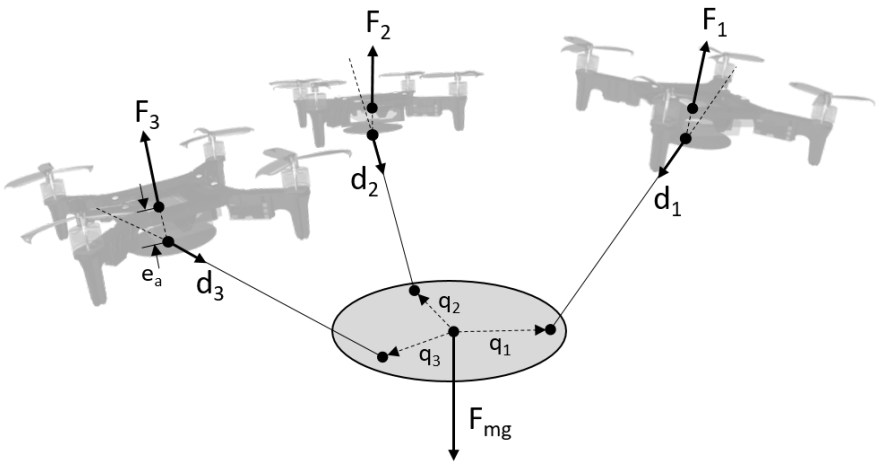

Figure 2.7: Cable-suspended payload transportation using unique points of attachment for each agent. This diagram assumes that the payload is attached by an inelastic cable.

As opposed to the point-mass eccentric loading (2.36), the eccentric torque induced by the multi-anchor rigid body is dependent on the orientation of the payload itself, and thus the tension vectors need to be rotated into the inertial frame before they can be applied to the quadrotor inertial dynamics,

$$T_{ecc} = e_a \hat{e}_a \times \left(R^T T_{L,i} d_i \right) \tag{2.39}$$

2.3 Formation Control

2.3.1 Network Formation Consensus

Consensus is an attractive means of formation control across a swarm of agents for its simplicity and low computational cost. While there is much room to explore the impact of various graph configurations across large swarms of agents, this thesis will focus on a fully-connected graph. For a swarm of n agents, consensus on some variable x for agent i

becomes [17],

$$x_i^+ = x_i + \frac{1}{n}\sum_{i\sim j}^{n}(x_j - x_i) \tag{2.40}$$

Where x_i^+ represents the updated value of the x state for the i-th agent, and $i \sim j$ denotes an edge existing between agent i and j (where $i \neq j$). Figure 2.8 graphically illustrates this protocol for a 2-D case, across a fully connected graph of $n = 4$ agents.

In the simplest sense, a formation consensus protocol then seeks to maintain some bias between its own measurement and the neighboring agents. In the 1-D case, this can simply be achieved introducing some bias b applied along the direction of the difference away from the agent,

$$x_i^+ = x_i + \frac{1}{n}\sum_{i\sim j}^{n}\left(x_j - x_i - b\frac{x_j - x_i}{\parallel x_j - x_i \parallel}\right)$$

Which can be simplified to,

$$x_i^+ = x_i + \frac{1}{n}\sum_{i\sim j}^{n}(\parallel x_j - x_i \parallel - b)\frac{x_j - x_i}{\parallel x_j - x_i \parallel}$$

Now supposing that this bias is a 2-D radial distance between agents, then we can adjust the bias to be applied to a radial distance between the agents, with the radius equal to that of the norm difference between the two,

$$r_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

$$c_i = \frac{1}{n}\sum_{i\sim j}^{n}(r_{ij} - b) \tag{2.41}$$

Figure 2.8 shows a graphical representation of a consensus across radial difference. Which can then be placed into terms of x and y by multiplying the component of the unit vector of that difference,

$$x_i^+ = x_i + c_i\frac{r_{ij,x}}{r_{ij}} = x_i + \frac{1}{n}\sum_{i\sim j}^{n}(r_{ij} - b)\frac{(x_j - x_i)}{r_{ij}}$$

$$y_i^+ = y_i + c_i\frac{r_{ij,y}}{r_{ij}} = y_i + \frac{1}{n}\sum_{i\sim j}^{n}(r_{ij} - b)\frac{(y_j - y_i)}{r_{ij}}$$

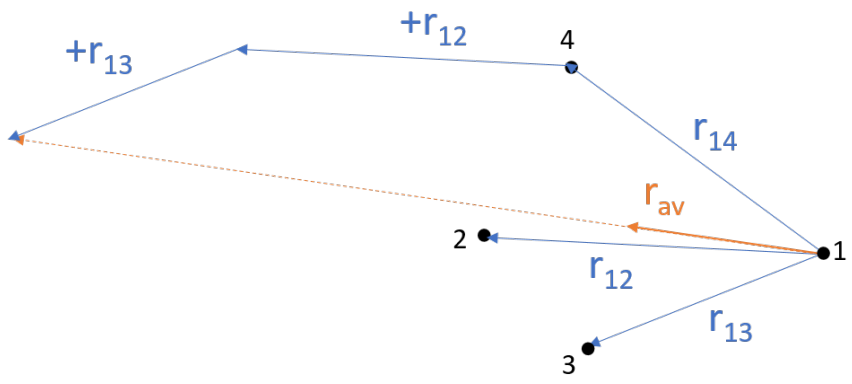

Figure 2.8: A vector representation of consensus over a formation of four agents, from the perspective of agent 1. Where $r_{ij} = r_j - r_i$ measured from a common global reference point.

Which simplifies to the 2-D formation protocol,

$$x_i^+ = x_i + \frac{1}{n}\sum_{i\sim j}^{n}\left(1 - \frac{b}{r_{ij}}\right)(x_j - x_i) \tag{2.42}$$

$$y_i^+ = y_i + \frac{1}{n}\sum_{i\sim j}^{n}\left(1 - \frac{b}{r_{ij}}\right)(y_j - y_i) \tag{2.43}$$

$$r_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

Figure 2.9 shows this formation protocol acting on a 2-D system.

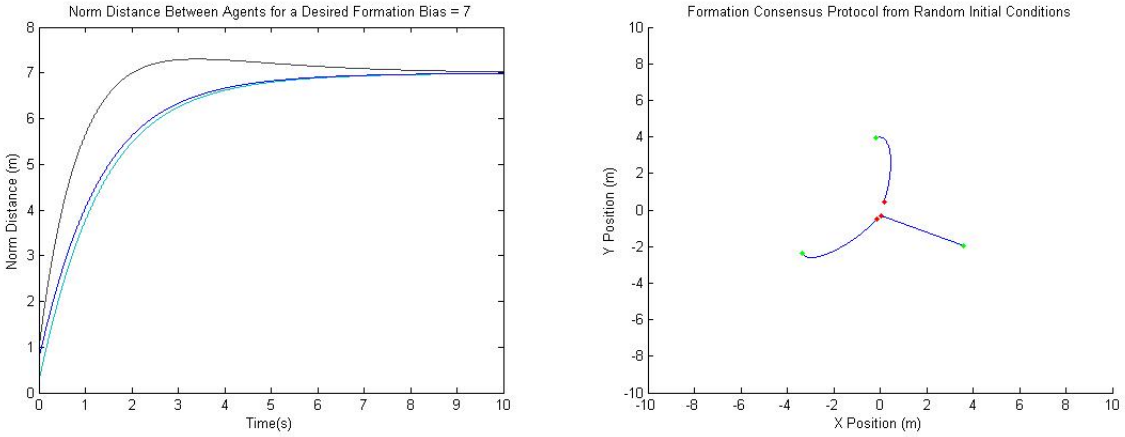

Figure 2.9: Plot of the 2-D formation consensus protocol depicted in this section, for a single-integrator system for $b = 7$. A single integrator was used to show the progression of convergence in time. (Left) Is the norm-distance between each agent, with the desired distance being 7m. (Right) A plot of 2-D formation protocol moving from a random initial condition (red) to the desired formation bias (green).

2.3.2 Unit-vector Consensus

If we consider the 1-D consensus algorithm (2.40) acting on each axis in the 2-D x-y plane, as illustrated in figure 2.8, it becomes clear that the consensus sum itself is the vector pointing from a given agent, to the network average. In the case of position, this average would correspond to the average position, and therefore the consensus sum corresponds to the vector distance from a given agent to the formation average,

$$x_{av} - x_i = \frac{1}{n} \sum_{i \sim j}^{n} (x_j - x_i)$$

Suppose now we consider the average unit-direction from a given agent in 2-D to their neighbors,

$$x_i^+ = \frac{1}{n} \sum_{i \sim j}^{n} \left(\frac{x_j - x_i}{r_{ij}} \right)$$

$$y_i^+ = \frac{1}{n} \sum_{i \sim j}^{n} \left(\frac{y_j - y_i}{r_{ij}} \right)$$

$$r_{ij} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

This would correspond to the consensus of the unit vectors for formation itself, and can be thought of as pointing towards the average direction of each agent relative to agent i. This is not the same as the vector pointing towards the average, as the magnitude of the vector is not taken into account. The final unit vector can then be used to define an agent-specific destination for inertial trajectory generation around some common destination,

$$x_{f,i} = x_f - r_{d,i}\frac{x_i^+}{r_i^+}, \qquad y_{f,i} = y_f - r_{d,i}\frac{y_i^+}{r_i^+} \tag{2.44}$$

$$r_i^+ = \sqrt{(x_i^+)^2 + (y_i^+)^2}$$

Where $r_{d,i}$ is the desired radial distance that the quadrotor is to maintain around the destination x_f, y_f, and introduction of the negative sign reorients the unit vector. The negative sign in front of the adjustment term serves to drive the agents from one another, while being restricted to the unit circle. Note that the z axis is not accounted for in this particular protocol, as we want the quadrotors to operate in the same plane for the purposes of this thesis.

The advantage of using this approach is that the agents will naturally distribute themselves around a unit circle, with an additional degree of freedom for payload manipulation in the addition of a radial distance input $r_{d,i}$, which can be agent specific. Figure 2.10 demonstrates this formation protocol across $n = 3$ and $n = 30$ agents.

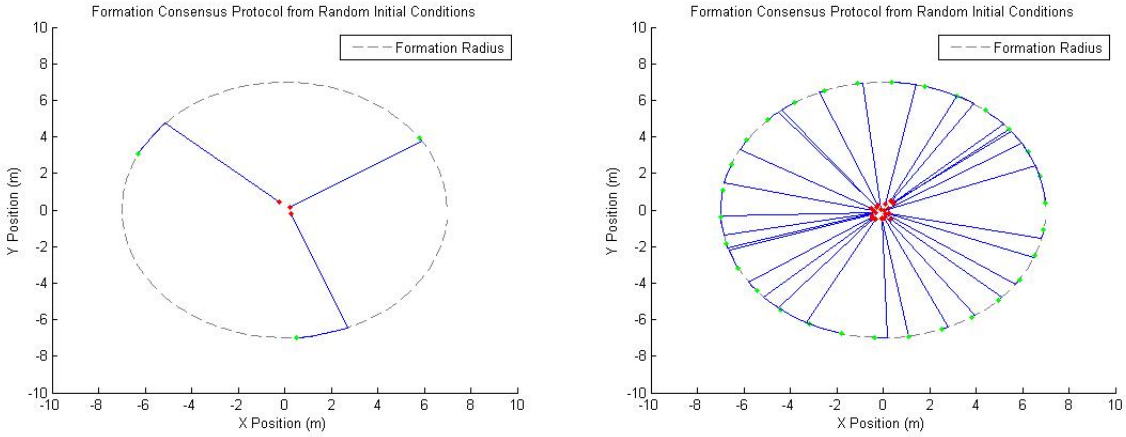

Figure 2.10: Plot of the 2-D unit-vector formation consensus protocol depicted in this section, for a desired formation radius of $r_{d,i} = 7$, where formation protocol moving from a random initial condition (red) to the desired formation bias (green). (Left) For $n = 3$ the position shifts immediately from the initial position and then tracks along the circle of radius $r_{d,i}$ as each agent approaches their final position, and the agents distribute evenly around the unit-circle. (Right) For $n = 30$ (and higher numbers of agents in general).

Formation Angle Control: One interesting aspect of using a unit-vector consensus approach, is that the formation will automatically distribute the agents around a unit-circle evenly from one another. If we wanted to control the formation orientation, we can simply have one agent ignore the destinations of the other agents, and place itself at a formation point,

$$x_d = k_{d,i} \cos(\psi_{form})$$

$$y_d = k_{d,i} \sin(\psi_{form})$$

where ψ_{form} is the desired orientation of the formation (this can be thought of a as a global pointing angle). Figure 2.11 shows the formation orientation control in action for $n = 3$ and $n = 10$ agents, where agent 1 is setting the formation pointing angle, $\psi_{form} = 45^o$.

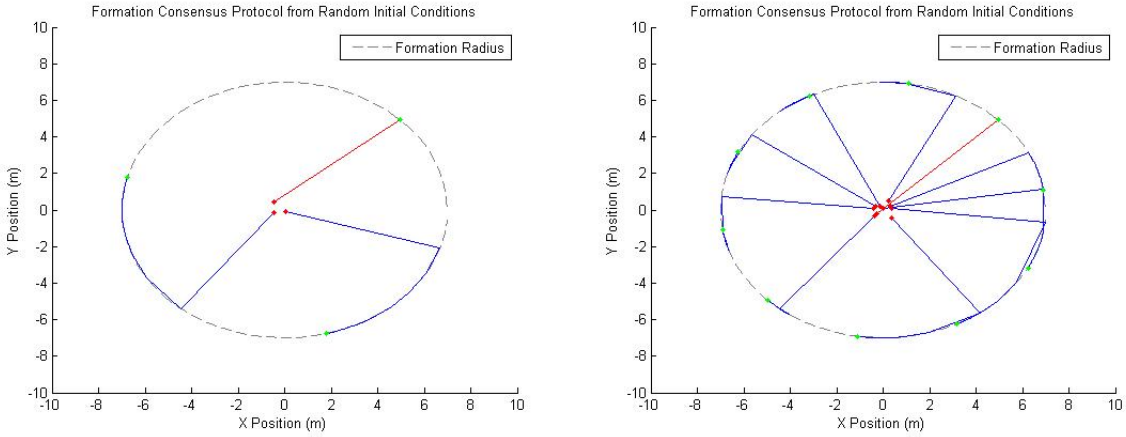

Figure 2.11: Plot of the 2-D unit-vector formation consensus protocol depicted in this section, for a desired formation radius of $r_{d,i} = 7$, where agent 1 is going to a fixed point rotated by $\psi_{form} = 45^o$ from the global reference axis. (Left) For $n = 3$, all agents move from agent 1 (red line) distributing evenly around the unit circle. (Right) For $n = 10$ (and higher numbers of agents in general).

2.3.3 Combined Formation and Unit-vector Consensus with Damping

While the unit-vector consensus approach serves as a good means for formation position placement around a destination, the primary purpose of the protocol as defined by (2.44) is to provide a destination for the global trajectory protocol to approach. This leaves the issue of formation maintenance, and specifically in-transit formation cohesion to be addressed. Ultimately we want the cohesion protocol to define virtual rigid linkages between agents that retain the formation shape, while still being compatible with the inertial trajectory protocol.

One of the outputs of the trajectory protocol is a desired velocity set-point. To enforce our formation control, we can simply redirect that desired velocity by summing the cohesive control response with the desired velocity vector. Therefore we define a cohesive control response, c_i that uses the formation consensus protocol (2.41) as a starting point,

$$c_i = \frac{1}{n} \sum_{i \sim j}^{n} (r_{ij} - b)$$

From here, this protocol can be thought of as a proportional controller with respect to the radial distance between to agents at a discrete time-step k with a proportional gain $K_{p,f}$,

$$c_i(k) = \frac{1}{n}\sum_{i\sim j}^{n} K_{p,f}\left(r_{ij}(k) - b\right)$$

As the platform has inherently non-linear dynamics, it would be beneficial to have a damping term present to dampen out any oscillations incurred from the interactions of the formation [6, 29]. Damping is accomplished by taking the backward Euler approximation of the radial distance change and some derivative gain $K_{d,f}$,

$$c_i(k) = \frac{1}{n}\sum_{i\sim j}^{n}\left(K_{p,f}\left(r_{ij}(k) - b\right) + K_{d,f}\frac{\left(r_{ij}(k) - r_{ij}(k-1)\right)}{dt}\right) \tag{2.45}$$

where in this case, dt is the difference between each of the discrete time-steps k.

Redirecting the radial protocol into the x-y plane,

$$C_i(k) = \begin{bmatrix} c_{i,x}(k) \\ c_{i,y}(k) \\ c_{i,z}(k) \end{bmatrix} = \frac{1}{n}\sum_{i\sim j}^{n}\left(\left(K_{p,f}\left(r_{ij}(k) - b\right) + K_{d,f}\frac{\left(r_{ij}(k) - r_{ij}(k-1)\right)}{dt}\right)\frac{1}{r_{ij}}\begin{bmatrix} x_j - x_i \\ y_j - y_i \\ 0 \end{bmatrix}\right)$$

Figure 2.12 shows a comparison for a 2-D single-integrator system.

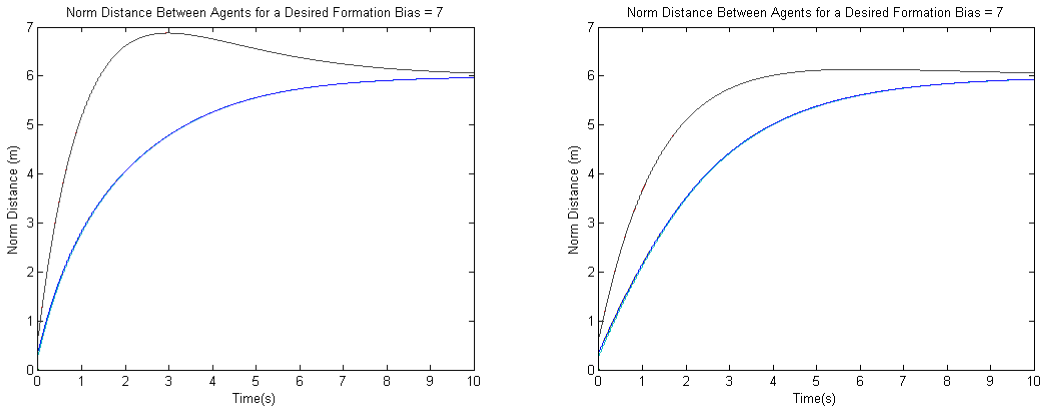

Figure 2.12: Plots of the inter-agent norm distance of the PD consensus protocol outlined in this section, executed in 2-D. (Left) Protocol using $K_{p,f} = 1$ and $K_{d,f} = 0$. (Right) Protocol using $K_{p,f} = 1$ and $K_{d,f} = 0.75$. The addition of the damping term results in a reduction of overshoot.

We then introduce this into control response into the platform dynamics by redirecting the current x-y velocity set-point,

$$\dot{\xi}'_d(t) = \dot{\xi}_d(t) + C_i(k) \tag{2.46}$$

For safety and feasibility, we want to ensure the quadrotor response does not exceed the desired maximum velocity used by the trajectory protocol,

$$\dot{\xi}'_d(t) = \left\{ \begin{array}{cc} v_{max} \frac{\dot{\xi}'_d(t)}{\|\dot{\xi}'_d(t)\|}, & if \quad \| \dot{\xi}'_d(t) \| > v_{max} \\ \dot{\xi}'_d(t), & else \end{array} \right\} \tag{2.47}$$

If the quadrotor uses the unit-vector approach outline in equation (2.44) for position placement, the desired distance between each agent needs to be determined, to have the correct inter-agent buffer b. Consider figure 2.13, which shows the $n = 3$ formation, where we're seeking to determine the desired buffer distance between agents 1 and 2, which are at different radial distances from the formation destination.

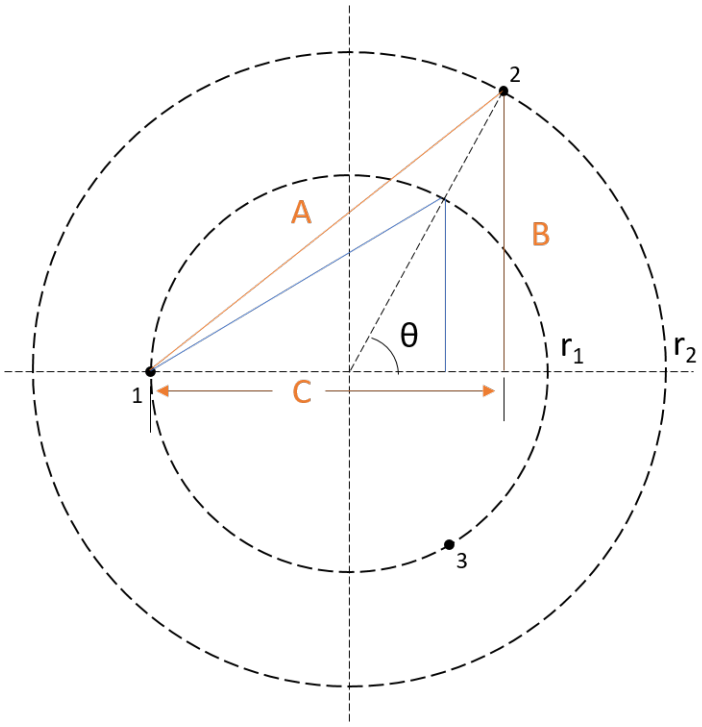

Figure 2.13: Diagram of $n = 3$ agents, where agent 2 is at a larger radial distance than agents 1 and 2. Since agent 2 has only increased in radial distance, it shares a common angle θ with the un-scaled radial distance.

We can then recognize that

$$B = r_2 \sin(\theta)$$

$$C = r_1 + r_2 \cos(\theta)$$

Then leveraging the Pythagorean theorem,

$$A = \sqrt{B^2 + C^2}$$

$$A = \sqrt{\left(r_2 \sin(\theta)\right)^2 + \left(r_1 + r_2 \cos(\theta)\right)^2}$$

$$A = \sqrt{r_2^2 sin^2(\theta) + r_1^2 + 2r_1 r_2 \cos(\theta) + r_2^2 cos^2(\theta)}$$

Which can be simplified to,

$$A = \sqrt{r_2^2 + r_1^2 + 2r_1 r_2 \cos(\theta)}$$

We can then recall that unit-vector consensus will distribute the agents evenly around a unit circle. As such, with respect to Agent 1,

$$\theta = 180^o - \frac{360^o}{3} = 60^o$$

It is important to note that this only works for $n \leq 3$. Once $n > 3$, the ordering of the agents around the unit circle becomes relevant. While there are a number of ways to approach this, the scope of this thesis is interested in a formation of $n = 3$ agents. This results in the desired inter-agent buffering of,

$$b_{ij} = \sqrt{r_i^2 + r_j^2 + 2r_i r_j \cos(\theta_{ij})} \tag{2.48}$$

Where for $n \leq 3$,

$$b_{ij} = \sqrt{r_i^2 + r_j^2 + 2r_i r_j cos\left(180^o - \frac{360^o}{n}\right)} \tag{2.49}$$

Chapter 3
SIMULATION DESIGN AND SETUP

As part of the hardware development process, a simulation suite was produced in parallel to help test and tune new control and trajectory algorithms. Discussed in this chapter are many of the design choices and assumptions made during the simulation development process.

3.1 Quadrotor

Data Segregation: One challenge that is presented when simulating a live system, is feasible data accessibility. More specifically, the simulated agent may not have access to all of the data that is required to produce a high-fidelity simulation. A simple example of this is the current state information. In order to properly approximate the non-linear dynamics, the simulated true state information is propagated to produce the response of the system; however, the simulated platform only has access to the information made available through emulated sensors. These sensors have a limited buffer for storage, can add noise, and can only be used to detect specific states. The remaining states may need to be filled using estimation algorithms.

Figure 3.1 shows the data partitioning used by the final quadrotor simulation suite. The simulator contains two primary data structures, one for the simulation-specific data, which may not be accessible by the simulated agent, and the other which contains information readily available to the agent. Information is passed from the simulation structure to the agent structure through an emulated sensor, and agent response to state information is passed back from the agent structure to the simulation through an emulated actuator. This structuring helps ensure that the agent response is only informed by accessible data, and

that those responses only come from the agent itself.

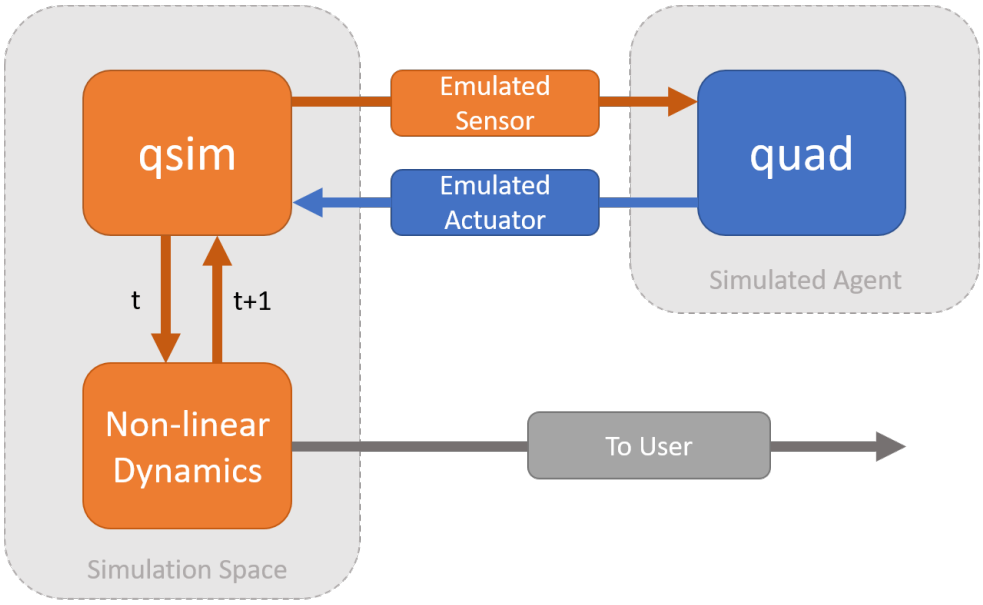

Figure 3.1: Diagram representing the data partitioning contained within the final quadrotor simulation suite. The core concept is that control response of the agent (quad) structure is only informed by data made available to the agent through sensors or on-board constants. The simulation (qsim) data is used for true state update, and as a source for measurements made by the sensors.

General System Discretization: Quadrotors and robotic systems in general have a number of discrete-time components. In particular, the controller update rate dictates the speed at which the agent can issue updates to its control response. Sensors also have samples rates, which dictates the rate at which data can be sampled to inform the control response. For a high-fidelity simulation, each of these control and sensor discretizations must be emulated to produce a response as similar to the live system as possible.

The quadrotor simulation suite uses a series of running timers, which are progressed by the discretization of the simulation at the beginning of each loop. When the timer reaches its set discretization, any discrete-time tasks related to that timer are executed, and the timer is reset at the end of the loop. Figure 3.2 shows a pseudo-code example of one of

these timers being used to update a given action. A greater-than check was used in favor of a modulus function check (if $mod(timer_1, qsim.dt) == 0$) to avoid potential incompatibilities with the discretization of the element and the simulation. For example, if the user input $qsim.dt = 0.001$ and $dt_1 = 0.0066$, then the modulus operator would always show a remainder and thus never issue an update.

```
for t = 1 : qsim.t_max/qsim.dt
  timer_1 = timer_1 + qsim.dt
  ...
  if timer_1 ≥ dt_t1
    Discrete-time Task 1
  end
  ...
  if timer_1 ≥ dt_t1
    timer_1 = 0
  end
end
```

Figure 3.2: Psuedo code example of the discrete-time timers used by the quadrotor simulation suite. Greater-than checks are used in place of modulus checks to avoid simulation and system discretization incompatibilities.

For the purposes of simplicity, there are four primary discretized systems emulated in the simulation suite are: 1) rotational controller update rate, 2) position controller update rate, 3) GPS buffer rate, and 4) GPS sample rate. The distinction made between the GPS buffer and sample rates is important because for systems like the Vicon positioning system used in the RAIN lab, the position data is only made available at a specific rate. For GPS

specifically, there is a significant and inconsistent delay that can be introduced in the process of data transfer to the agents. As such a second timer was included to allow for a variable discretization for the sampling rate of the position information from a buffer which is updated at the rate of the sensor itself.

This scheme implicitly assumes that all other delays and discretizations that exist on the live platform are small in comparison to those monitored by the simulation, and are thereby approximately continuous.

Sensor Emulation: As shown in figure 3.1, the purpose of the emulated sensors is to pass data from the simulated space, to the simulated agent. In reality, sensors are not able to make a perfect measurement of the system state. Vibrations, electrical noise, and calibration errors are just a few of the ways that state measurements can be corrupted from their true values. While these other potential sources of measurement variance can be emulated, we assume that the primary source of data corruption is noise. As such, the emulated sensors of the quadrotor simulation suite inject noise into each measurement upon sampling. The noise injected into each signal will be characterized from live system measurements, however can be modified as needed depending on the purpose of the simulation. The two primary sensors that are emulated are an external position and yaw tracking system, and an on-board Inertial Measurement Unit (IMU). These sensors are updated at their respective sample rates using independent timers discussed in the General System Discretization section above.

Noise Emulation: All sensor noise on the system is assumed to be normally distributed and consistent during operation of the platform. This assumption implies that all system noise has a consistent distribution during operation, and is independent of state or control inputs. In reality, the noise influencing the live platform will often be a combination of state and control inputs. We make this assumption in part for simplicity, but also with the understanding that the system will be operating around its linearized

point. For example, noise from sources like propeller flapping will be dependent primarily on the speed of the motors, which will be operating around hover. If the system is to maintain a controlled hover condition, then the deviations from the noise characteristics at hover can be assumed to be relatively small, or at the very least, short in duration. Therefore the noise injection into sensor signals is given by,

$$\nu = \nu_{true} + n(\mu, \sigma)$$

where ν is some measured state, $n(\mu, \sigma)$ is the standard normal distribution, with some mean μ, and standard deviation σ collected from experimental data.

The only exception to the normally distributed noise assumption is the wireless communication delay of GPS data, which will be characterized as a noise function that will be matched with experimental data in the next chapter. For now we will write this as a generic functional e_{dt} biasing the sample rate,

$$dt_{GPS} = dt_{GPS} + e_{dt}$$

Control Emulation: The control response of a given controller is governed by the update rate of the parent controller computing the control response. Therefore in the delay time that exists between each controller update, the control response is held constant until the next update.

If the system is emulating multiple micro-processors that are tasked with enforcing a control policy, care must be taken to ensure that the control is passed back to the simulation space through the controller that has access to the actuator.

Actuator Emulation: As show in figure 3.1, the emulated actuator is responsible for passing control responses from the simulated agent to the simulation space. Due to the small size of the system that is being simulated, the motors are assumed to enforce control responses instantaneously relative to the dynamics of the system.

3.1.1 Formation and Multi-agent Simulation

Inter-agent Data Segregation: The quadrotor simulation suite was designed to simulate an arbitrary n number of agents, which is predefined by the user. As was done in the case of the single agent, the same principal of data segregation was applied to each quadrotor in the simulated swarm. Separation of agent-specific values allows for inter-agent networking techniques to be designed and tested before implementation on the live platform.

Figure 3.3 shows how the data segregation principal was applied in the context of an arbitrary number of agents. To avoid making a number of unnecessary independent structures, which would be difficult to pass around and access properly within the existing simulation structure, an indexing approach was used instead. All agent specific data is then updated in that agent's relevant index, and accessed only if the accessing agent can feasibly acquire that information. The indexing is also necessarily applied to the simulation space as well, to store the relevant simulation information for each agent to reference when making a measurement or issuing a control response.

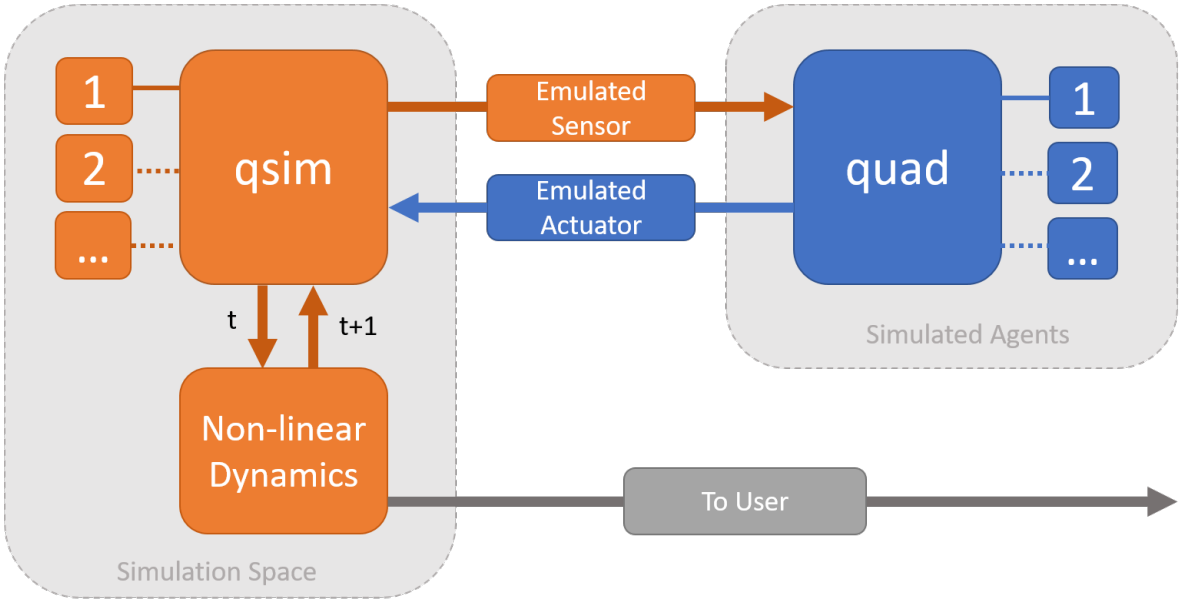

Figure 3.3: Diagram representing the modified data partitioning contained within the final quadrotor simulation suite to account for a user-defined number of simulated agents. The modification made was the addition of agent-specific indexing added to the simulated agent (quad) and simulation space (qsim), where each agent is updated at the same time-index.

3.2 Payload

Simulation of the payload dynamics is handled in the same manner as discussed earlier with regards to the quadrotor. While the current iteration of the payload does not have any special considerations like dynamics discretization, actuators, or sensors, special care must be taken when considering the forces acting on the system, in particular the tension.

Cable Suspension: Cables serve primarily as forcing control inputs from the supporting quadrotors, and can be modeled in a variety of manners. If they are assumed to be inelastic, then the cable serves to transfer any forces acting through the cable, with no consideration for cable stretching or spring coefficient. As such, the simplified cable model simply redirects the force from the agents if it is in tension.

There are two primary conditions that will result in a loss of cable tension:

Negative tension: For the purposes of this thesis, we will define negative tension as any resulting force acting through the cable from the quadrotor towards the attached payload. Since the cable is unable to transfer compressive forces, the cable will slack and zero net force will be transferred. To ensure that these forces are not unintentionally introduced into our payload dynamics, the cable tension in each line is updated depending on the next state of the payload and agents.

$$if: \quad T_{L,i}(t+1) < 0 \implies T_{L,i}(t+1) = 0$$

Cable-slacking: This condition occurs either when the norm distance between the supporting agent's anchor and its respective payload anchor is less than the total cable length, or when the tension is negative. The negative tension condition has already been discussed in a previous paragraph. The alternative case is handled by checking the norm of the difference of the updated quadrotor and payload inertial state,

$$\eta_i = \| \ (R_i(t+1)e_a\hat{e}_a + \xi_i(t+1)) - (q_i(t+1) + \xi_L(t+1)) \ \|$$

$$if: \quad \eta_i < L_i \implies T_{L,i}(t+1) = 0$$

Load Distribution: In the event that there is a loss of tension to one or more lines, the load is assumed to distribute across the remaining agents still in tension. The quadrotor simulation suite handles this by first checking how many of the agent lines can make a positive tension contribution. Then the total inertial forces acting on the payload are divided over the total number of agents in tension. If the above conditions for taut cable and positive tension are met, then the tension is the component of the total forces acting on the payload in the direction of the supporting agent cable,

$$T_{L,i}(t+1) = \frac{1}{n_t}\left(r_i \cdot \left(-\sum^{n}(T_{L,i}d_i) - m_L g e_3\right)\right)$$

where n_t is the number of agents in tension ($n_t \leq n$).

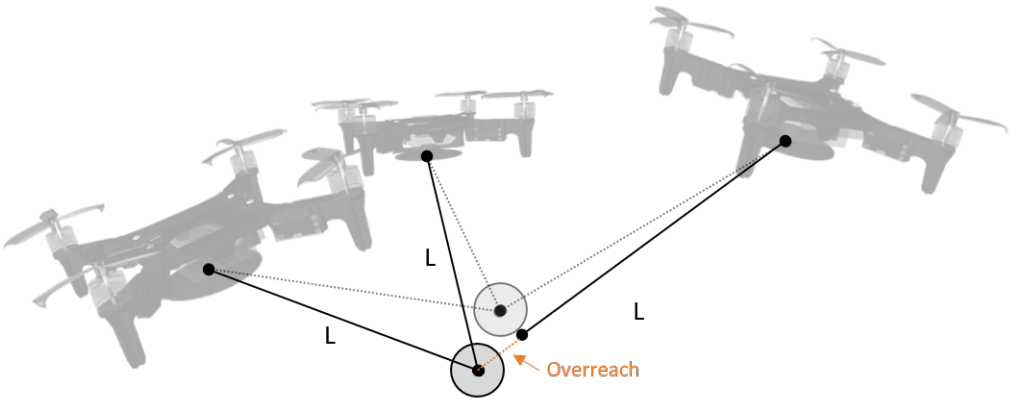

Figure 3.4: Diagram illustrating the position correction required when the payload attempts to move beyond the reach of the cable length. Since the cable is modeled as inelastic, the position is bounded within the norm distance from the agent allowed by the cable.

Cable Limit: Since the cables are assumed to be inelastic in nature, they have a fixed maximum distance they are allowed to extend when in tension. Enforcing this condition on the payload itself is not trivial, in particular when handling multiple supporting agents. This is because in enforcing the maximum norm distance limit on the position of the payload itself, the bounded position may place the anchors of other agents in a state violating their maximum cable lengths or cause them to be placed in a slacking condition. Figure 3.4 illustrates the challenge presented by enforcement of finite cable length on the payload, for a point-mass. The problem compounds further when introducing the distributed anchor points on a rigid body.

Rather than restrict the motion of the payload itself, we can instead restrict the motion of each agent, relative to its own anchor [23],

$$\xi_i(t+1) = \begin{Bmatrix} \xi_L(t+1) + q_i(t+1) - R_i(t+1)e_a\hat{e}_a + L_i\frac{\eta_i}{\|\eta_i\|} & if & \| \eta_i \| > L_i \\ \xi_i(t+1) & else & \end{Bmatrix}$$

This method of enforcement essentially treats the payload as a coupled system with independent forcing actuators, which have constraints acting on their dynamics.

Once the agent position has been corrected, the current configuration of the quadrotor simulation suite continues simulating the quadrotor dynamics as is. Future iterations of the simulation code will introduce inelastic collisions upon re-establishing tension, which will better reflect the live system [23]. However due to the small scale of the masses being dealt-with, this may not be necessary and will be determined in system validation if this simple bounding condition is sufficient for emulation.

Chapter 4

FINAL SYSTEM DESIGN

In the end, an in-house custom platform was determined to best meet the RAIN lab's current and future research needs. While platforms like Crazyflie have excellent documentation and resources available to help interface and adapt novel algorithms on-board, the underlying requirement is to use their preexisting architecture, which may harbor limitations and hidden costs associated with expanding functionality. The final in-house platform was developed to be fully Arduino-based, affording future projects the design freedom to change and upgrade flight-components with relative ease, so long as they are compatible or have some down-loadable package to interface with them. As Arduino fosters a massive maker community, the likelihood of such packages being already developed and optimized is very high. The final platform was also designed to be fully 3D printed, allowing for quite replacement of frame updates to be cycled through at a low cost.

This section will forego the details of the rapid-prototyping process, and instead focus on the current design iteration of the platform, briefly mentioning any practical challenges that were encountered during the development process. Final controller and estimation gains will also be discussed here, as will any modifications that were made to implement the final trajectory and control policies outlined in chapter 2.

4.1 Quadrotor

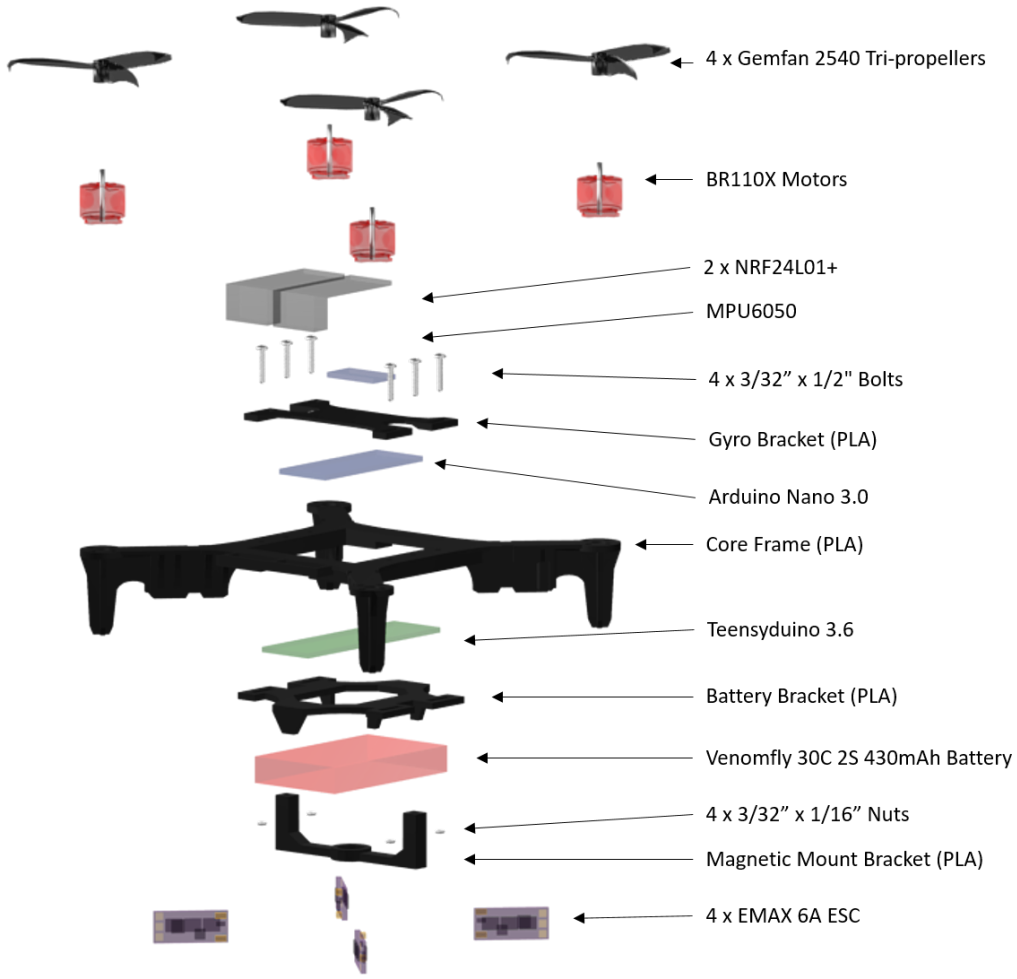

Figure 4.1: Exploded-view of the final RAIN-DROP quadrotor design.

4.1.1 Physical Design

Figure 4.1 shows the exploded-view of the final physical design of the RAIN lab's Drone Research Operations Platform (RAIN-DROP). The RAIN-DROP uses a printed PLA frame with print fill-density of 20%. The frame type is a symmetrical 'X' configuration, placing the pitch and roll axes between symmetric motor pairs. Design print-ability was an

important consideration, as the available 3D printers each used fused-deposition-modeling (FDM), meaning the model would be constructed from the bottom face to the top face. FDM printing presents a number of challenges when printing models with sharp overhangs, as an overhang will have no supporting material to deposit new filament onto, and often requires the introduction of removable supports to provide a printing base. The addition of supports adds another step to the manufacturing post-processing, and adds cost in wasted plastic. As such, the RAIN-DROP was specifically designed to be printed with one side face-down, without the addition of any support material. Figure 4.2 shows the printed components as they would be assembled on the print-bed.

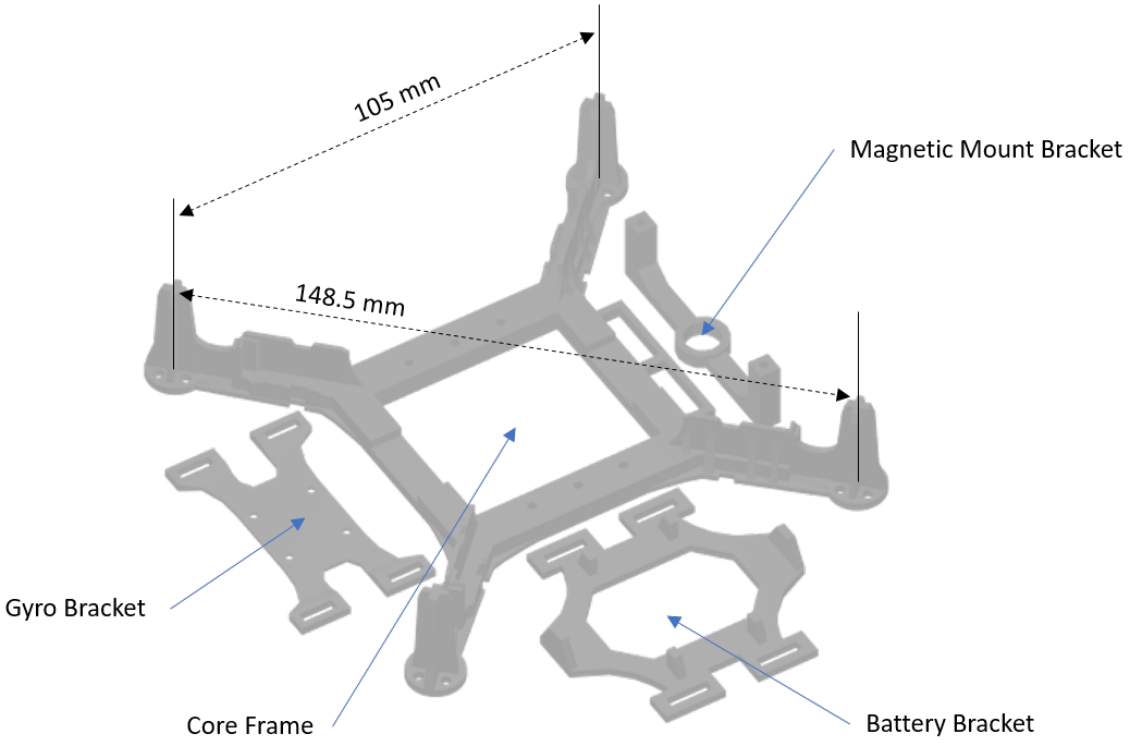

Figure 4.2: The printed form of the RAIN-DROP frame and peripherals, as they would appear on the bed of an FDM printer.

Below each of the arms are specially designed sockets for the electronic speed controllers (ESCs) which mount each ESC below their respective motors, taking advantage of the down-thrust of the propellers for cooling. The central cavity of the core frame was intentionally left open to allow for easy packing of wires, addition of new peripherals, and upgrading of on-board micro-controllers. The magnetic bracket serves as the attachment point for any object transportation applications, as well as securing the battery. The use of a replaceable bracket to secure the battery itself allows future users of the RAIN-DROP to upgrade the battery used by the platform, without replacing the core frame itself.

Figure 4.3: View of the RAIN-DROP showing the ESC mounting sockets, battery bracket, and magnetic mount bracket as designed to secure to the frame.

Mass and Moment of Inertia: The final mass of the RAIN-DROP without the battery was 100.9 g. As the quadrotor frame is a relatively complex shape, we can approximate the inertial tensor by determining the inertial tensor of a simpler composite shapes encompassing the primary mass contributing volumes. Figure 4.4 shows one

potential composite shape that can be used to approximate the total system inertia.

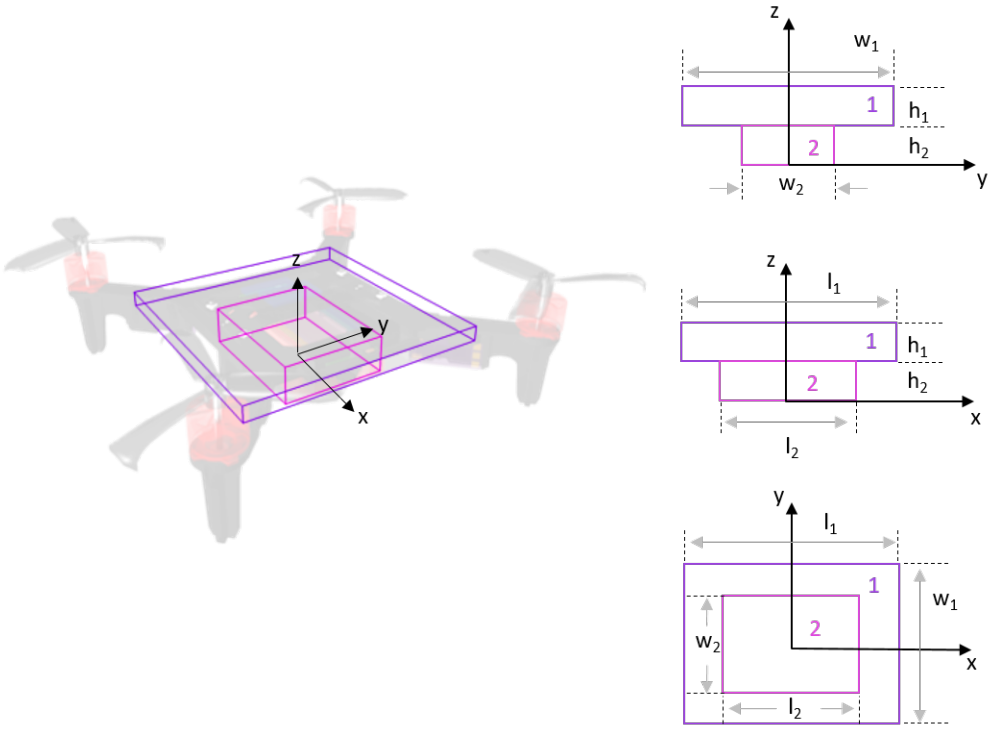

Figure 4.4: Graphical representation of the quadrotor moment of inertia approximated by a composition of two rectangular prisms, where the reference axis is centered at the base of (2).

From the parallel axis theorem in dynamics, we can shift the center of an inertial mass at a general point P, along a parallel axis to another relative point [9],

$$I_{xx} = (I_{x'x'})_P + m\left(y_P^2 + z_P^2\right)$$

$$I_{yy} = (I_{y'y'})_P + m\left(x_P^2 + z_P^2\right)$$

$$I_{zz} = (I_{z'z'})_P + m\left(x_P^2 + y_P^2\right)$$

where x_P, y_P, z_P are measured relative to the axis the inertia is being shifted towards. If we want to determine the inertial tensor about the composite centroid, $\bar{x}, \bar{y}, \bar{z}$, we simply take

the difference of the centers of each composite shape element, and subtract the centroid location, thereby shifting the reference point to the centroid itself.

$$\xi_P = \xi_i - \bar{\xi}$$

where ξ_i is the position of the i-th element, measured relative to the same origin as the total composite centroid. The centroid of a 3D body can be found from,

$$\bar{\xi} = \frac{\sum^i V_i \bar{\xi}_i}{\sum^i V_i}$$

where $\bar{\xi}_i$ is the centroid of the i-th volume, V_i comprising the composite shape. Once the centroid has been determined, we can leverage the parallel axis theorem relative to the centroid of the composite shape, determining our principal inertial values,

$$I_{xx} = \sum^i \left(I_{xx,i} + m_i((y_i - \bar{y})^2 + (z_i - \bar{z})^2) \right)$$

$$I_{yy} = \sum^i \left(I_{yy,i} + m_i((x_i - \bar{x})^2 + (z_i - \bar{z})^2) \right)$$

$$I_{zz} = \sum^i \left(I_{zz,i} + m_i((x_i - \bar{x})^2 + (y_i - \bar{y})^2) \right)$$

Similarly the parallel plane theorem reveals the products of inertia,

$$I_{xy} = \sum^i \left(I_{xy,i} + m_i(x_i - \bar{x})(z_i - \bar{z}) \right)$$

$$I_{yz} = \sum^i \left(I_{yz,i} + m_i(y_i - \bar{y})(z_i - \bar{z}) \right)$$

$$I_{zx} = \sum^i \left(I_{zx,i} + m_i(x_i - \bar{x})(z_i - \bar{z}) \right)$$

For our system, we found the inertial value of the composite shape shown in figure 4.4, where $m_1 = 0.100$ kg, $h_1 = 5$ mm, $w_1 = 70$ mm, and $l_1 = 70$ mm, and with $m_2 = 0.029$ kg, $h_2 = 10$ mm, $w_2 = 30$ mm, and $l_2 = 50$ mm. The inertia for each composite element can be found readily from the rectangular prism inertia relationship,

$$I_{xx,i} = \frac{m_i}{12}(w_i^2 + h_i^2)$$

$$I_{yy,i} = \frac{m_i}{12}(l_i^2 + h_i^2)$$
$$I_{zz,i} = \frac{m_i}{12}(w_i^2 + l_i^2)$$

Using the above relations, we calculated the approximate inertia matrix of the RAIN-DROP to be,

$$I_{QR} \approx \begin{bmatrix} 44.897 & 0 & 0 \\ 0 & 48.764 & 0 \\ 0 & 0 & 89.883 \end{bmatrix} \text{kg} \cdot \text{mm}^2$$

Due to our approximated shape, the products of inertia (I_{xy}, I_{xz}, I_{yz}) are zero, resulting in a principal inertia matrix. This principal inertia approximation assumes that the respective density for composite shape element is constant. While it is also possible to account for individual motors by using the parallel axis theorem and the parallel plane theorem (as the products of inertia would no longer be trivial), this approximation is sufficient for the purposes of this thesis.

4.1.2 Computational Hardware and Architecture

For the RAIN-DROP, the computational responsibilities were segregated into a high-level and low-level processes, which were delegated to two independent processors. The high-level processes of the platform include inertial control and estimation, trajectory generation, communication, formation control, and any custom research algorithms. The low-level processes include attitude estimation and control, battery monitoring, interlock enforcement, and actuation command.

Due to the lower computation demands of the low-level processes, the RAIN-DROP uses an Arduino Nano (8-bit, 16 MHz ATMEGA328P). We denote this processor as the *Pilot*, as it is tasked with the baseline actuation and operation of the platform. The processes handled by this micro-controller primarily relate to the attitude control and control enforcement, which need to be updated at a consistent rate to operate as designed.

Comparatively the computational demands of the high-level processes can vary

significantly with respect to the low-level. When selecting the processor to be used, we wanted to ensure the platform had the capacity to house and execute complex algorithms, without needing to rely on off-board computational resources. This mindset lead to the selection of the Adruino TEENSY 3.6, which uses a much higher-power MK66FX1M0 processor (32-bit 180 MHz). We denote this processor as the *Navigator*, which is predominantly responsible for the inertial dynamics and control of the platform. As the attitude control and low-level functions are being handled by the Pilot processor, this affords greater flexibility in computational delays on-board the Navigator.

Inter-processor Communication: In order to pass command data from the Navigator to the Pilot processors, a communication pathway is needed. One very fast approach to accomplish this is to use the Serial Peripheral Interface (SPI). This protocol sets one microprocessor as a master, and any attached processors or devices as slaves. A major key advantage of SPI communication is simultaneous transfer. As the data is sent from the master processor's transfer buffer, any data contained within the slave transfer buffer is simultaneously pushed into the master's buffer (or pushed into the next slave transfer buffer along a chain, depending on configuration). The RAIN-DROP uses this simultaneous transfer approach to pass attitude and battery data from the Pilot to the Navigator, while simultaneously receiving its next attitude and thrust set-points. The primary drawback to SPI transfer is data corruption during transfer. A simple error checking scheme was introduced to prevent corrupt data from being implemented by the pilot controller, by simply sending the data twice and matching the two packages. In the rare instance an error occurs, the message is ignored and updated on the next communication loop and not brought into the command loop. For safety concerns, the RAIN-DROP uses a timer to monitor when it has last received a full data transfer without any errors. If the timer grows to be beyond a certain threshold, the system enters a shut-down state.

4.1.3 Sensors and Estimation

Sensor Hardware: Table 4.1 lists the current sensors hardware used by the RAIN-DROP. As the platform was not designed to fly outside the RAIN lab flight-space, the Vicon data was deemed sufficient for operations. A later expansion that can be made is the addition of a barometric or time-of-flight sensor for height detection, and a compass for yaw correction.

Sensor	State Measured	Units	Max Update Rate	Platform Update Rate
MPU-6050 IMU	$\dot{\varphi}$	deg/s	1 kHz	150 Hz
	$\ddot{\xi}_B$	g		
Vicon	ξ	mm	100 Hz	100 Hz
	ψ	deg		
Voltage Divider	V_{bat}	ADC	+50 kHz	150 Hz

Table 4.1: Tabulated list of the sensor hardware used by the RAIN-DROP. The current update rate denotes the rate at which the information is updated on the RAIN-DROP in its current configuration.

Noise Characterization: On the RAIN-DROP, the predominant noise encountered is vibration, in particular with respect to the gyro and accelerometer measurements during operation. Vibrations can originate from multitude of sources ranging from propeller damage and mis-alignments, to poorly-securely components and natural frame flexing. These vibrations will only increase in severity as the platform size-profile decreases, as the reduction in arm-length unavoidably placing the IMU closer to sources of vibration.

Gyro and Accelerometer Noise Characterization: The IMU noise characteristics were collected by recording the raw Accelerometer and Gyroscope measurements while operating at half-maximum throttle with the propellers mounted. Figure 4.5 shows the results from a four-second data collection interval. The noise produced on the accelerometer

is quite significant, and needs heavy filtering to be usable. In addition, the high frequency noise experienced on the Gyro will also need to be filtered to ensure usability

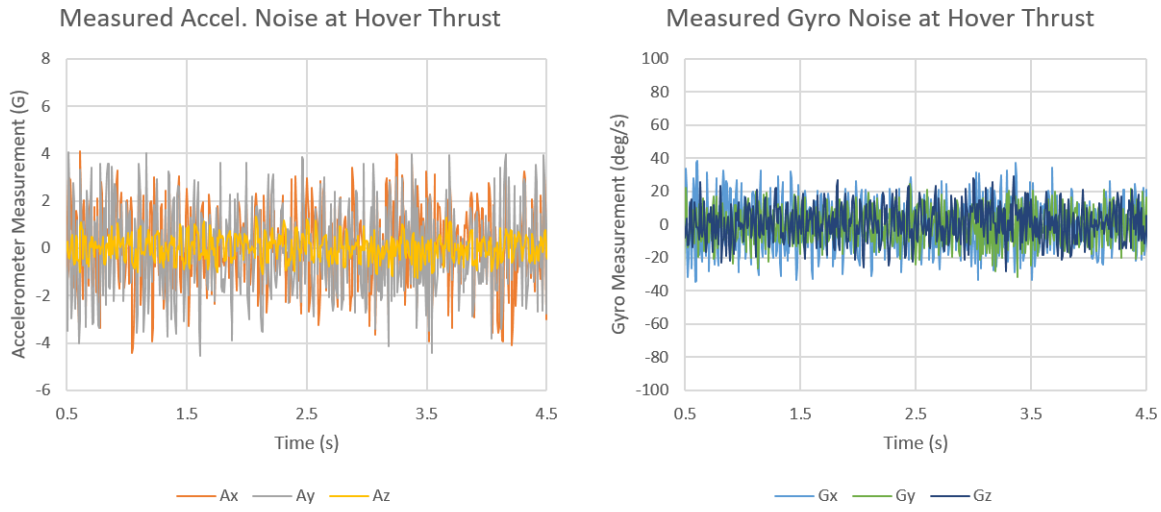

Figure 4.5: Figure of the noise measurements collected over a four-second interval Agent 2 commanded to half-maximum throttle. (Left) Raw Accelerometer measurements, which had a measured standard-deviation of $\sigma_{acc} = [1.647, 1.785, 0.517]$ G. (Right) Raw Gyro measurements, which had a measured standard-deviation of $\sigma_{acc} = [14.74, 10.10, 10.27]$ deg/s.

Vicon Position and Yaw Noise: The Vicon positioning system used by the RAIN lab has a listed accuracy of ± 0.1 mm. Figure 4.6 shows the plot of the measured position error relative to the mode of the data set. Indeed the position data has random noise introduced into the data stream of ± 0.1 mm. The discrete nature of the noise intrusion is most likely due to the resolution limitations on the Vicon positioning system used, where noise that would read at a higher resolution is rounded to discrete value (eg. $1.026 \rightarrow 1.1$). For the sake of simulation, this discrete noise will be modeled as a rounded normal distribution with a standard deviation of $(\sigma_x, \sigma_y, \sigma_z) = (0.04, 0.015, 0.32)$ mm scaled by 0.1. As the yaw is relatively high resolution, a simple normal distribution with $\sigma = 0.024^o$ can be used.

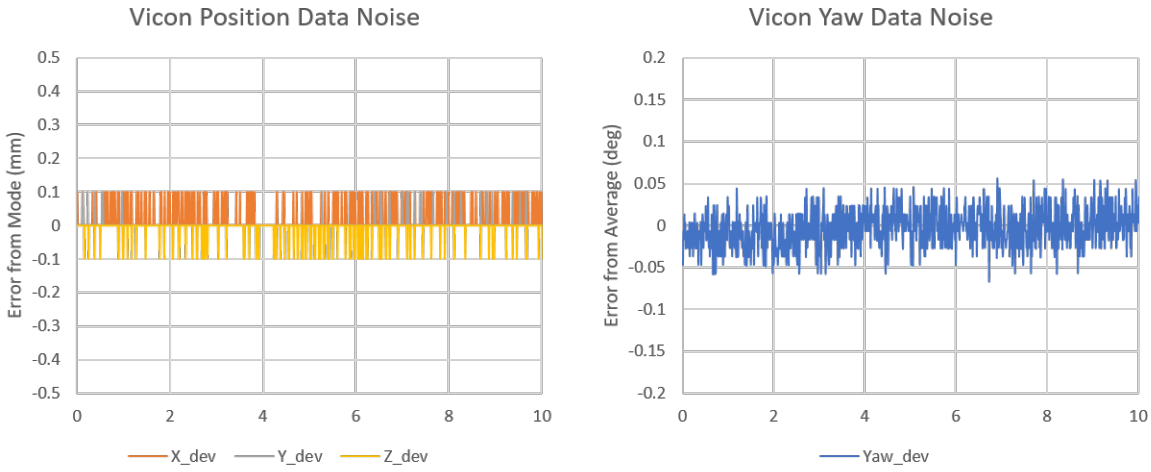

Figure 4.6: Plot of the measured noise from the Vicon data stream as seen by the agent. The Vicon system is used for both position and yaw information. (Left) Plot of the error relative to the mode of the position data, with a sample standard deviation of $(\sigma_x, \sigma_y, \sigma_z) = (0.04, 0.015, 0.32)$ mm. (Right) Plot of the error relative to the average of the yaw data. $\sigma_\psi = 0.024^o$

Vicon Data Stream Sample Rate: The Vicon position and yaw data is sent over wireless to each of the agents measured in the field of view of the cameras. There are a number delay sources that exist between the data generation by the Vicon hardware and data reception by the RAIN-DROP platforms. Additionally, as the data for each agent is collected and sent as a single package from the ground-station computer, the number of agents also impacts the time between data packets. Figure 4.7 shows the measured delay between Vicon data updates measured on agent 2, while in the presence of 2 other agents also receiving data. Unfortunately it is clear that this particular noise profile is not normally distributed, and therefore needs to be emulated. Since the data transmission is lower-bounded by the maximum update rate of 100 Hz from the Vicon system, we can define a biased one-sided normal function,

$$e_{dt} = c_{dt}|norm(0, \sigma_{dt})| + b_{dt}$$

Where σ_{dt} is the standard deviation, c_{dt} is a scaling factor, and b_{dt} is a bias factor. To emulate the delay incurred from random packet drops packet drops we introduce an overlay normal,

$$e_{dt} = \begin{cases} c = 1 & if \quad |norm(0, \sigma_{drop})| \geq 0.01 \\ c = 1.5 & if \quad |norm(0, \sigma_{drop})| < 0.01 \end{cases}$$

which is governed by σ_{drop}. Our final sample rate is then computed by,

$$dt_{GPS} = 0.01 + e_{dt}$$

Figure 4.7 shows how this function compares against the measured sample delays for $\sigma_{dt} = 0.0029$, $\sigma_{drop} = 0.003$, and $b_{dt} = 0.001$.

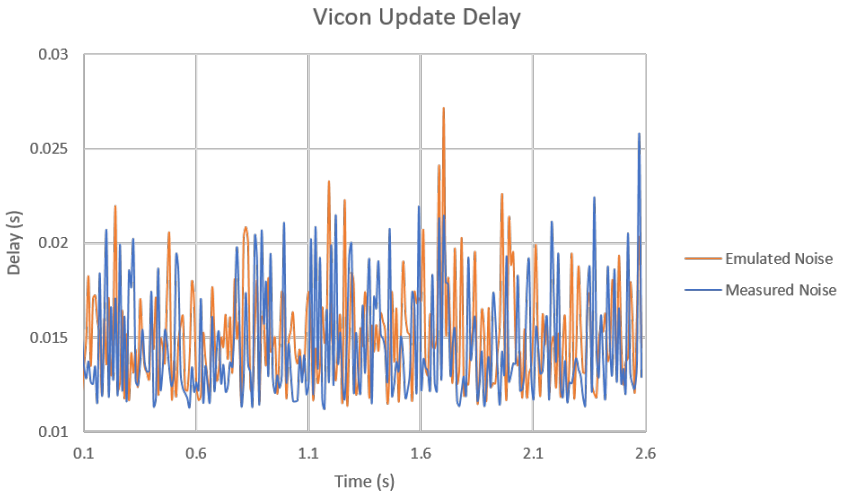

Figure 4.7: Plot of the measured and emulated delay between Vicon data updates received by an agent in the presence of other agents which are being update on a common channel. The emulated data was produced by the custom function shown in this section, with $\sigma_{dt} = 0.0029$, $\sigma_{drop} = 0.003$, and $b_{dt} = 0.001$.

Attitude Estimation: Since we measure the attitude rates directly using a gyro, we can generate the resulting Euler angles of the platform by integrating using forward Euler approximation,

$$\varphi_g(t) = \varphi_g(t-1) + dt\dot{\varphi}(t)$$

where φ_g denotes Euler angles estimated from the gyroscopic rates.

While gyroscopes tolerate system vibrations well (assuming that they are well-mounted and position at the center-of-rotation), their primary drawback is low-frequency drift. This occurs due to the angle being integrated from the rate values, which may have some low-frequency bias that change over time from the initial calibration point [3, 20]. One tactic to combat this is use of a complementary filter [3, 8].

As outlined in Chapter 2, the complimentary filter combines a weighted average of two estimates of a given state. In addition to the gyro, the attitude information can also be estimated from the 3-axis accelerometer pointing vector and a mangetometer [8, 20]. Unfortunately, both these sensors are highly susceptible to system noise, namely platform vibrations for the accelerometer, and magnetic fluctuations from the motors for the magnetometer. If these measurements are filtered heavily using a low-pass approach, then the final value will eventually converge to the nominal value. Since the acceleration vector is predominantly pointing in the direction of gravity, the measurements made from each of the accelerometer axis can be used to determine the pitch and roll axis [8],

$$\phi_a(t) = \tan^{-1}(\frac{a_y(t)}{a_z(t)})$$

$$\theta_a(t) = \tan^{-1}(\frac{-a_x(t)}{\sqrt{a_y(t)^2 + a_z(t)^2}})$$

We can then use a complementary filter to augment the final estimated pitch and roll angle,

$$\phi_c(t) = (1 - k_c)\phi_g(t) + k_c\phi_a(t)$$

$$\theta_c(t) = (1 - k_c)\theta_g(t) + k_c\theta_a(t)$$

Normally in the case of a GPS tracked quadrotor, yaw information would need to be gathered by a magnetometer. Fortunately the Vicon position tracking system is capable of providing an external yaw measurement, which we can use to update our gyro estimate.

$$\psi(t) = \psi_v(t)$$

As the attitude update rate is 50 Hz faster than that of the Vicon, the gyro integrated yaw estimate is used to maintain control of the yaw and yaw rate in between Vicon updates.

Since them IMU will encounter high-frequency noise from both electrical static and mounting vibrations, a low-pass filter was applied to both the acclerometer and gyro measurements to minimize the intrusion of noise into the control response,

$$\begin{bmatrix} \phi_g(t) \\ \theta_g(t) \end{bmatrix} = (1 - k_g) \begin{bmatrix} \phi_g(t-1) \\ \theta_g(t-1) \end{bmatrix} + k_g \begin{bmatrix} \phi_{g,raw}(t) \\ \theta_{g,raw}(t) \end{bmatrix}$$

$$\begin{bmatrix} a_x(t) \\ a_y(t) \\ a_z(t) \end{bmatrix} = (1 - k_a) \begin{bmatrix} a_x(t-1) \\ a_y(t-1) \\ a_z(t-1) \end{bmatrix} + k_a \begin{bmatrix} a_{x,raw}(t) \\ a_{y,raw}(t) \\ a_{z,raw}(t) \end{bmatrix}$$

The final RAIN-DROP attitude filter and estimation gains are tabulate in table 4.2.

Gain	Value
k_g	0.5
k_a	0.04
k_c	0.0075

Table 4.2: Final attitude estimation gains for the RAIN-DROP quadrotor test-bed.

Inertial Estimation: The position and yaw information provided by the Vicon tracking system has very little noise (as compared to a traditional GPS), and in general does not need to be filtered. To produce the velocity and acceleration states we leverage a low-pass filter and the finite difference approximation.

$$\dot{\xi}(t) = \frac{1}{dt}(\xi(t) - \xi(t - dt)), \qquad \xi = \begin{bmatrix} x & y & z \end{bmatrix}^T$$

$$\ddot{\xi}(t) = \frac{1}{dt}(\dot{\xi}(t) - \dot{\xi}(t - dt)), \qquad \dot{\xi} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T$$

These estimates are then used as artificially measured values, which will have some noise associated with them. A low-pass filter is used on both the acceleration and velocity values to address any estimation noise before entering the trajectory and control loops.

$$\dot{\xi}_f(t) = (1 - k_v)\dot{\xi}(t-1) + k_v\dot{\xi}(t)$$

$$\ddot{\xi}_f(t) = (1 - k_A)\ddot{\xi}(t-1) + k_A\ddot{\xi}(t)$$

where k_v and k_A are the associated filter gains. Table 4.3 lists the final inertial filter gains used by the RAIN-DROP platform.

Gain	Value
k_v	0.65
k_A	0.4

Table 4.3: Final inertial estimation gains for the RAIN-DROP quadrotor test-bed.

4.1.4 Actuators

The RAIN-DROP uses 4 brushless DC motors as the primary actuators for flight and inertial control. In order to operate brushless DC motors, a electronic speed controller is used to handle proper timing to charge and discharge the correct coils to maintain rotation of the motor bell.

Motors: Gemfan 1105 7500KV brushless DC motors.

Electronic Speed Controllers (ESC): EMAX Crazepony BLHELI_S 6A Bullet ESC for 2S batteries.

Arduino Pulse-width Modulation (PWM): In order to issue commands to each ESC, the Pilot controller needs to send a PWM pulse of a given length (in microseconds).

The Arduino IDE has a built-in command designed for PWM communication with servos. However, pulse-length consistency is a challenge with the Arduino based PWM fucntion. When using the Arduino-based commands there is an approximately $\pm 50 - 100$ μs oscillation added to the end of the signal. An alternate approach toggle each command pins directly, using the microsecond timer, which reduces the command pulse error to around ± 25 μs [3].

Propellers: Gemfan 2540 Tri-blade propellers.

Thrust-to-command Conversion: As mentioned, the ESCs require a PWM command to execute a desired thrust. Figure 4.8 shows the experimentally collected thrust-force output for a given command pulse-length. The measurements were made using a Vernier ± 50 N force sensor, secured at the end of a cantilever. The qaudrotor was then secured to the opposing end of the cantilever, and given command set-point. The force measurement was collected, and a 2nd-order polynomial was fit to the data set. The fit function was used as a basis to designate a manually adjusted polynomial with a intercept of 1000 μs, which corresponds to the true zero thrust command. This process determined the final thrust-to-command conversion polynomial used on the RAIN-DROP,

$$u_{i,cmd} = 40\left(4\frac{u_i}{1000}\right)^2 + 250\left(4\frac{u_i}{1000}\right) + 1000$$

Where u_i is the calculated input force for the i-th motor in mN. There is a multiplication factor of 4 applied to account for the polynomial fit being with respect total thrust output of the system, and not an individual motor. The $\frac{1}{1000}$ is the simple conversion from mN to N.

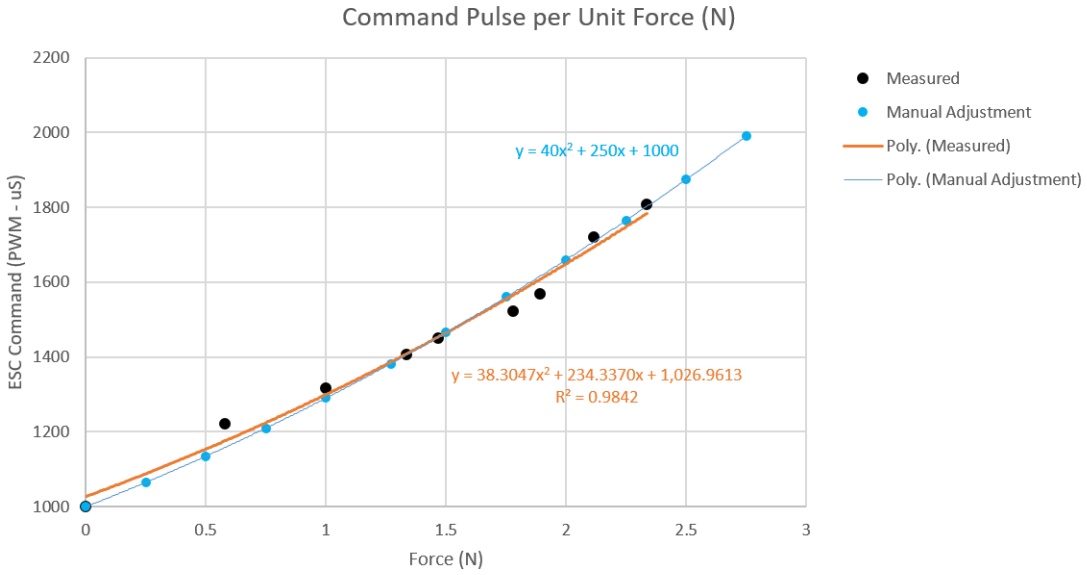

Figure 4.8: Plot of the total force output of the RAIN-DROP platform, for a given μs command pulse-length. The measurements had a 2nd-order polynomial fit to them, which was used as a basis to define a manually adjusted polynomial for implementation.

Motor Thrust Trimming: The motor thrust capabilities of each motor are generally not consistent across the platform. These discrepancies can derive from a multitude of sources such as: propeller damage, system mass in-balances, or even inconsistent battery placement. Regardless of the source, agent-specific motor-thrust trimming is a simple approach to address motor thrust imbalance. The RAIN-DROP uses a simple motor-thrust biasing term that is applied to the final motor command value.

$$u_{i,cmd} = u_{i,cmd} + u_{i,trim}$$

where $u_{i,trim}$ is the trimming value on that axis. This trim value can be computed from a desired trim about the roll-pitch axis r_{trim}, p_{trim} by using the mapping matrix,

$$u_{trim} = \begin{bmatrix} 1 & -1 \\ -1 & -1 \\ 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} r_{trim} \\ p_{trim} \end{bmatrix}$$

Application of the trim values in this manner helps the user intuitively trim along a desired axis of interest.

4.1.5 Communication Hardware

Transceiver: The RAIN-DROP uses two on-board NRF24L01+ transceivers, which operate on a 2.4 GHz frequency band, capable of transmitting up to 2 MB/s. There are multiple configurations of this particular transmitter, differing primarily on antenna type. The etched PCB antenna has a low physical profile, with a range of around 50-100 m. On the RAIN-DROP transceiver is dedicated for handling all inbound data from the ground-station, while a second transmitter is available for inter-agent communication and telemetry feedback.

4.1.6 Power Systems

The primary power system of the RAIN-DROP is the battery. Battery selection not only determines the flight-time of the craft, but also the discharge rate. The discharge rate of a given battery is measured in C, which is a metric relating the Amp-hr of the battery to the continuous discharge rate. Often smaller capacity batteries will have lower maximum discharge rates, which may produce reduced output from the motors and a noticeable voltage drop on the battery during peak current demand.

Battery: VenomFLY 2S (8.3-7.4 V) 30 C (50 C Burst) 430mAh.

A discharge rate of 30 C (50 C burst) means this battery can deliver a continuous discharge current of $30C * 0.43\frac{\text{A}}{\text{hr}} = 12.9$ A (21.5 A burst). The platform uses four 6 A ESCs with a per-motor power draw of around 3-6 A, therefore the power demanded by the system can be upwards of 12-24 A. As we can see, this is the primary area for improvement for performance, since the current demands of the platform will produced noticeable voltage drops if the current draw exceed the 12.9 A of available current for extended periods of time.

Voltage-level Thrust Compensation: During normal operation, the measured voltage accross the battery will reduce as it is depleted. A reduction in battery voltage results in a reduction in motor thrust output [3]. We can compensate for this thrust reduction by applying a compensation factor based on the battery measurement at a given point in time.

To determine the compensation factor, a simple experiment was set up to measure the force output of the quadrotor at a hover thrust. The force output was then held at a constant value, by increasing the command-pulse to the ESCs. Figure 4.9 shows the command-pulse compensation needed for a given battery ADC measurement. The slope of the compensation curve is -3.08, starting around 158 ADC. Implementing this on the live platform for free-flight tests, using the measured values as a basis, the final thrust compensation function was determined to be,

$$u_{i,comp} = \left\{ \begin{array}{cc} -2.0(V_{bat} - 175) & if \quad 130 <= V_{bat} <= 175 \\ 0 & else \end{array} \right\}$$

where the function is defined to only operate between a starting voltage measurement of 175 ADC, and the minimum battery voltage of 130 ADC. Note that 130 ADC corresponds to the minimum safe voltage for a 2S battery pack ($\sim$ 6.6 V).

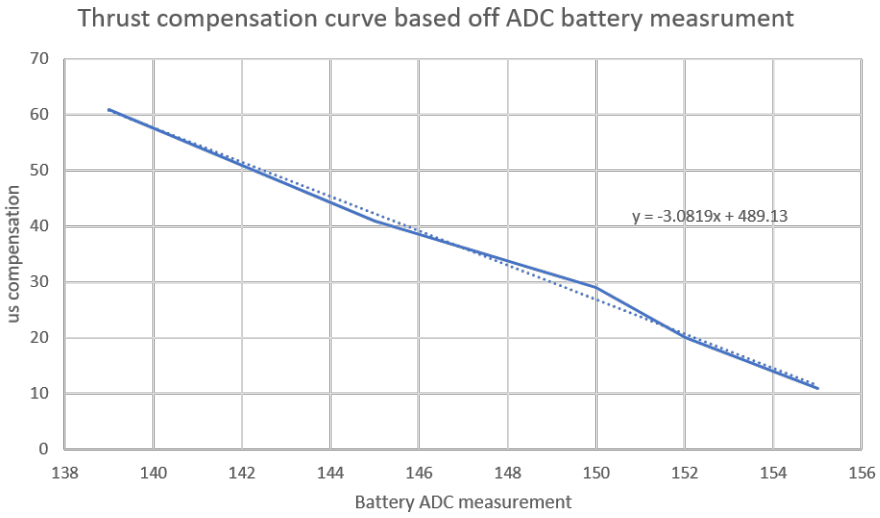

Figure 4.9: Plot of the additional command-pulse length needed to maintain a constant thrust output, for a given battery ADC measurement.

4.2 Communication Architecture and User Interface

4.2.1 Architecture

In general the communication requirements for the RAIN-DROP are partitioned into two catagories: 1) state update and 2) user command update. This natural data segregation, lead to the construction of the dual data pipeline structure shown in figure 4.10.

Communication with any RAIN-DROPs within the flight-space is handled two NRF24L01+ transmitters, each connected to a dedicated micro-controller. Use of two independent processors allows for uninterrupted through-put for both inbound and outbound data, where one is set as a permanent transmitter and the other as a receiver.

The transmitter is connected to a SAMD21 (Arduino Zero) processor, which has two UART serial connections to the user PC. One UART connection is dedicated to a Matlab interface that handles data acquisition from the Vicon position system, and the other is dedicated to the Python Graphical User Interface (GUI) set up in Blender. All outbound communications occur on the NRF24 channel 1. For control purposes, GPS data is given

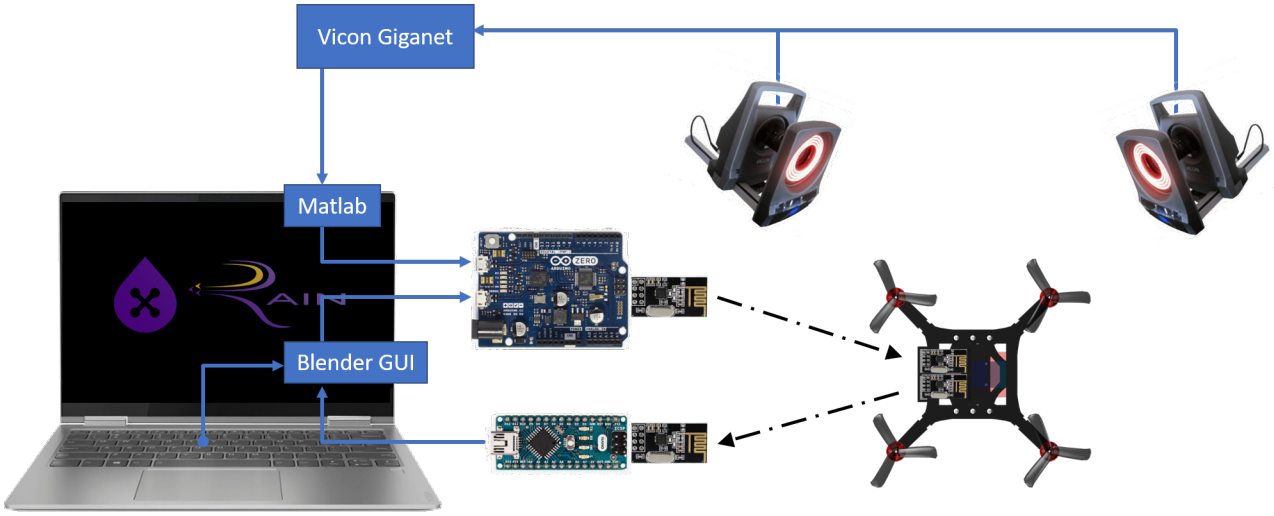

Figure 4.10: High-level communication architecture for the data being set to and from the quadrotor. Note that the data pipelines between the position tracking system and the user-interface are kept separate to avoid any delays in the position tracking updates due to the interface.

priority, as the GUI may be running a number of graphical or computational elements that can slow it down. In light of this, GPS data is sent along with the most-recent command information previously received. A watchdog timer is then used to monitor the connection with the GUI, and issues a system shut-down command in the event of a connection loss.

The receiver is connected to an Arduino Nano, which uses another separate UART connection to the GUI to display and record telemetry feedback from the RAIN-DROP. Return telemetry is passed along on channel 50, to avoid data collisions with the outbound data stream from the SAMD21.

4.2.2 Communication Protocol

A custom communication protocol was written for the RAIN-DROP, to both ensure proper data handling during operation, and sufficient flexibility to account for any custom applications developed by future RAIN lab members. Special attention was paid to ensure

that a fixed package size was used to ensure consistent data update timing, and efficient data usage.

Data Packet Structure

Ground-station to Quadrotor Data Structure: Outbound data would contain GPS data, user commands, or any other information being passed to the quadrotors. The structuring of the outbound byte-stream is as follows,

GPS/Command Data Packet		
Agent ID (1-byte)	Flight-data mode (1-byte)	Relevant Data (5 x 2-byte values)

Quadrotor to Ground-station Data Structure: Inbound data is the information being passed back from the quadrotors, which is currently only telemetry. The return package is afforded to be slightly larger, as the RAIN-DROP will have already completed its control updates, and is waiting for the next position update.

Telemetry Data Packet				
Agent ID (1-byte)	F-D mode (1-byte)	Data Received (4 x 2 bytes)	Pilot Data (4 x 2-bytes)	Desired Position and Yaw (4 x 2-bytes)

Agent ID: The indemnification number of a given RAIN-DROP.

Flight-Data Mode: A 1-byte value comprised of two 4-bit numbers (Flight Mode — Data Mode). The 4-bit flight-mode value corresponds to the overall category of control the quadrotor is seeking to execute. Within each flight-mode, there are then 16 potential 4-bit data modes that can be used, which generally indicate what data is being sent to the agent, and/or how it should be handled within the mode of operation.

4.2.3 Formation Communication

For the purposes of this thesis, communication across the formation consists of gaining knowledge of neighboring agent position and yaw information. This information is then used to inform the formation control policy and generate a response. Rather than having each agent establish direct communication link with neighboring agents, we instead leverage the common inbound data channel. This common channel broadcasts the position and command information of each agent in the flight space. Normally when each agent parses the inbound data, it first checks to determine if the agent ID tag on the data package matches its own unit ID. If it does, the data is processed normally, otherwise it is ignored. As such, if the agent ID of the package matches that of a squad-mate or neighbor, then the data can be readily populated into a buffer for formation control. Figure 4.11 illustrates this common channel data buffering.

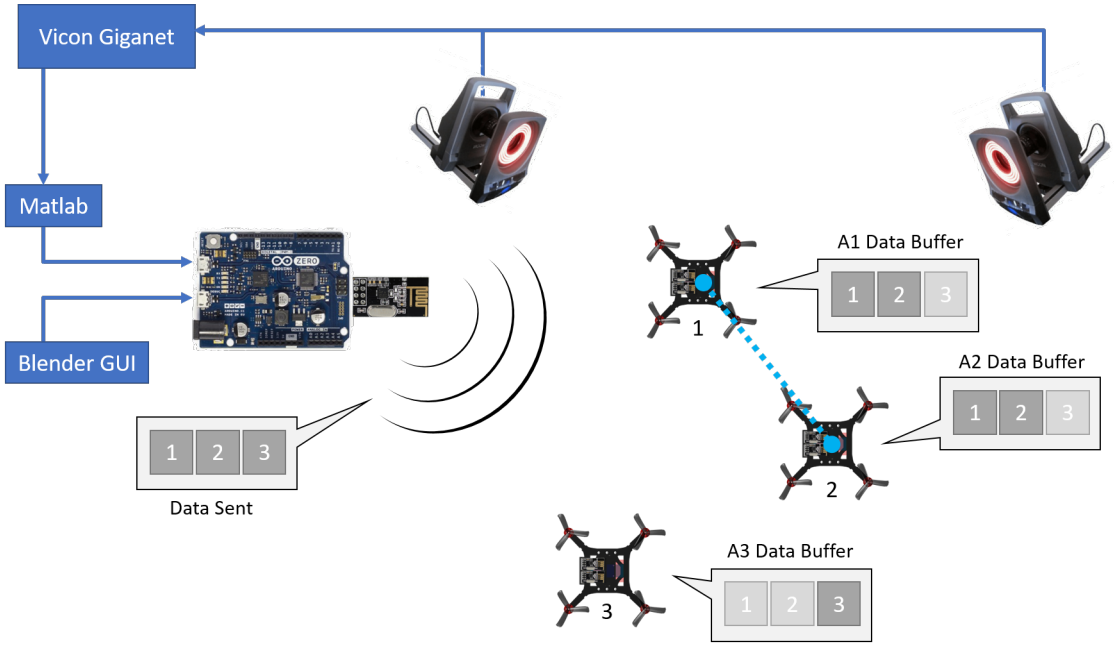

Figure 4.11: Graphical representation of the formation communication used by the RAIN-DROPs. The position and command information for each agent is generated and sent over a common communication channel. Once received, agents retain any data relevant to themselves or any squad-mates. In this figure, Agents 1 and 2 form a squad and retain each others information, while Agent 3 is independent and retains only its own information.

4.2.4 Graphical User Interface

A graphical interface was developed in order to provide users of the RAIN-DROP an intuitive means of issuing commands to quadrotors within the flight space. In addition, this GUI was designed to present the return telemetry in an equally intuitive form. As such, the GUI was developed in Blender 3D, which houses a fully functional Python environment. Python is an open source language that supports a number of custom packages and libraries maintained by a variety of open source communities, and allows for a great deal of flexibility in the functionality of the final interface.

Figure 4.12 shows a screen-shot of the position tracking user interface. Return telemetry is shown in real-time, which moves each quadrotor to the respective position and

orientation in a to-scale digital environment.

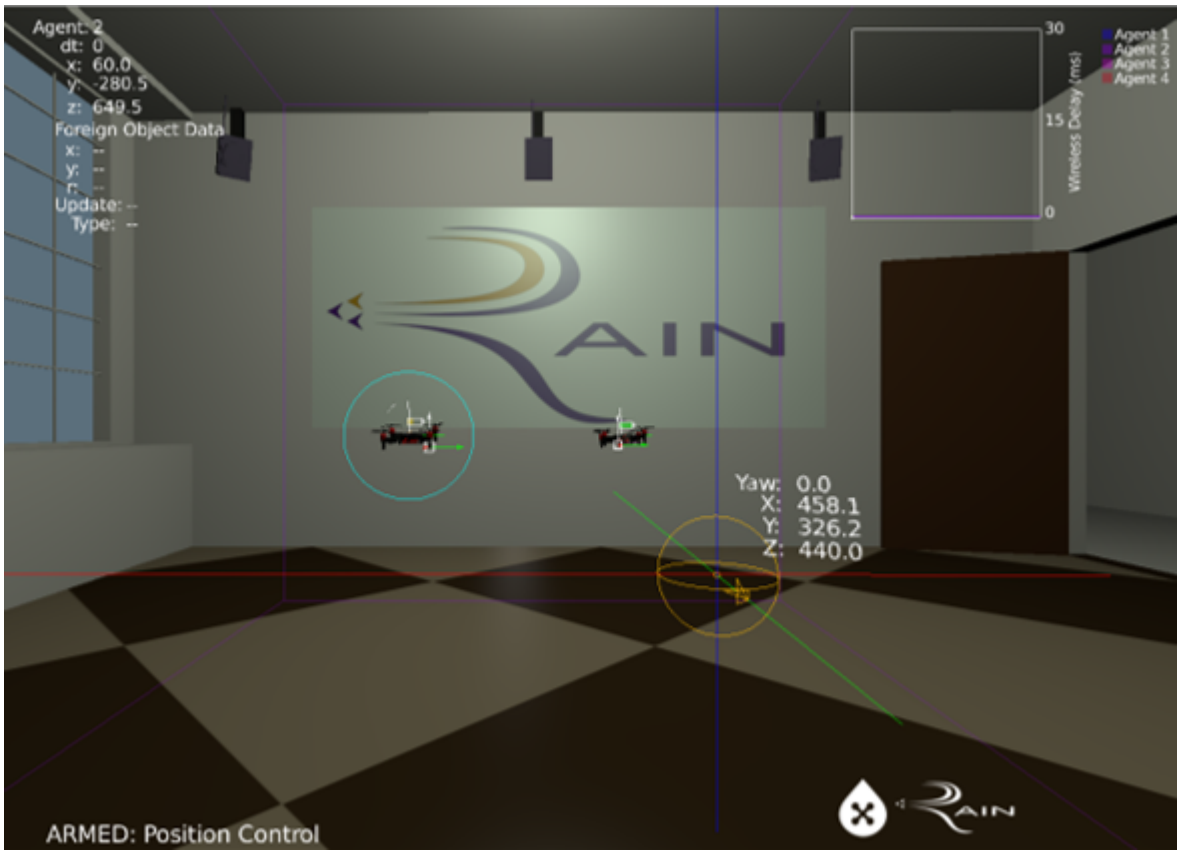

Figure 4.12: Screen-shot of the RAIN-DROP position control GUI. Return telemetry is passed back from the quadrotors, and updated in real-time for intuitive user control of RAIN-DROPs in the flight space. The rendered environment and models are to-scale, and the system is compatible with Virtual-Reality displays.

4.2.5 Software and Hardware Interlocks

Safety is always a concern when designing any robotic system. This extends not only to the physical safety of the users, but additionally to that of the hardware itself. This section takes into consideration some of the conditions in which the quadrotor issues a shut-down command to protect either itself or the users from harm.

Emergency Machine Off (EMO): This is a user initiated command, which may be issued in the event of an emergency or hazardous condition. The command places all agents in a *disarmed state*, immediately shutting down their motors regardless of their position within the flight space.

Inverted Condition: This condition triggers automatically when the quadrotor measured angle exceeds $\pm 90^o$ in either the ϕ or θ axis. Once triggered, the quadrotor will shut down until being disarmed and rearmed.

Low Battery: LiPo batteries have a minimum resting voltage, before incurring long-term damage to the battery capacity. For most LiPo batteries this voltage is ≈ 3.3 V/cell, therefore for a 2S battery, this is 6.6 V. Under-load, the minimum ADC corresponding to a safe resting voltage is 132ADC. Therefore if the voltage ADC measurement is read below this safety threshold for more than 300 cycles (≈ 2 s), the low-battery condition is triggered. In this condition the quadrotor motors are shut down, and the system needs to be fully restarted to clear the flag.

Wireless Communication Time-out: A persistent watchdog timer monitors the incoming data stream for the agents within the flight space. This timer is reset for each agent once they receive an update addressed to that agent. If the timer grows beyond 1 s, the system enters an EMO condition that is cleared once wireless communication has been re-established. In addition, all estimation, destination, and errors are cleared to prevent the system from issuing an unexpected control action, or retaining corrupted data.

SPI Communication Time-out: The communication pathway between the Navigator and the Pilot is monitored by a watchdog timer on the Pilot. This timer is reset whenever a complete package without errors has been received. If the watchdog timer grows beyond 0.5 s, the system shuts down, until SPI communication has been reestablished.

User-interface Communication Time-out: This condition is monitored on the SAMD21 ground-station transmitter. A watchdog timer monitors the time since the last update from the user interface. If this timer grows beyond 2 s, then a system-wide EMO command is issued. This is done to ensure that if the GUI has closed or locked-up, the agents are shut down for a safe recovery by the user.

4.3 Control

4.3.1 Final Control Architecture

The RAIN-DROP uses a cascade control architecture, which intakes a destination set by a user, and generates the relevant control inputs to the on-board motors of the quadrotor. Figure 4.13 shows the final RAIN-DROP control architecture. The core concept in this cascade structure is the natural partitioning of inertial and attitude control responsibilities into navigation and execution. The *Navigator* handles all inertial trajectory, estimation, and control responsibilities, including formation control and wireless communication. This then leaves the *Pilot* controller free to maintain platform stability while waiting for updated attitude and thrust set-points from the Navigator.

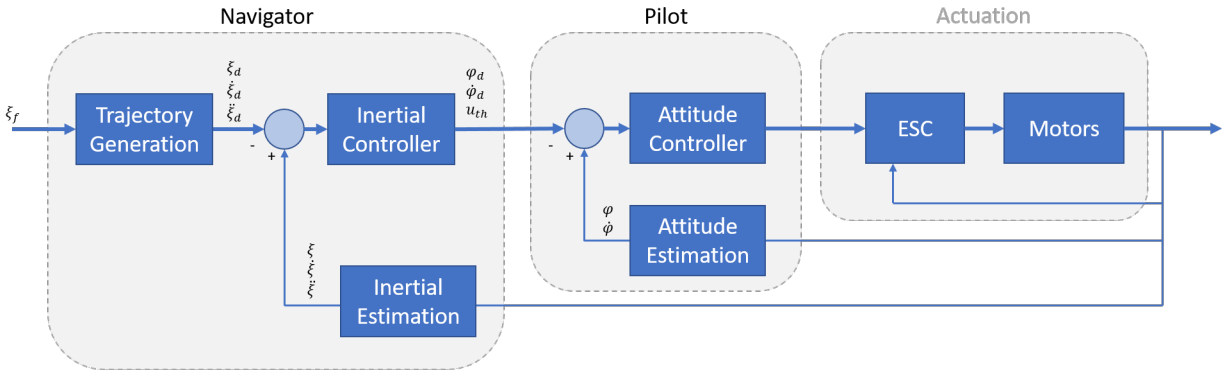

Figure 4.13: Graphical representation of the where each element of the control and estimation scheme is located in the cascade structure. The grey borders indicate the bounded items are handled by the same microprocessor. As the ESCs are purchased off-the-shelf, they are not considered as part of the cascade control, but rather the actuator itself.

4.3.2 Final Attitude Controller

The RAIN-DROP attitude control uses the PD control policy outlined in chapter 2, equation (2.17),

$$u_r = -M^T \begin{bmatrix} K_{\phi,p} & 0 & 0 & K_{\phi,d} & 0 & 0 \\ 0 & K_{\theta,p} & 0 & 0 & K_{\theta,d} & 0 \\ 0 & 0 & K_{\psi,p} & 0 & 0 & K_{\psi,d} \end{bmatrix} \begin{bmatrix} \varphi - \varphi_d \\ \dot{\varphi} - \dot{\varphi}_d \end{bmatrix}$$

The final tuned attitude control matrix of the RAIN-DROP is,

$$K_r = M^T \begin{bmatrix} 0.2875 & 0 & 0 & 0.375 & 0 & 0 \\ 0 & 0.2875 & 0 & 0 & 0.375 & 0 \\ 0 & 0 & 2.5 & 0 & 0 & 1.25 \end{bmatrix}$$

Another attitude control consideration is addressing the yaw discontinuity. Since the platform can rotate freely about the z-axis, the discontinuity at $\pm 180^o$ must be accounted for by adjusting the error when the discontinuity appears in the error,

$$e_\psi = \begin{Bmatrix} \psi - \psi_d & if & -180 \leq \psi - \psi_d < 180 \\ \psi - \psi_d - 360 & if & \psi - \psi_d \geq 180 \\ \psi - \psi_d + 360 & if & \psi - \psi_d < -180 \end{Bmatrix}$$

where the then angle ψ is bounded $\psi \in [-180, 180)$.

As mentioned in chapter 2, the gain-tuning approach offers a relatively quick means to set up a stabilizing controller for the attitude control. In order to garner a better understanding of how close these gains are to becoming unstable, we can plot the roots of the attitude closed-loop transfer function. Shown in figures 4.14 and 4.15 are plots of the closed-loop poles of the closed-loop transfer function for a range of $K_{\varphi,p}$ and $K_{\varphi,d}$. As we can see, both the current proportional and derivative gains for the roll and pitch axes are within the stability margin, and have some room to be increased and decreased if needed.

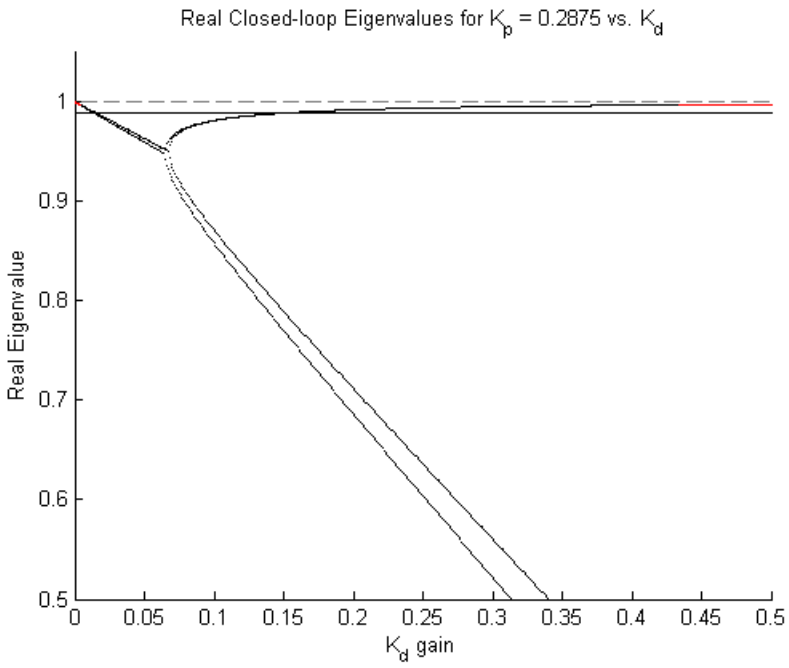

Figure 4.14: Figure showing the plot of the closed-loop poles for the linearized attitude response for a range of derivative gains $K_{\phi,d}$ and $K_{\theta,d}$, discretized at $\Delta t = 0.0066$ s, and the proportional gains are held constant at their tuned values. Roots $|\lambda| \geq 0.996$ are highlighted in red to indicate roots that are becoming close to unstable ($\lambda \geq 1$) for practical application. Additionally the yaw gains were both held constant at their tuned values.

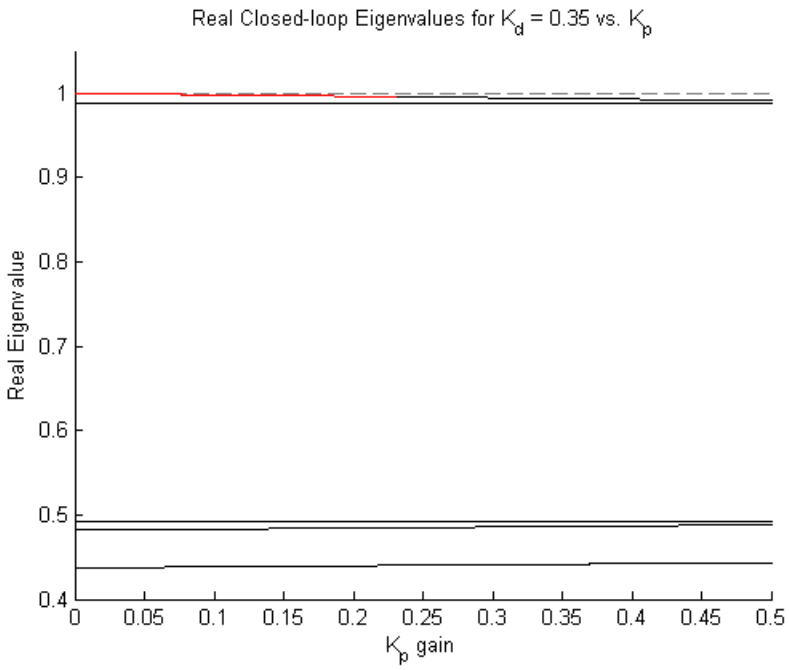

Figure 4.15: Figure showing the plot of the closed-loop poles for the linearized attitude response for a range of proportional gains $K_{\phi,p}$ and $K_{\theta,p}$, discretized at $\Delta t = 0.0066$ s, and the derivative gains are held constant at their tuned values. Roots $|\lambda| \geq 0.996$ are highlighted in red to indicate roots that are becoming close to unstable ($\lambda \geq 1$) for practical application. Additionally the yaw gains were both held constant at their tuned values.

Control Saturation: On a practical level, the platform can only execute the control bandwidth provided by the actuators. On a control-response level, a high-gain stabilizing controller can produce an unstable responses if requested control is too high, or induces a response outside the capacity of its actuators. Saturation limits are a way to both prevent over-reaction of the system, and ensure that the system response stays within a desired limit.

For the RAIN-DROP the maximum force output per motor was measured to be $F_{i,max} = 0.69$ N/motor. As each motor can only provide a positive thrust our motor thrust bandwidth is then,

$$u_{r,i} \in \begin{bmatrix} 0, & 0.69 \end{bmatrix} \text{ N}$$

Since negative control actions are enforced by reduction of motor speed for a given pair, this means that for a controller acting about a given axis only has the bandwidth of the maximum thrust provided by 2 motors. Meaning the bandwidth for the total attitude control response is,

$$c \in \begin{bmatrix} -0.69, & 0.69 \end{bmatrix} \text{ N}$$

These will serve as our maximum attitude control saturation limits. The final attitude control constraints for each motor on the RAIN-DROP were tuned to,

$$c = \pm 0.6 \text{ N}$$

for each controller.

4.3.3 Final Position Controller

The position control policy used by the RAIN-DROP is the velocity PID developed in chapter 2, equation (2.18), with an integral on the z-axis. The final K matrix used by the RAIN-DROP inertial controller is,

$$K = H \begin{bmatrix} 0.03125 & 0 & 0 & 0.03125 & 0 & 0 & 0.01 & 0 & 0 \\ 0 & 0.03125 & 0 & 0 & 0.03125 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0.18 & 0 & 0 & 0.01 \end{bmatrix}$$

When the platform rotates about it's z-axis, the H matrix will change how the control gains are applied to the globally defined inertial values. However, as was found in chapter 2, equation (2.16), the x-y inertial error itself can be rotated about the z-axis to match that of the quadrotor body-frame. The RAIN-DROP uses this error rotation to use a static control matrix, and will prove useful for control algorithms which produce a control response for a given yaw.

Lastly the z-axis integral control gain used by the RAIN-DROP is,

$$K_{z,i} = 2.5$$

Determined primarily through direct experimentation.

Control Saturation: Saturating the inertial control values ensures that that the platform is able to maintain hover, and additionally bound the induced angle of the control response. For the RAIN-DROP platform, the maximum thrust for the platform is $F_{max} = 2.75$ N. In order to maintain hover,

$$F_H = mg = 0.129 \text{ kg} * 9.815 \frac{\text{m}}{\text{s}^2} = 1.266 \text{ N}$$

Therefore the maximum possible angle from vertical for the platform to achieve while maintaining enough thrust to hover is,

$$\alpha_{max} = acos\left(\frac{1.266 \text{ N}}{2.75 \text{ N}}\right) = 62.59^o$$

This corresponds to a total norm max translational control response of,

$$c_{xy,max} = 2.75 \text{ N} * \sin(\alpha_{max}) = 2.44 \text{ N}$$

However, we must also take into consideration that our inertial control policy operates off an implicit small-angle approximation, by considering the inertial and attitude responses to be independent. As the angle of the platform deviates from hover, the approximation error increases, thereby resulting in non-trivial coupling between the inertial and attitude state responses. Figure 4.16 shows a plot of the small angle approximation error for a generic angle α.

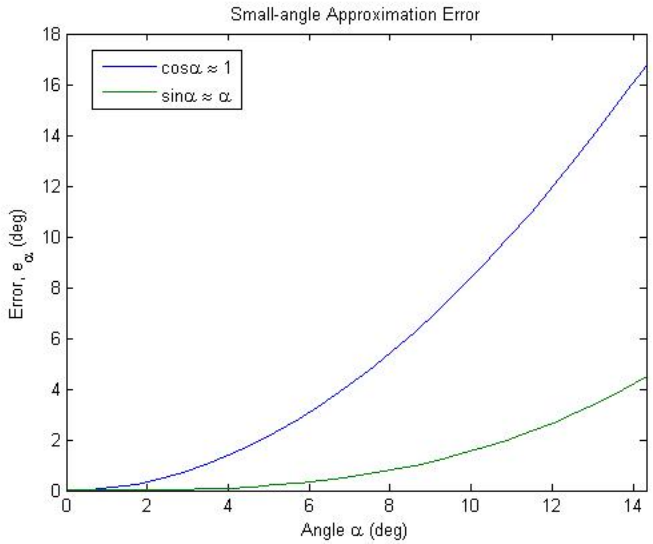

Figure 4.16: Plot of the error associated with the small angle approximation $cos\alpha \approx 1$ and $sin\alpha \approx \alpha$.

Supposing we want to bound the approximation error within $e_\alpha \pm 6^o$, then we need to ensure that the translational thrust response does not result in $\alpha >\sim 10^o$. Therefore, our maximum inertial control response associated with translation should be bounded within,

$$c_{xy} = \pm 2.75\text{N} * \sin(10^o) = \pm 0.4775\text{N}$$

For the z-control response, the maximum thrust available for increase is $F_{max} - F_H = 1.483$ N. However, to ensure the quadrotor motors are not spun down to 0, a minimum thrust level must be maintained. For the RAIN-DROP it was decided that the z-controller would have authority over $\sim 20\%$ of the hover thrust. Meaning,

$$c_z = \pm 0.2 F_H = 0.253\text{N} \approx 0.3\text{N}$$

Checking that our translational constraints and z constraints are compatible, the angle that would be produced from the requested translational thrust at constraint would be,

$$\alpha = asin\left(\frac{0.4775}{1.266 + 0.3}\right) = 14.82^o$$

Which is greater than our desired 10^o limit. To address this, we can tighten our translational control constraint to,

$$c_{xy} = 0.3\text{N}$$

Which results in our angle being bounded,

$$\alpha = asin\left(\frac{0.3}{1.266 + 0.3}\right) = 9.25^o$$

Application of these bound on the RAIN-DROP is done by bounding the control response for each respective axis. This was primarily done for implementation simplicity, however this results in the bound needing to be more conservative since the constraint limits. The defined action constraint cube would then need to be small enough to be contained entirely within the action space allowed by the actual constraining ellipsoid. Figure 4.17 shows a graphical representation of the constrained action space. More specifically for the RAIN-DROP, this means that combined x_{max}, y_{max} translations will produce a larger angle α than allowed by our constraints. However, as the $\sim 10^o$ is an approximate constraint, therefore we can assume that our response is sufficient for our current purposes, and tighten further if needed.

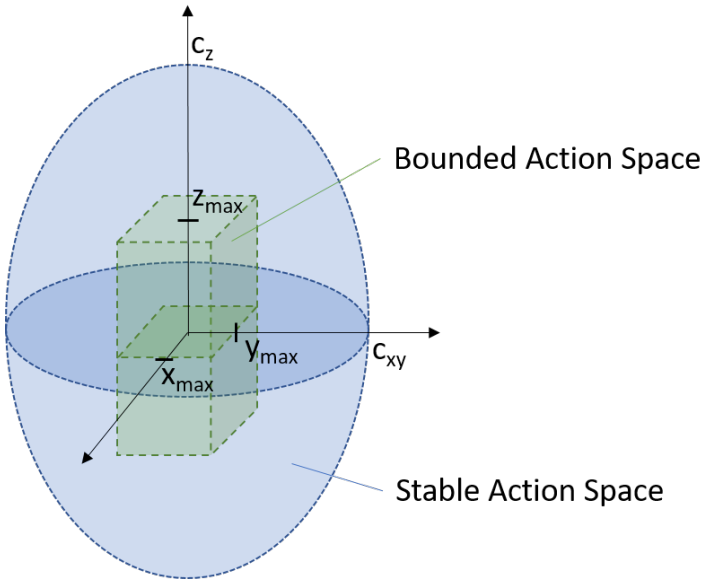

Figure 4.17: Illustration of the use of axis-specific constraints. As seen in the diagram, the axis-specific constraints must be tighter to ensure the control response within all directions is bounded by the hover and translational constraints.

4.3.4 Final Inertial Trajectory Protocol

The trajectory policy used by the RAIN-DROP was developed and tabulated in chapter 2, table 2.1. It was found to be useful to add adjustable trajectory tracking gains for each trajectory set-point. This affords an easy means to effectively adjust the control response of the static K matrix during different stages of motion, without needing to modify the K matrix itself. In particular the static K matrix for the inertial control may produce a good response for holding position, but may also be poorly suited for smooth translation.

As such, we can influence the entries of the static control gain and account for model inaccuracies by introducing trajectory tuning parameters that adjust the reference point as needed to achieve a desired output response. Introduction of these tuning parameters is

shown table 4.4. Each tuning parameter for a reference of interest m is denoted as,

$$k_{m,\xi} = \begin{bmatrix} k_{m,x} \\ k_{m,y} \\ k_{m,z} \end{bmatrix}$$

Figure 4.18 shows the effect of the tracking gain on the velocity profile of the live system compared to the simulated set-point. The zero-spikes are artifacts due to post-processing estimation of the data, and not observed on the actual system estimator.

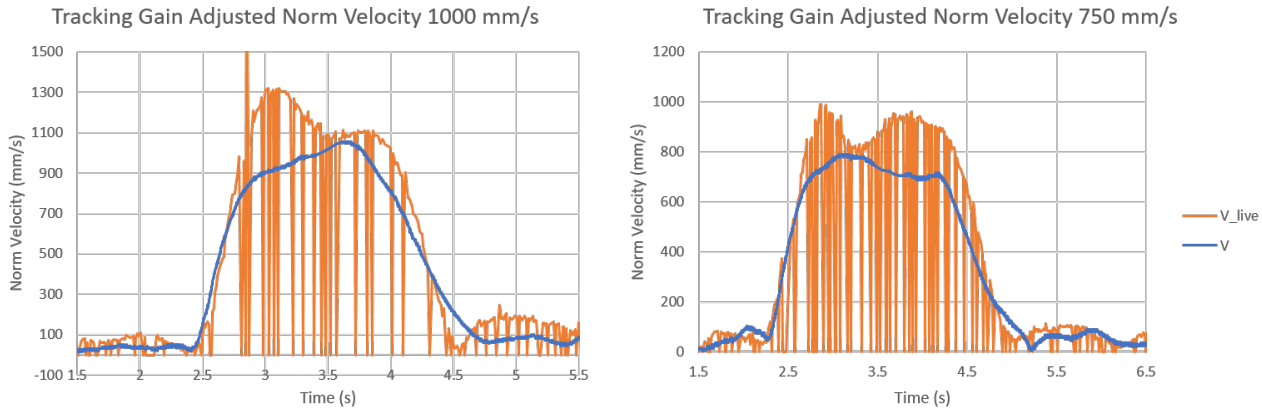

Figure 4.18: Plot of the measured and simulated norm velocity response of a quadrotor using the defined inertial trajectory generation policy with a wind-down region of $r_{wd} = 600$ mm. (Left) Measured and simulated norm velocities with $k_{v,\xi} = 1.35$ applied to the live system, with a $v_{max} = 1000$ mm/s. (Right) $v_{max} = 750$ mm/s. Neither simulated trajectory uses the tracking gain.

Where m is a generic placeholder for the desired value being adjusted (position, velocity, acceleration). The final trajectory tracking gains used by the RAIN-DROP were experimentally determined through a coarse tuning process,

$$k_{p,\xi} = \begin{bmatrix} 1.0 \\ 1.0 \\ 1.0 \end{bmatrix}, \quad k_{v,\xi} = \begin{bmatrix} 1.35 \\ 1.35 \\ 1.35 \end{bmatrix}$$

Where the max velocity and dead-band were set as $v_{max} = 1000$ mm/s and $r_{db} = 100$ mm, respectively. The beginning of the wind-down region was determined via simulation. Figure 4.19 shows the trajectory protocol simulated without any wind-down region, which produces the expected oscillatory response.

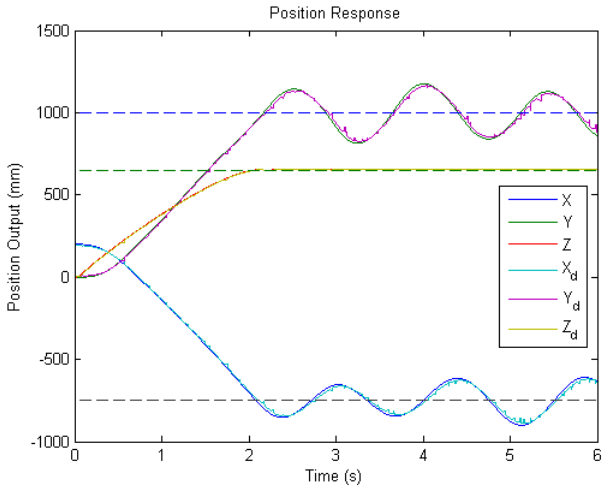

Figure 4.19: Plot of the simulated position response of a quadrotor use the defined inertial trajectory generation policy with no wind-down region, where $r_{db} = 100$ mm and $v_{max} = 1000$ mm/s.

We can take note that the simulated platform takes approximately 0.6 s to reach the maximum velocity set-point. Using this response timing, we can determine wind-down distance required for the current controller to reach 0,

$$r_{wd} = t_{wd} v_{max}$$

where t_{wd} is the minimum time required for the system to stop from a given velocity v_{max}. For $v_{max} = 1000$ mm/s and $t_{wd} = 0.6$, our wind-down region becomes,

$$r_{wd} = 600\text{mm}$$

Figure 4.20 shows the response of the updated trajectory policy using the wind-down region.

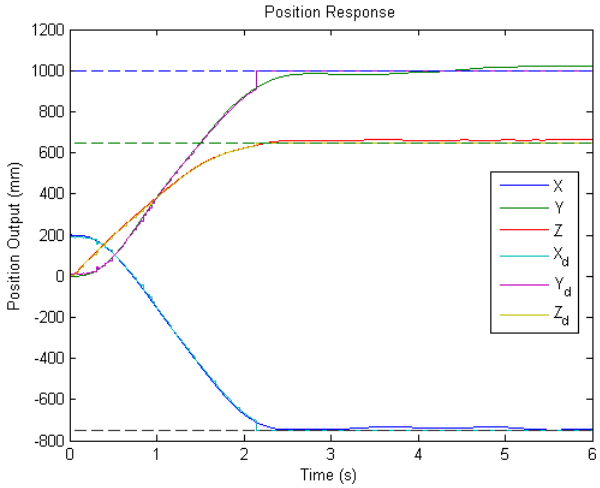

Figure 4.20: Plot of the simulated position response of a quadrotor use the defined inertial trajectory generation policy with a wind-down region $r_{wd} = 600$ mm, where $r_{db} = 100$ mm and $v_{max} = 1000$ mm/s.

Final Velocity and Position Protocol with Tuning Parameters

Transit	**for:** $(r_t > r_{wd})$	$\dot{\xi}_d(t) = k_{v,\xi} v_{max} \frac{\xi_f - \xi(t)}{r_t}$ $\xi_d(t) = \xi(t) + k_{p,\xi} dt(\dot{\xi}_d(t) - \dot{\xi}(t))$
Wind-down	**for:** $(r_{db} \leq r_t \leq r_{wd})$	$\dot{\xi}_d(t) = k_{v,\xi} \frac{v_{max}}{r_{wd}} \frac{\xi_f - \xi(t)}{r_t}$ $\xi_d(t) = \xi(t) + k_{p,\xi} dt(\dot{\xi}_d(t) - \dot{\xi}(t))$
Position Hold	**for:** $(0 \leq r_t < r_{db})$	$\dot{\xi}_d(t) = k_{v,\xi} \frac{v_{max}}{r_{wd}} \frac{\xi_f - \xi(t)}{r_t}$ $\xi_d(t) = \xi_f$

Final Acceleration Protocol with Tuning Parameters

for: $(r_t \leq r_{db})$	$\ddot{\xi}_d(t) = 0,$

Table 4.4: Tabulated final trajectory policy for position, velocity, and acceleration with tuning parameters, where $\xi = (x, y, z)^T$. The fundamental goal is to generate a continuously viable trajectory towards a global destination $\xi_f = (x_f, y_f, z_f)^T$, each tuning parameter $k_{m,\xi}$, corresponds to some parameter acting on the acceleration, velocity or position terms, specified for each axis element in ξ. The RAIN-DROP currently uses $r_{db} = 100$ mm, $r_{wd} = 0.6 v_{max} = 600$ mm and $v_{max} = 1000$ mm/s.

4.3.5 Final Attitude Trajectory Protocol

The current output of the inertial control policy used by the RAIN-DROP is the thrust input for each motor. This was originally done so the inertial control values could be superimposed over the attitude control, to mimic the summation of a general K matrix (of the form $u = -Kx$). However, as discussed in chapter 2, using superposition results in the attitude controller fighting against the inertial control commands, due to the reference point of the attitude and rate being zero.

Instead we can leverage the attitude controller to enforce inertial control actions by using the integration approach developed in chapter 2, equations (2.23) and (2.22). The only practical modification that was made for implementation, was the integration of the desired angle. Instead of integrating from the previous desired angle, we assume that the platform

is tracking the rate and angle such that $\varphi(t) = \varphi_d(t-1)$. This modification results in the final attitude trajectory policy,

$$\dot{\varphi}_d(t) = dt\ddot{\varphi}_d(t) \tag{4.1}$$

$$\varphi_d(t) = \varphi(t) + dt\dot{\varphi}_d(t) \tag{4.2}$$

Where the desired thrust bias for each motor is the average of the computed inertial thrust values,

$$u_{th} = \frac{1}{4}\sum_{i=0}^{4} u_i \tag{4.3}$$

4.4 Final Formation Control

4.4.1 Unit-vector Consensus and Formation Cohesion

The RAIN-DROP formation control uses the unit-vector consensus protocol outlined in chapter 2, equation (2.44) for position placement and orientation around a user-specified destination. Formation cohesion is then maintained using the PD consensus control approach developed in chapter 2, equation (2.45). The unit vector consensus protocol us used to leverage the existing inertial trajectory protocol to track towards a desired location around the common destination. Figure 4.21 shows this protocol in action with a formation of $n = 3$ agents, and a uniform desired formation radius of $r_{d,i} = 200$ mm around the destination. As we can see in the figure, each of the agents follows along the standard linear trajectory produced by the inertial trajectory policy, and the agents converge to their placements around the destination. While the agents do indeed arrive at the desired inter-agent spacing using unit-vector consensus, we can also note that this spacing is inconsistent during transit. This is expected, as the inertial trajectory generation in this configuration does not account for the states of the other agents when approaching the agent-specific final destination.

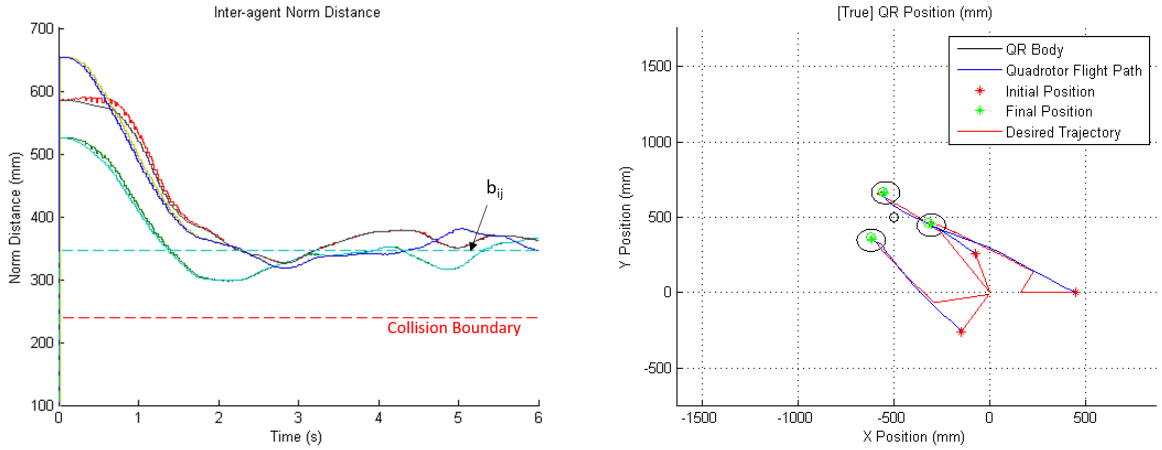

Figure 4.21: Simulation results of $n = 3$ quadrotors initialized at a different formation spacing, using the unit-vector consensus protocol without cohesion forcing. The formation radius is $r_d = 200$ mm for all agents. (Left) Plot of the norm distance between each agent. This shows formation following along the linear path defined by the inertial trajectory generation. Due to the lack of a cohesion term, the formation only reaches the final desired inter-agent spacing by virtue of each arriving at their specified destination around the common destination. (Right) Top-down view of formation during flight moving from $[0, 0, 0]$ mm to $[-500, 500, 500]$ mm.

The formation cohesion policy is then introduced to provide inter-agent forcing, which aims to maintain the formation shape en-route to the destination. Figure 4.22 shows a simulated result of the formation cohesion augmented unit-vector policy used by the RAIN-DROP, with a $K_{p,f} = 1.5$ and $K_{d,f} = 0.8$. The norm distance converges to the desired buffer distance approximately 1 s earlier than the unit vector only approach, and maintains that formation spacing throughout the remaining translation.

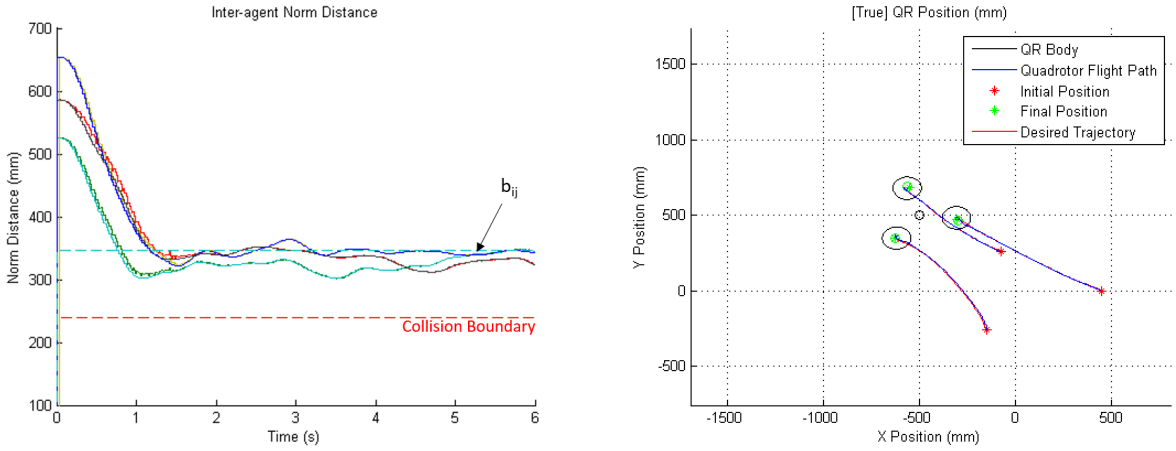

Figure 4.22: Simulation results of $n = 3$ quadrotors initialized at a different formation spacing, using the unit-vector consensus protocol with formation coheision forcing. (Left) Plot of the norm distance between each agent. The formation rapidly converges to the desired inter-agent spacing, where $r_d = 200$ mm defined for all agents. r_d is used to compute the desired buffer spacing show in the figure. (Right) Top-down view of formation during flight moving from $[0, 0, 0]$ mm to $[-500, 500, 500]$ mm.

4.4.2 Formation Orientation

For formation orientation, the RAIN-DROP currently selects the lowest index agent to be the pointing anchor of the formation. This configuration essentially defines a directed connection between the pointing agent and the neighboring agents for the destination placement, and leaves the cohesive forcing in place for all agents. Figure 4.23 shows a simulation where agent 1 initiates a rotation from $\psi_{form} = 0^o$ to 45^o at $t = 3$ s.

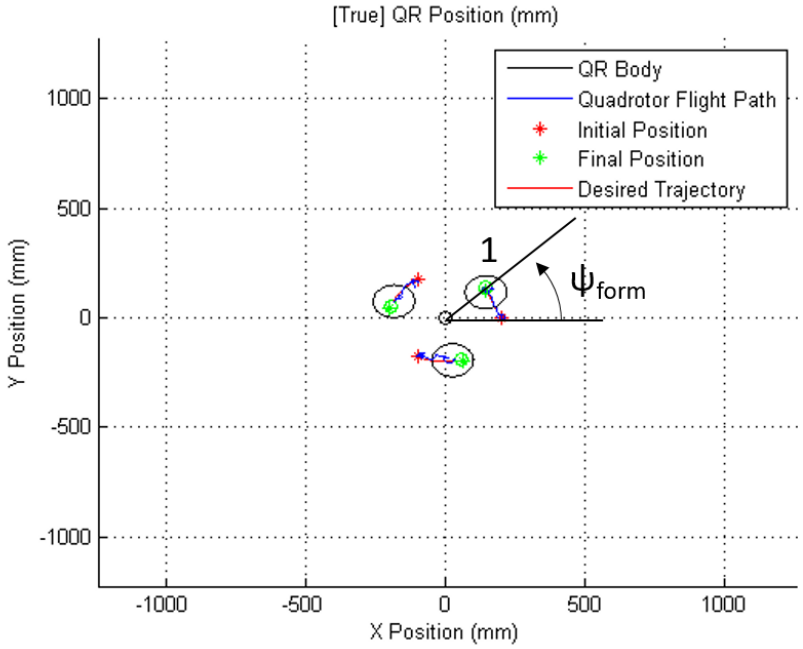

Figure 4.23: Simulation results of $n = 3$ quadrotors using the unit-vector consensus protocol with formation coheision forcing and orientation control. Plot is a top-down view of formation holding position, while rotating from $\psi_{form} = 0^o$ to 45^o at $t = 3$ s.

While this is a simple approach to implement, the chief drawback is that the lead agent is capable of quickly changing its set-point around the unit circle, which can result in collisions. The cohesion controller gain $K_{p,f}$ can be increased to account for this; however, due to model uncertainties and other simulation inconsistencies, there is a practical limit to how high this gain can be increased on the live system. Figure 4.24 shows the simulated response of two different $K_{p,f}$ gains for an instantaneous pointing angle change from $\psi_{form} = 0^o$ to 90^o. For the purposes of this thesis, we will stick to instantaneous changes in z-rotation $\Delta\psi_{form} \leq \pm 45^o$ when $n > 2$ to reduce the risk to the quadrotors during operation.

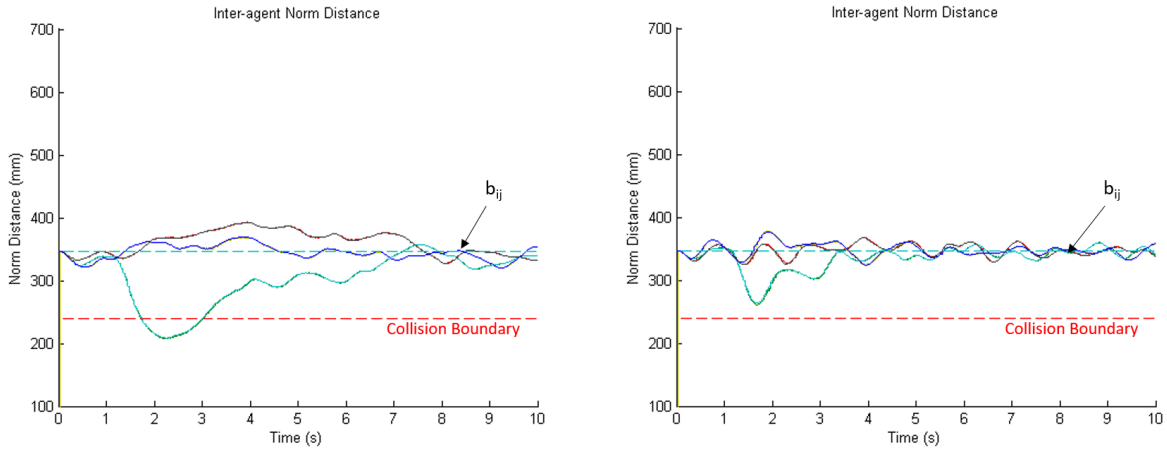

Figure 4.24: Simulation results of $n = 3$ quadrotors using the unit-vector consensus protocol with formation coheision forcing and orientation control rotating from $\psi_{form} = 0^o$ to 90^o at $t = 1$ s. (Left) Plot of the inter-agent norm distance with $K_{p,f} = 1.5$ and $K_{d,f} = 0.8$. We can see that after the rotation command is sent to agent 1, it collides with another agent unable to respond at a fast enough rate. (Right) Plot of the inter-agent norm distance with $K_{p,f} = 10$ and $K_{d,f} = 0.8$. Collision is avoided due to increased P gain, however the higher gain amplifies inter-agent oscillations.

4.4.3 Agent-specific Formation Placement

One of the attractive features of using the formation protocol developed in this thesis is that the radius of the agent with respect to the common destination can be unique to each agent. This agent-specific radius affords a simple means of control over the formation shape and distribution. Figure 4.25 shows the simulated response of a formation configuration where each agent is at a unique radius from the desired common destination.

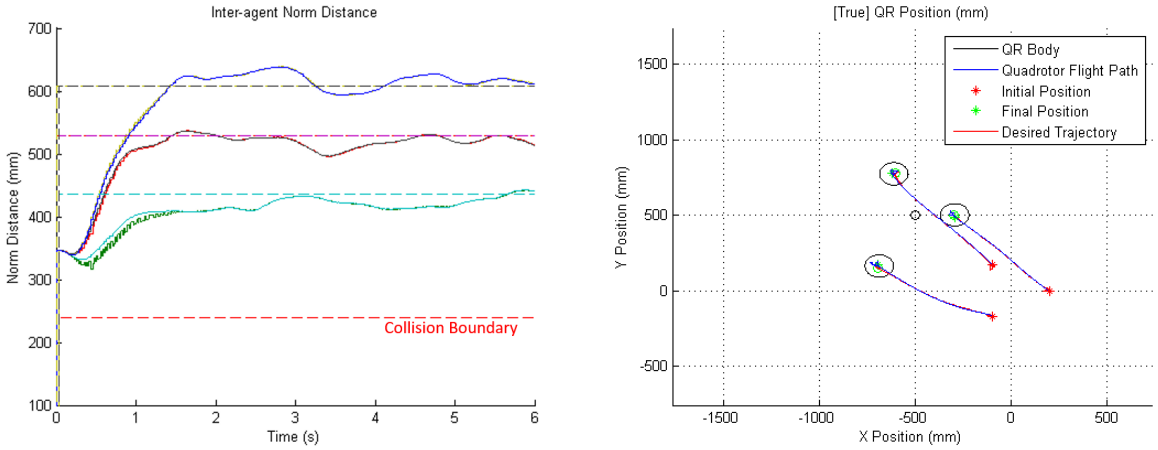

Figure 4.25: Simulation results of $n = 3$ quadrotors using the formation protocol developed in this section where each agent is at a unique radial set-point from a common destination, $r_{form} = (200, 300, 400)$ mm. Maneuver was executed in transit to the destination. (Left) Norm distance between each agent converges to the desired buffer distance dictated by the radial placement of each agent. (Right) Trajectory output of the formation.

4.5 Paylaod

4.5.1 Point-mass Payload

Figure 4.26: Conceptual rendering of the final point-mass payload attachment scheme for $n = 3$ agents.

In order to best physically mimic the point-mass payload scenario, we developed a simple magnetic anchoring system that attaches to a 1" steel ball bearing. Figure 4.26 shows a rendered mock-up of the final point-mass system. The use of spherical payload, allows for the magnetic anchors to slide into place as the supporting agents move about the payload. This added degree of freedom, allows the tension vector to approximately act on the center of mass of the payload. Figure 4.27 illustrates the ideal case where the magnetic anchor re-positions itself in response to a tension vector change in the absence of friction.

In reality, friction between the magnet and bearing surface will prevent the anchor from sliding into the ideal location, and therefore the tension vector will not point directly through the center of the mass of the payload. Figure 4.28 shows a photograph of the final

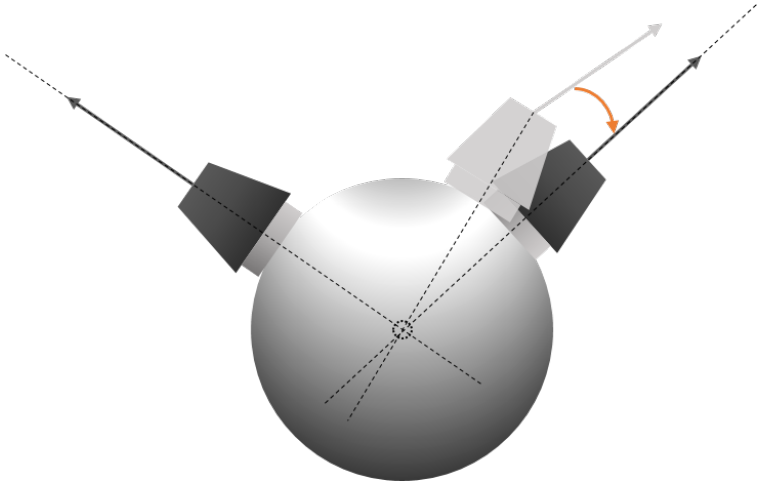

Figure 4.27: Figure illustrating the ideal sliding of a magnetic anchor in response to a change in tension vector acting on the anchor without friction. The anchor then slides along the surface until the normal vector is aligned with the tension vector acting on the anchor.

RAIN-DROP anchor mechanism for $n = 3$ agents acting on a point-mass, where the effects of friction are more apparent.

Figure 4.28: Photograph of the final point-mass payload attachment scheme for $n = 3$ agents. We can note that the tension vector is slightly skewed from acting through the center of mass of the payload, as friction prevents the magnetic anchor from sliding into the ideal placement.

4.5.2 Rigid-body Payload

Figure 4.29: Conceptual rendering of the final rigid-body payload assembly in its default configuration for $n = 3$ agents.

The RAIN-DROP rigid body payload was designed to offer a wide array of configurations for the research team to utilize. The core concept was to design a system that uses the same magnetic anchoring mechanism as the point-mass system, but with adjustable fixed anchor points, as opposed to the surface of a magnetic sphere. Sockets for multiple ball-bearing payloads and adjustable support arms were specifically designed to allow for future research endeavors into transportation of a payload with eccentric loading or asymmetric anchor distribution. Figure 4.30 illustrates configuration changes able to be made by the rigid body assembly.

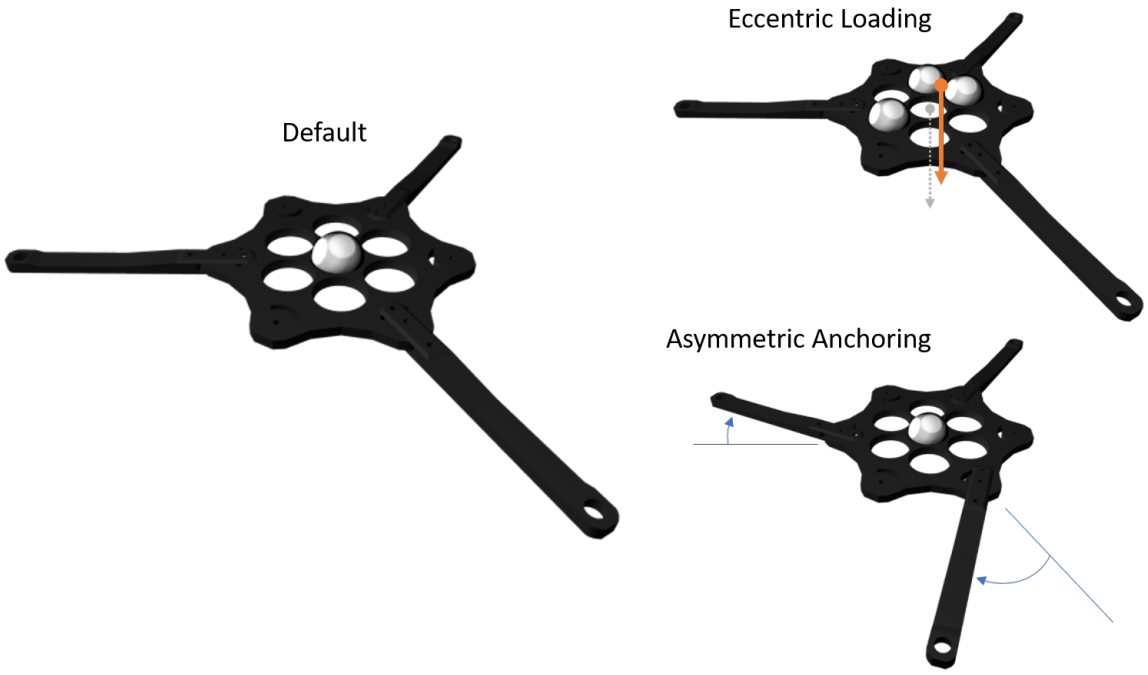

Figure 4.30: A 3D rendering of some of the configuration changes that can be made to the RAIN-DROP rigid-body payload assembly.

Chapter 5

FINAL IMPLEMENTATION AND FLIGHT PERFORMANCE VALIDATION

The fundamental purpose of this chapter is to compare the simulation suite discussed in chapter 3, against the actual dynamics of final system outlined in chapter 4. As the simulation suite is to be used for testing a variety of algorithms prior to implementation, we will seek to reconcile or explain any deviations observed between the measured and simulated results.

5.1 Single Agent

5.1.1 Rest-to-rest Translation

A simple experiment was performed to compare the simulated and measured performance of the RAIN-DROP. The experiment was a translational maneuver from one point in the flight space, to another at two different v_{max} set-points. Figure 5.1 shows the plotted position and trajectory telemetry of the RAIN-DROP and simulated quadrotor moving at $v_{max} = 1000$ mm/s and $v_{max} = 750$ mm/s. As we can see in both cases the simulated quadrotor arrives at the target destination ahead of the RAIN-DROP.

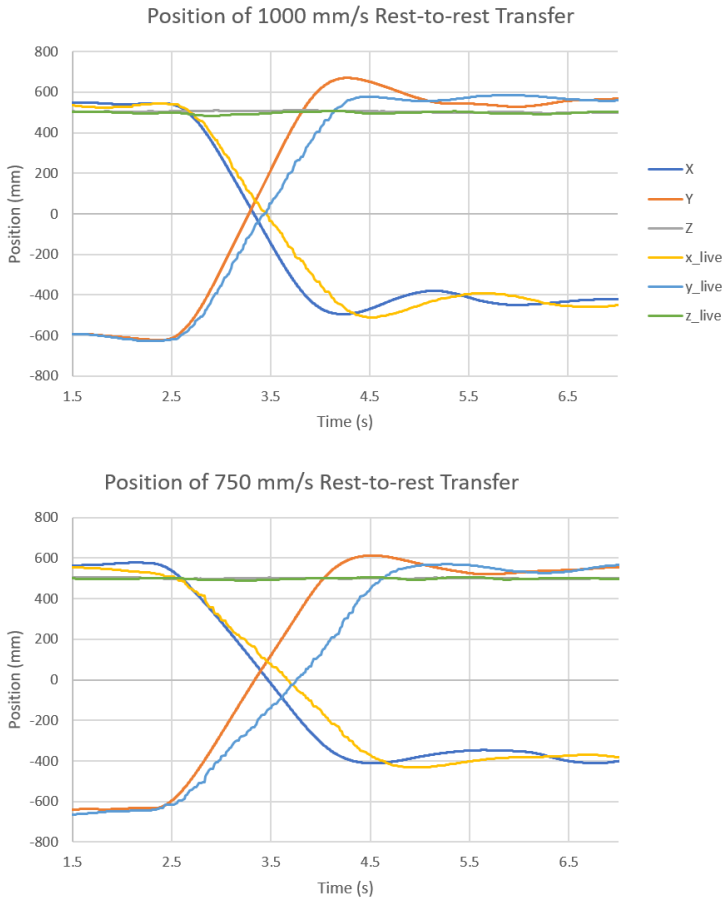

Figure 5.1: Plots showing the simulated and measured trajectories for a rest-to-rest maneuver performed different velocities, at a constant altitude. Simulation run using the tuned RAIN-DROP trajectory matching gain ($K_{v,\xi} = 1.35$). (Top) $v_{max} = 1000$ mm/s. (Bottom) $v_{max} = 750$ mm/s.

Recall that the RAIN-DROP uses a tuned trajectory tracking gain that matches the RAIN-DROP max velocity performance to the desired set-point. This gain serves to account for any assumptions made by the simulation that would result in an insufficient maximum velocity. This implies then that our platform has been tuned to match our simulation without the tracking gain. Therefore we can negate this gain within the

simulation by applying a validation gain to the trajectory tracking function,

$$k_{v,val\xi} = \frac{1}{k_{v,\xi}}$$

This then negates the effects of the current gain, but allows the user to modify the tracking gain in simulation to observe the effects as if it was on the live system. Figure 5.2 shows that negation of the tuned tracking gain returns the expected matching steady-state behavior.

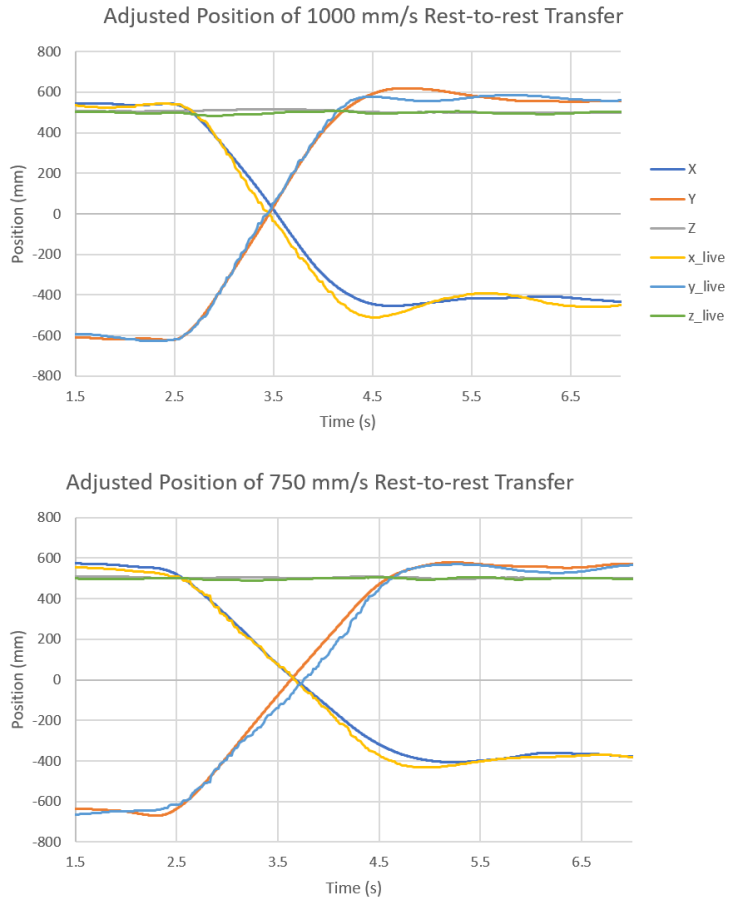

Figure 5.2: Plots showing the simulated and measured trajectories for a rest-to-rest maneuver performed different velocities, at a constant altitude. Simulation negating the tuned RAIN-DROP trajectory matching gain ($K_{v,\xi} = 1.35$) with $k_{v,val\xi} = \frac{1}{1.35}$. (Top) $v_{max} = 1000$ mm/s. (Bottom) $v_{max} = 750$ mm/s.

The last major component of interest is the attitude performance during translation, in particular the roll-pitch dynamics. Figure 5.3 shows the comparison between the simulated and measured RAIN-DROP telemetry, before and after introduction of the validation gain $k_{v,val\xi}$. The plotted simulation data is that of the noise-influenced true rotational state of the simulated quadrotor. This was done to help get a better sense of what the live system is doing, as the measured rotational data from the RAIN-DROP has a large degree of noise intrusion, which is also encountered within the simulated measured value. The first feature to notice is the attitude dip at 2.5 s, where the translational maneuver is initiated. The maximum norm angle produced from the maneuver is close to the translation attitude constraint we were seeking to enforce of around $\alpha <\sim 10^o$ in each axis. The negation of the validation gain shifted the secondary peak from 3.75 s to 4.5 s, which better matches the live pitch data, and corresponds to the end of the maneuver in figure 5.2. Unfortunately, it is difficult to discern whether or not the roll data deviations are due to noise intrusion or the platform itself. There are notable oscillations observed at the end of the maneuver in the x-live response, which is governed by the roll-dynamics (at zero yaw). These oscillations suggest that the roll would indeed see a dip, but not a persistent bias.

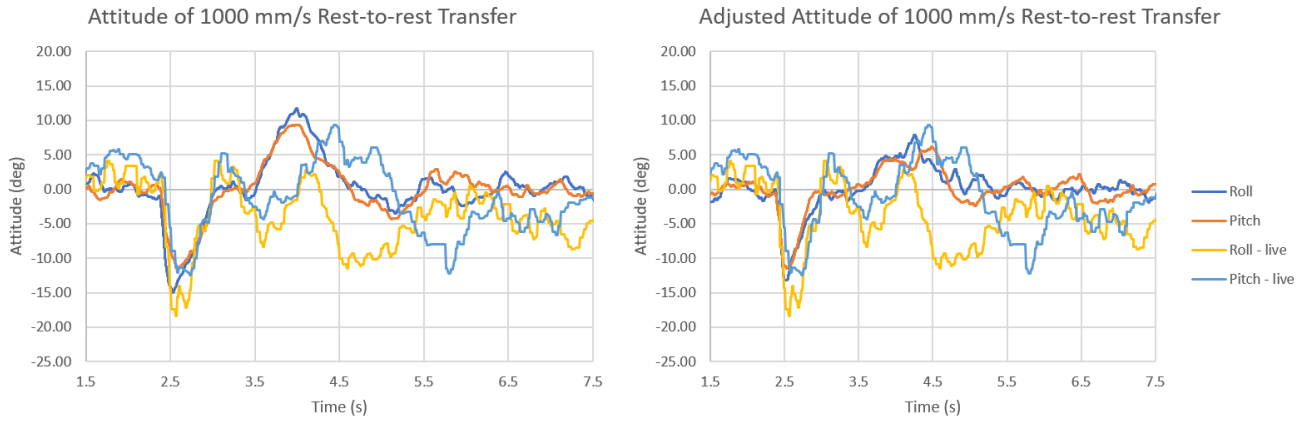

Figure 5.3: Plots showing the simulated and measured attitude information for the rest-to-rest translation maneuver performed at $v_{max} = 1000$ mm/s. (Left) Simulation run using the tuned RAIN-DROP trajectory matching gain ($K_{v,\xi} = 1.35$) with $k_{v,val\xi} = 1$. (Right) Simulation run while negating the tuned RAIN-DROP trajectory matching gain ($K_{v,\xi} = 1.35$) with $k_{v,val\xi} = \frac{1}{1.35}$.

Another point of comparison between the live and simulated angles is the persistent higher-frequency ($5 \sim 6$ Hz) oscillations observed in the live data. These suggest that the derivative response is over-active, where a delay in the actuation results in a delay in the correction of the rate of change. Checking this, we can suppose the delay behaves like a low-pass filter, where the response is delayed but approaching the final value,

$$u(t+1) = k_{motor} u(t) + (1 - k_{motor})\, u(t-1)$$

Figure 5.4 shows the effect of introducing this style of delay into the actuation dynamics has on the roll angle, where $k_{motor} = 0.05$. As suspected, we see the intrusion of higher-frequency oscillations into our attitude response. If this delay is increased, the oscillations become more pronounced.

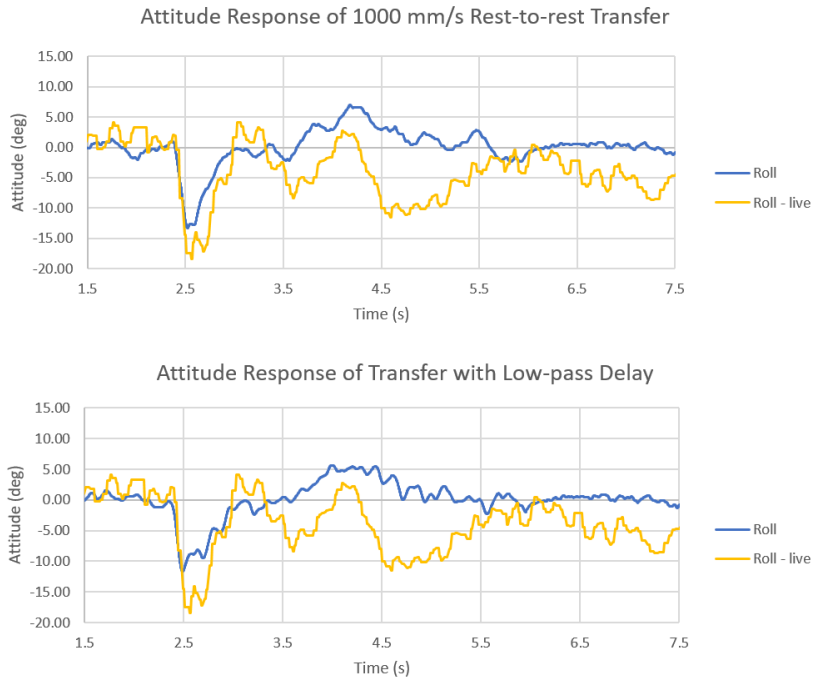

Figure 5.4: Plots showing the simulated and measured attitude information for the rest-to-rest translation maneuver performed at $v_{max} = 1000$ mm/s. (Left) Simulated and measured roll angles during the maneuver under instant-actuation assumption ($k_{motor} = 1$). (Right) Simulated and measured roll angles with introduction of a low-pass delay on the motor actuation where $k_{motor} = 0.05$.

5.2 Multi-agent Formation Flight

5.2.1 Formation Rotation 2 Agents

The formation controller was designed to maintain a desired distance between agents, while being able to control the orientation of the formation using an agent assigned as a pointer. There are a few challenges presented when attempting to validate the flight data for these systems. Namely, the telemetry feed may have inconsistent updates from one agent or another, due to wireless transmission write-collisions. As such, a video was used to match up the timing of actions taken in the simulation with the actions of the actual RAIN-DROPs. Figure 5.5 shows the comparison plots between the simulated and

measured dynamics of two agents flying in a formation. As the formation control is fundamentally using the same trajectory protocol as global inertial protocol, the simulation maintains good agreement with the position dynamics of the live platforms. Many of the delays in initiation of maneuvers can be attributed to the timing approximation using video. The key feature of interest is the oscillatory behavior of the point agent compared to the follower during the formation rotation.

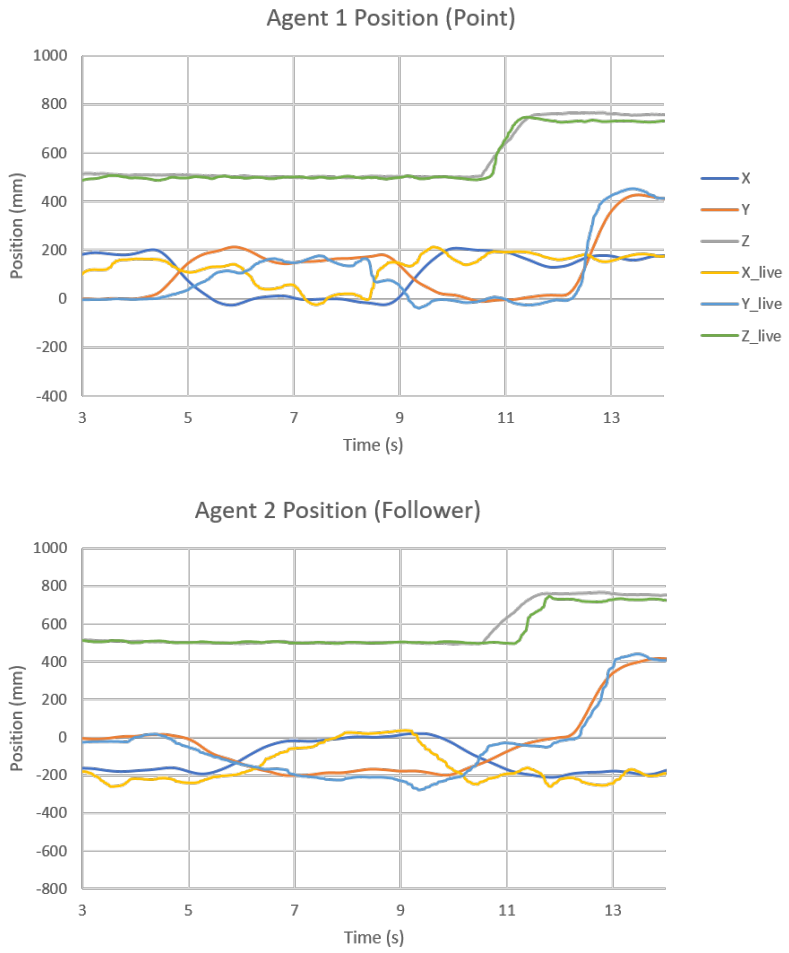

Figure 5.5: Superimposed position plots of two agents flying in a formation $r_{form} = 180$ mm and Agent 1 set as the formation point. The formation was commanded to rotate from $\psi_{form} = 0^o$ to $\psi_{form} = 90^o$ at 4.25 s, and back to $\psi_{form} = 0^o$ at 8.5 s. The formation was then instructed to translates from $[0, 0, 500]$ mm to $[0, 400, 500]$ mm at 12 s.

This oscillatory behavior is likely an a result of the current method used to enforce the formation protocol. Under the current protocol, when the formation is instructed to rotate from $\psi_{form} = 0^o$ to $\psi_{form} = 90^o$, the final destination of the point agent is the fixed, rotated point around the destination. The inter-agent cohesion control input is only introduced through the velocity set-point alone, meaning the position control will drive the system towards the destination without regard for the other agent, as it is already set by the global trajectory policy.

These conflicting control responses likely don't manifest in the control response of the simulated system in part due to the assumptions made in designing the simulation. In particular, the assumptions for instantaneous actuation and negligible drag force, which would otherwise produce a delay in the reaction of the system. Additionally there is an implicit assumption that each agent receives the relevant squad information with no data loss during operation. These inaccuracies are further compounded with each additional agent.

5.3 Point-mass Payload Transportation

The goal of this section is to compare a simulation of a formation of agents against the live system, under a known loading from a point-mass payload. In particular, we want to determine how well the simulation aligns in the absence of modeled inelastic collisions from agents regaining tension after a momentary loss. Since our current payload setup on the live system is not outfitted to track the payload position, the dynamics of the payload will need to be inferred from its influence on the agent pair.

5.3.1 Point-mass Payload 2 Agents

The two agent payload support scheme has an unsupported resonant mode, where the payload is free to swing [23]. As such, we want to ensure that these oscillations appear in our simulation, and that the response of the simulated agents represents that of the live system. Figure 5.6 shows the superimposed position states of both the simulated and

measured agents supporting a point-mass payload. Both the live and simulated systems show oscillations in their position dynamics of around 0.5 Hz. The period of a simple pendulum is known to be,

$$T_{pend} = 2\pi\sqrt{\frac{L}{g}}$$

For a formation radius of 200 mm, and a cable length of 609.6 mm, the expected resonant period be $T_{pend} = 1.52$ s, or 0.658 Hz, which is indeed close to the observed frequency. The reduction in the frequency of the oscillation likely comes from the ability of the quadrotors to translate during oscillations, and the down-draft of the motors producing some unmodeled damping of the payload motion. The underlying importance is that the simulation does indeed capture the unstable payload swing mode for two supporting agents.

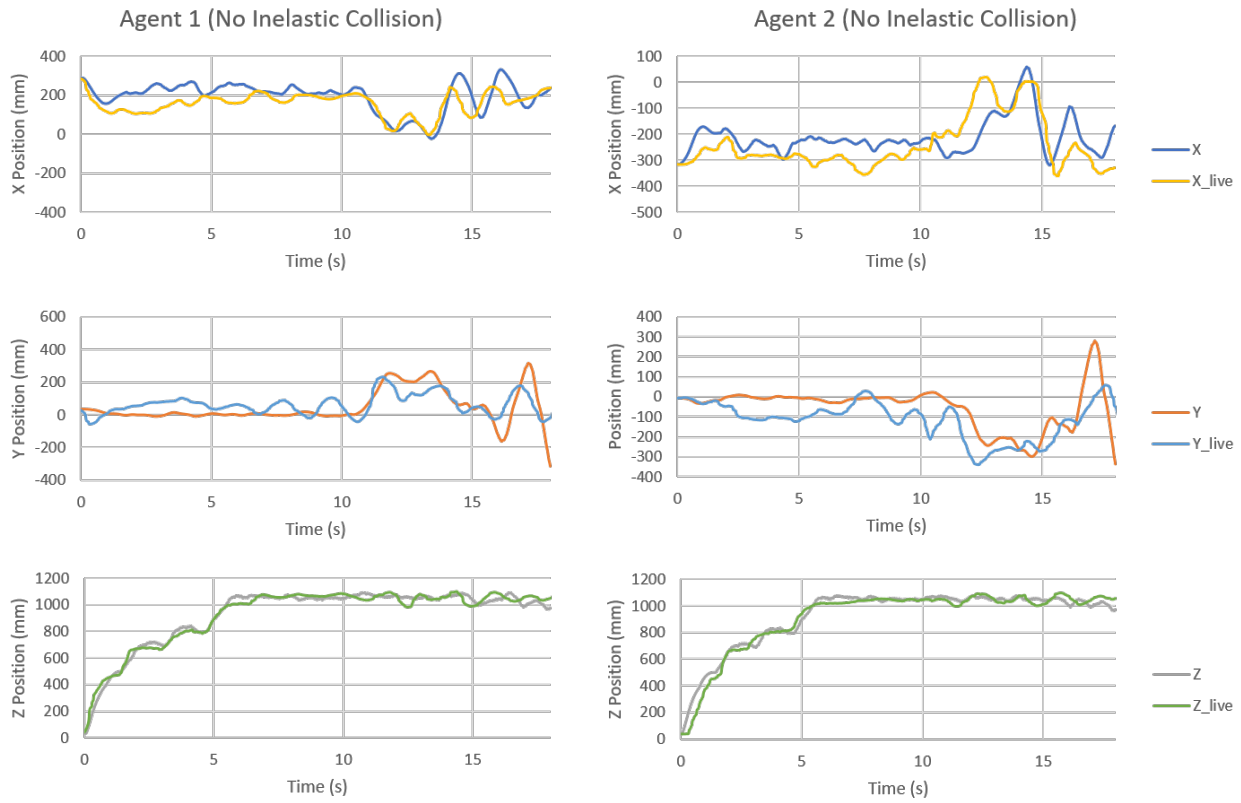

Figure 5.6: Superimposed simulated and measured position plots for two agents supporting a single $m_L = 0.7458$ kg payload using cables of $L_i = 609.6$ mm at a formation radius of $r_{form} = 200$ mm. The agents are incrementally instructed to a height of 1050 mm. At 10 s the formation angle is set to $\psi_{form} = 90^o$ and back to $\psi_{from} = 0^o$ at 13 s. (Left) Agent 1 position. (Right) Agent 2 position.

5.3.2 Point-mass Payload 3 Agents

The three-agent support scheme allows for the formation to control the unstable swing mode observed in two-agent system [23]. As such, we want to verify that the simulation and live systems indeed control the observed resonant mode and thereby can maintain control over the payload position. Figure 5.7 shows an image taken from the 3-agent payload transportation validation test. Figure 5.8 shows the superimposed plots of the simulated and measured positions for the 3 agent point-mass payload transportation

scheme, where $m_L = 0.07458$ kg and $L_i = 609.6$ mm. Indeed the oscillatory behavior observed in the position output of the two-agent scheme is no longer present, and the formation in both the simulated and live systems maintains control over the payload. In particular for agent 3, there is a large deviation from the expected behavior, which results from the agent falling out of formation placement around the unit circle, but still in a position to provide tensile support. Figure 5.9 shows the simulated position of the payload is under control during transit, which matches observations made during experimentation and the dynamics of the agents themselves.

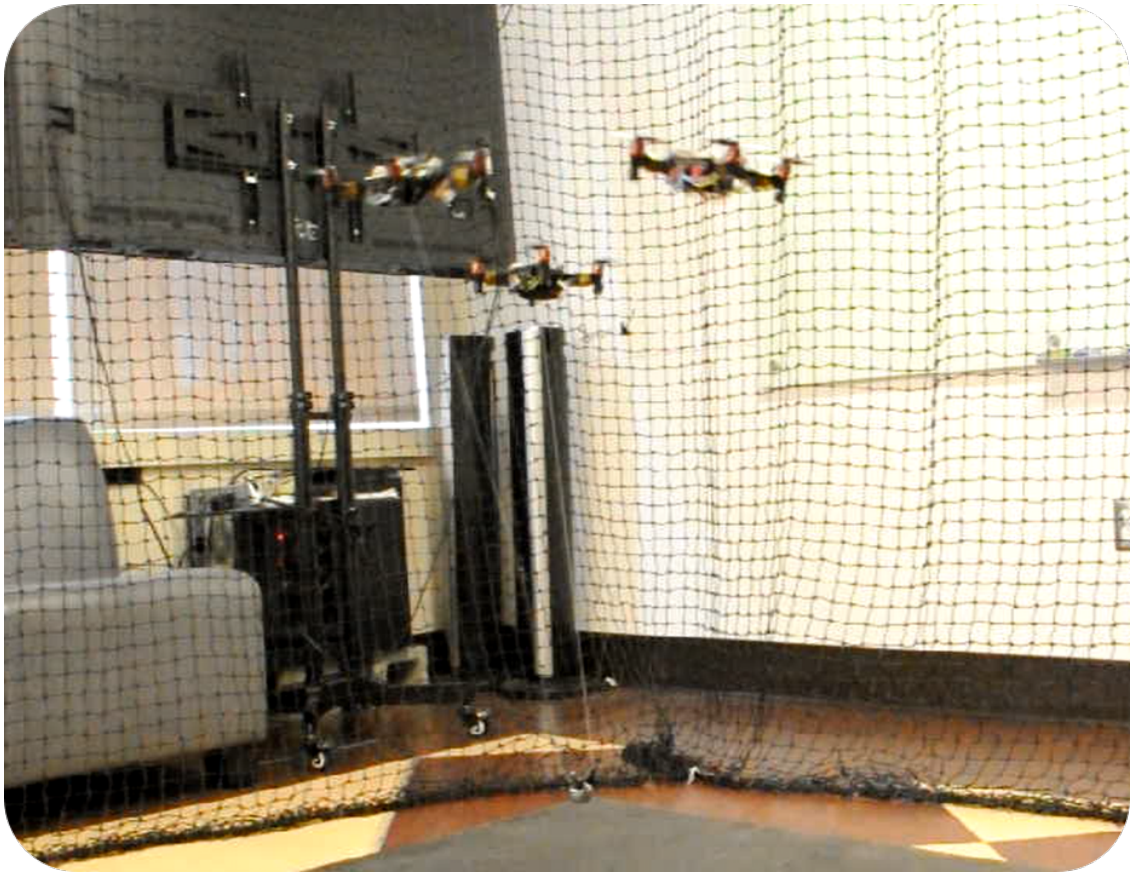

Figure 5.7: Screen-shot taken from the video of the 3 agent point-mass support scheme flight test.

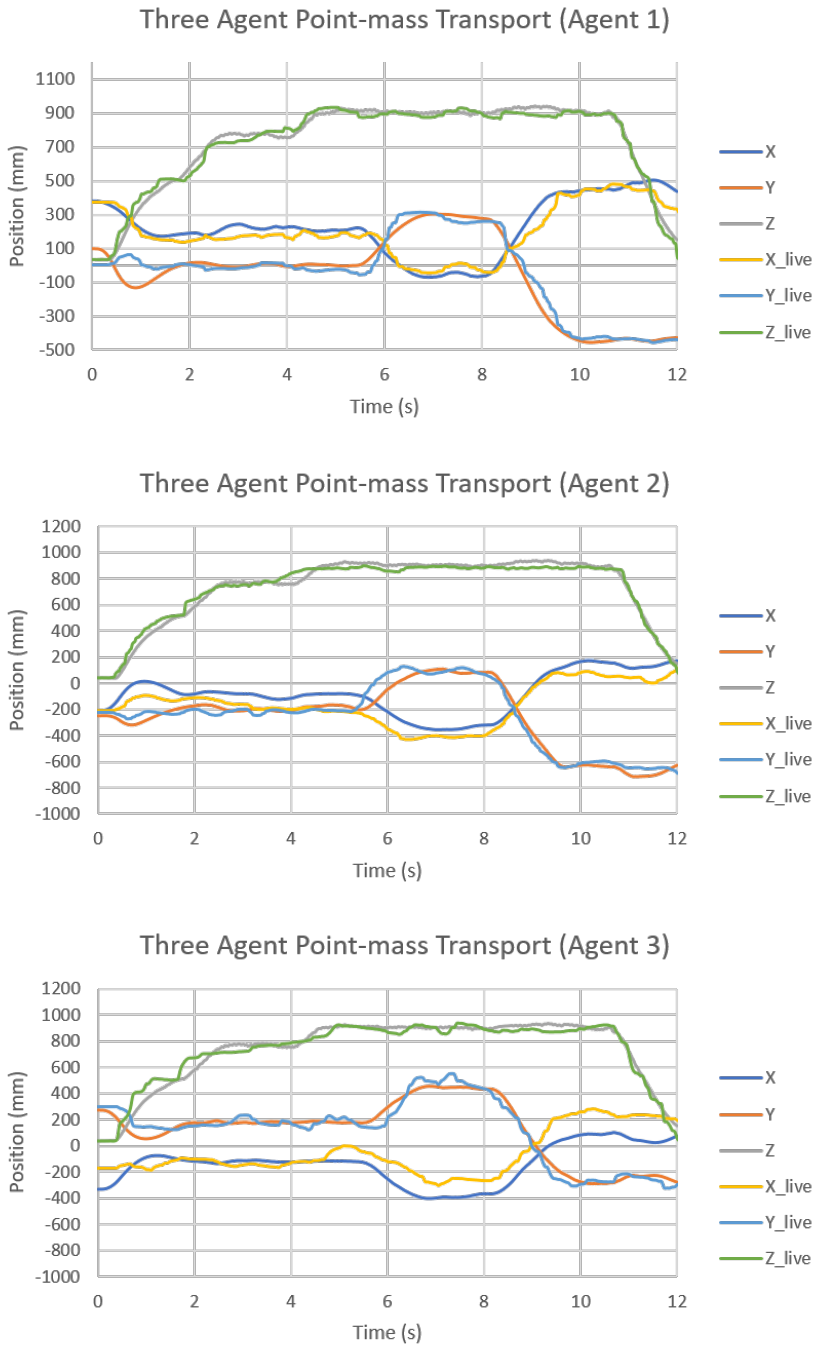

Figure 5.8: Superimposed simulated and measured position plots for three agents supporting a single $m_L = 0.7458$ kg payload using cables of $L_i = 609.6$ mm at a formation radius of $r_{form} = 200$ mm. (Top) Agent 1, serving as the formation point. (Middle) Agent 2. (Bottom) Agent 3.

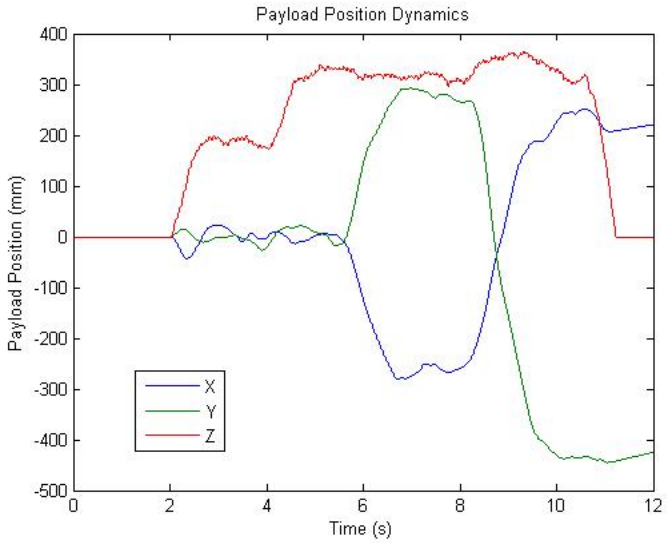

Figure 5.9: Figure of the simulated payload dynamics for the simulated three agent support scheme. Simulation uses a single $m_L = 0.7458$ kg payload using cables of $L_i = 609.6$ mm at a formation radius of $r_{form} = 200$ mm.

Chapter 6
DISCUSSION AND CONCLUSIONS

6.1 Current Achievements

Throughout this thesis, the fundamental goal has been building the underlying hardware and simulation infrastructure to facilitate future research within the RAIN lab in the area of distributed object manipulation. Beyond the base RAIN-DROP hardware, the underlying control and trajectory generation techniques were developed and successfully implemented on the live platform. The performance of the live system was assessed and validated against the developed simulation tools. Furthermore, the general steps taken to develop the necessary tools has been detailed in the preceding chapters to provide a guide to other research groups seeking to enter into this area of application.

As the RAIN lab is primarily a network controls lab, much of our research focus was viewed through the lens of formation control. The final 2D formation protocol developed in this thesis was successfully implemented on three platforms. The first component of the protocol uses a unit-vector consensus approach to place each agent around a common destination, at an agent specific radius. If one agent is designated as a formation point, then its placement around that destination is set based off a desired formation angle, allowing the other agents to distribute based of its location. The second component of the protocol was formation cohesion, which maintains the formation shape during flight.

To the knowledge of this author, the use of consensus approaches has not yet been employed as a means of defining an invisible support frame for payload transportation. However, there have been works that have drawn connections between consensus and mechanical systems, springs in particular [6, 17, 29]. Much like a spring, consensus approaches can be prone to oscillations in the presence of disturbances. As such, we

furthered our mechanical analogy and introduced a consensus damper, which works against the rate of change of the consensus value [6, 29]. Together, this formation protocol was used as the basis for our support frame to transport a point-mass payload.

6.2 Future Directions and Opportunities

As mentioned earlier, the primary achievement of this thesis was in development of the tools and infrastructure necessary to design and test object transportation protocols. Therefore, there are a great number of open challenges that remain unaddressed by this work.

2D to 3D Formation Control: Presently the formation controller designed within this thesis is restricted to the 2D x-y plane, which works well for systems that have eccentric anchor placements, or variable length cables. However, the current design of the RAIN-DROP hardware uses a fixed-length support cable. For a point-mass, this fixed cable length means that 2D changes in radial distance to the center of formation will result in the cable slacking or unintended force impulses applied to the payload. Introduction of a z-component for the formation control would help ensure that the norm distance between the agent and payload anchors matches the known length of the support cable. The current formation placement scheme can be modified to place the agents at a desired horizontal distance, and automatically adjust the z-axis placement based of the agents known cable length.

Payload State Estimation: In the case of the RAIN-DROP hardware, the system has no prior knowledge about the mass or shape of the payload it is supporting. In many real-world applications, it may not be practical to know the state of the payload, which presents the open challenge of inferring object state information from formation data. Such information could then be used to inform agent responses, and even be used to stabilize resonant modes, like that observed in the two agent support configuration.

Rigid-body Payload: While much of the rigid body payload transportation hardware and tools have been developed in this thesis, the focus was primarily centered on the point-mass configuration. This was in part done due to time-constraints, but also due to the use of the 2D formation protocol. If tension is lost and regained on a point-mass payload, it primarily affects the inertial dynamics. When tension is lost on a rigid-body payload, both the inertial and attitude dynamics are affected, resulting in unstable payload swing during transport. Introduction of a 3D consensus protocol will likely help address some of the cable tension loss encountered with the current configuration. Additionally, a payload estimation scheme will be equally vital to help ensure complete control over any potential payload swing.

RAIN-DROP Modifications: The primary challenge presented by the current RAIN-DROP hardware was that of flight time. Present battery technology limits the ability of small-scale batteries to deliver sufficient amperage to the motors to achieve maximum thrust. Such current over-draw conditions result in a large battery voltage drop during operation, and in particular during payload transportation. Future works should consider increasing the size of the battery to increase the operating current deliverable by the power supply, and following best practices to reduce excess weight on the agents.

BIBLIOGRAPHY

[1] Bitcraze. Crazyflie 2.1 product.

[2] Samir Bouabdallah and Roland Siegwart. Full control of a quadrotor. Number 1-4244-0912. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007. Autonomous Systems Lab.

[3] Joop Brokking. Project ymfc-al - the arduino auto-level quadcopter.

[4] Cazepony. Crazepony mini quadrotor.

[5] A. Chapman, E. Schoof, and M. Mesbahi. Advection on networks with an application to decentralized load balancing. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2680–2681, Oct 2012.

[6] Qifeng Chen, Sandor M. Veres, Yaonan Wang, and Yunhe Meng. Virtual spring-damper mesh-based formation control for spacecraft swarms in potential fields. 2015.

[7] Fishpepper.de. Pepperf15h micro-quadcoper.

[8] V.N.S. Huynh T.S. Le H. Q.T. Ngo, T.P. Nguyen and C.T. Nguyen. Experimental comparison of complementary filter and kalman filter design for low-cost sensor in quadcopter. 2017. International Conference on System Science and Engineering (ICSSE).

[9] R.C. Hibbler. *Engineering Mechanics: Dynamics*. Pearson Prentice Hall, 12th edition edition, 2010. ISBN-10: 0-13-607791-9.

[10] David Q. Mayne James B. Rawlings and Moritz M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2nd edition edition, October 2017. ISBN 9780975937730. Page 24.

[11] P. Kotaru, G. Wu, and K. Sreenath. Dynamics and control of a quadrotor with a payload suspended through an elastic cable. In *2017 American Control Conference (ACC)*, pages 3906–3913, May 2017.

[12] Unsik Lee and Mehran Mesbahi. Feedback control for spacecraft reorientation under attitude constraints via convex potentials. *IEEE Transactions on Aerospace and Electronic Systems*, 50(4):2578–2592, 2014.

[13] Giuseppe Loianno and Vijay Kumar. Cooperative transportation using small quadrotors using monocular vision and inertial sensing. In *IEEE ROBOTICS AND AUTOMATION LETTERS*, 2018.

[14] R. Lozano M. E. Guerrero, D. A. Mercado and C. D. Garcia. Passivity based control for a quadrotor uav transporting a cable-suspended payload with minimum swing. 2015. IEEE 54th Annual Conference on Decision and Control (CD).

[15] C. Masone, H. H. Bülthoff, and P. Stegagno. Cooperative transportation of a payload using quadrotors: A reconfigurable cable-driven parallel robot. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1623–1630, Oct 2016.

[16] Daniel Mellinger, Michael Shomin, Nathan Michael, and Vijay Kumar. Cooperative grasping and transport using multiple quadrotors. In *In Proceedings of the international*, 2010.

[17] Mehran Mesbahi and Magnus Egerstedt. *Graph Theoretic Methods in Multiagent Networks*. Princeton University Press, stu - student edition edition, 2010.

[18] Jonathan Fink Nathan Michael and Vijay Kumar. Cooperative manipulation and transportation with aerial robots. *Autonomous Robots*, 30(1):73–86, Sep 2010.

[19] Katsuhiko Ogata. *Modern Control Engineering*. Number 0-13-060907-2. Tom Robbins, 1970,1990,1997,2002.

[20] Pedro Garcia Gil Pedro Castillo Garcia, Laura Elana Munoz Harnadez. *Indoor Navigation Strategies for Aerial Autonomous Systems*. Joe Hayton, 2017.

[21] Hossein Rastgoftar and Ella Atkins. Continuum deformation of a multiple quadcopter payload delivery team without inter-agent communication. pages 539–548, 06 2018.

[22] E. Schoof, A. Chapman, and M. Mesbahi. Efficient leader selection for translation and scale of a bearing-compass formation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1816–1821, May 2015.

[23] Koushil Sreenath and R. Vijay Kumar. Dynamics, control and planning for cooperative manipulation of payloads suspended by cables from multiple quadrotor robots. In *Robotics: Science and Systems*, 2013.

[24] M. Szmuk, C. A. Pascucci, and B. AÇikmeşe. Real-time quad-rotor path planning for mobile obstacle avoidance using convex optimization. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, Oct 2018.

[25] M. Szmuk, C. A. Pascucci, D. Dueri, and B. Açikmeşe. Convexification and real-time on-board optimization for agile quad-rotor maneuvering and obstacle avoidance. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4862–4868, Sep. 2017.

[26] Ascending Technologies. Asctec hummingbird product.

[27] Kamran Turkoglu and Ankyda Ji. *Development of a Low-Cost Experimental Quadcopter Testbed using an Arduino controller for Video Surveillance.*

[28] Greg Welch and Gary Bishop. *An Introduction to the Kalman Filter.* University of North Carolina at Chapel HillDepartment of Computer ScienceChapel Hill, NC 27599-3175, SIGGRAPH, 2001. Copywright by ACM, Inc.

[29] Behçet Açikmeşe Yue Yu and Mehran Mesbahi. Mass-spring-damper networks for distributed optimization in non-euclidean spaces. Department of Aeronautics and Astronautics, University ofWashington, Seattle, WA, March 2019.

[30] Marco Pavone Zijian Wang, Sumeet Singh and Mac Schwager. Cooperative object transport in 3d with multiple quadrotors using no peer communication. In *International Conference on Robotics and Automation (ICRA)*, 2018.