8-21-2023

A Multi-Aerial Vehicle Platform for Testing Collaborative Mobility Formation Controllers

Matthew Snellgrove

A MULTI-AERIAL VEHICLE PLATFORM FOR TESTING COLLABORATIVE MOBILITY FORMATION CONTROLLERS

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science

in

The Department of Mechanical and Industrial Engineering

by
Matthew Thomas Snellgrove
B.S., Louisiana State University, 2022
December 2023

Acknowledgments

I would like to thank LaSPACE and their Graduate Student Research Assistance Program (GSRA) whose funding has been extremely valuable to myself and this project. I would like to thank Dr. Marcio de Queiroz whose guidance helped push this project forward and helped in moving past constant hurdles. Lastly, I would like to thank my peers Joshua Nguyen, Parsa Molaei, and Farid Sahebsara for their constant help and advice.

Table of Contents

List of Figures

Abstract

Multi-agent systems have potential applications in a multitude of fields due to their ability to perform complex tasks with a high degree of robustness. Decentralized systems are multi-agent systems where sensing, control, and communication responsibilities are distributed amongst the agents giving this scheme a particularly high potential for robustness, adaptability, and complexity; however, this comes with its own list of challenges. Formation control is a topic in multi-agent systems that aims to have the agents moving as a virtual rigid body through space. In the literature, there have been successful implementation of formation control algorithms on robot agents in both the 2D and 3D case; however, these have all been done using a centralized setup.

The aim of this project was to create a 3D decentralized platform for testing formation controllers on mini quadcopters. This was done using three Parrot Mambo quadcopters and Simulink/MATLAB for the control and communication programming as well as simulation. The distance-based formation control scheme, which is inherently decentralized, was used to test the platform. Implementation of formation acquisition was shown in simulation and real-life testing with perfect acquisition shown in simulation and up to 10% steady state error on the hardware. This error in the real-life test is likely attributable to noisy and inaccurate sensors, communications delays, and unmodeled dynamics. The platform was successfully created and demonstrated and could serve as a testbed for further research. Overall, the hardware results demonstrate working but potentially improvable results.

Chapter 1. Introduction

1.1. Background

As the initial hurdles of unmanned robotic system development have been passed, focus is now being placed on advancing the use of these systems for performing coordinated operations in groups, autonomously or with limited human supervision. Research has progressed from fundamental studies of biological swarms in nature to the development and application of systems theoretical tools for modeling such behaviors to, more recently, the synthesis and experimental validation of engineered multi-agent systems.

Over the past several years, a considerable amount of work has been conducted in this area under various names: collaborative mobility, multi-agent systems, networked systems, cooperative control, and swarming. The term *collaborative mobility* refers to a network of autonomous mobile agents that cooperate in performing complex tasks beyond their individual capabilities [1]. Such systems have the potential to impact a variety of defense, civilian, scientific, and commercial applications that involve some form of situational awareness. Examples include patrolling, surveying, and scouting over large geographical areas with unmanned robotic vehicles or mobile sensor networks. More specifically, a group of autonomous (ground, underwater, water surface, or air) vehicles could be deployed in disaster areas without putting first-responders in harm's way. Another potential application is a team of vehicles collaboratively transporting an object too large and/or heavy for any single one to transport. For space missions, a multi-agent system could be deployed as explorers that precede human missions, as crew helpers, or as custodians of assets left behind. *Formation control* is an important problem in collaborative mobility where the objective is for agents to form and maintain a prescribed geometric shape in 3D space while maneuvering as a virtual rigid body [1].

This requirement is intrinsic to many of the applications mentioned above.

Engineered multi-agent systems provide several advantages to system operation: more efficient and complex task execution, robustness when one or more agents fail, scalability, versatility, and adaptability. On the other hand, multi-agent systems introduce a host of unique challenges, including coordination and cooperation schemes, distribution of information and subtasks, negotiation between team and individual goals, communication protocols, sensing, and collision avoidance. These challenges are exacerbated by the fact that often the task is to be completed with limited computational, communication, and sensing resources.

An important concept in multi-agent systems is the idea of centralization and decentralization. In general, centralized systems have one central information acquiring entity, which itself may be an agent in the system, which shares this information to the agents in the system. This information may be about the state of the agents themselves, the environment they are operating in, or high-level instructions. In this system, the central agent serves as a single point of failure. This sort of system also does not scale well due to the high communication overhead required by the central agent. In a decentralized system, information is acquired locally by the agents themselves, usually via onboard hardware and software. This information can then be passed on to other agents on a need-to-know basis depending on the functions of the system. It is apparent why this is sometimes called a *distributed* scheme since information and high-level instructions are acquired and distributed throughout the system. It is also easy to see the potential for this scheme to be robust to failure and to be highly scalable.

1.2. Literature Review

In recent years, several research platforms for multi-agent systems have been developed for a variety of research purposes. Reference [2] gives a survey of such platforms. Included is a

testbed developed at the University of Utah which uses Acroname Garcia robots (2D ground vehicles) to research integration of robot agents with Wireless Sensor Networks (WSNs) and a testbed developed at the University of Seville which allows users to perform remote experiments using several ground vehicles, a WSN, and a camera network.

Robot agents that can operate in 3D space, such as unmanned aerial vehicles (UAVs), are far more interesting and challenging for formation control since they can form 3D structures in space. Among the multi-UAV research platforms described in the literature, virtually all are rotorcraft due to their vertical takeoff and landing capability which demands less physical space and allows for indoor operation. Quadrotor arrangements are the most common because it is the smallest number of rotors that allows adequate control of its motion along the 6 degrees of freedom. Custom quadrotors were used for outdoor operation with GPS localization in [3] and for indoor operation with VICON localization in [4]. The indoor experiments in [5] were based on commercial Blade Nano QX quadrotors along with the VICON camera system and a central control station. DIY quadrotors from 3D Robotics equipped with GPS were employed in [6] to evaluate a cycle-free formation control law. In [7], the RadioLink F450 quadrotor was used for indoor experimentation of cooperative robots navigating in densely cluttered environments, also using the VICON system. One common quality these research projects all share is they're implemented in a centralized fashion, either through a camera system or GPS which captures some portion of the state of the agents, i.e., not all information is acquired locally by the agent's own capabilities.

1.3. Research Goal

The goal of this project was to create a platform for testing formation controllers on UAVs in a decentralized manner and to demonstrate its working results. According to the

existing literature and the best knowledge of those working with the project, this sort of platform has yet to be demonstrated on real hardware. This comes with challenges as well as project specific constraints. The UAV control must be modifiable, safely operable indoors, have a cost of less than $350, and the development time of the project must be one year or less. Additionally, the UAVs must have the ability to acquire all relevant information onboard, which is the main divergence this work takes on from existing work, and the platform must allow for the distribution of information among the agents in the system. It is also important that the developed platform be user-friendly, allowing subsequent researchers to test other capabilities such as more agents, other formation controllers, or high-level tasks such as collision avoidance.

These goals were accomplished, and the means are described in this thesis. Chapter 2 presents the dynamic model of the system and then follows into the control methods used to perform the formation control. The high-level control is governed by the so-called *distance-based formation control* approach for the double-integrator model which is defined in Section 2.3 as well as the supporting graph theory. Section 2.4 discusses the low-level control that was developed which includes PID control as well as a sliding mode controller for pitch and roll regulation. In Chapter 3, the multi-UAV platform and real-life control implementation are discussed. We will look at the UAV selected, the Parrot Mambo, as well as the MathWorks developed package, the Simulink Support Package for Parrot Mambo, that was used to upload Simulink code to the drones. We will also discuss the communication scheme developed for inter-agent communication via Simulink and the use of the ROS Toolbox in MATLAB. Finally, simulation and real-world results are presented and discussed with concluding remarks to wrap up in Chapter 4.

Chapter 2. System Modeling and Control

2.1. Dynamic Model of System

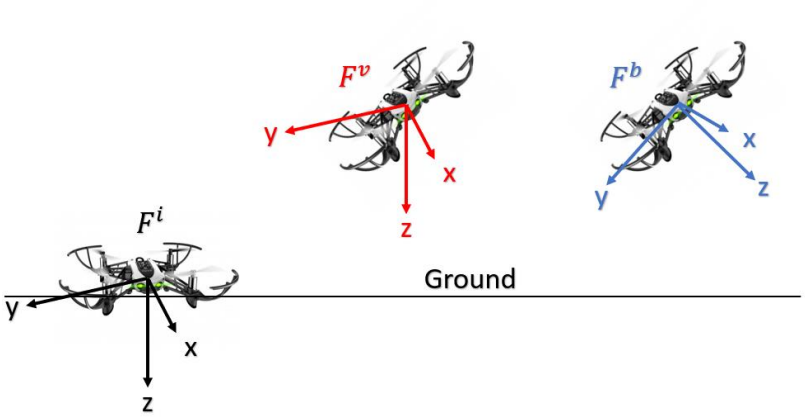

Figure 2.1. Relevant reference frames

The dynamic model description in this section follows the presentation in [8]. Before deriving the system model, we need to define three reference frames (see Figure 2.1). The Inertial Frame, F^i, has the following axis orientation: x-axis sticking out the front of the quadcopter, the z-axis pointing to the ground, and the y-axis completing the right-hand coordinate system. It is rigidly attached in space to the initial position of the UAV's center of mass and does not move. The Vehicle Frame, F^v, is attached to the center of mass and translates with the UAV, but its orientation is always aligned with that of the inertial frame. The Body Frame, F^b, also moves with the center of mass, with its initial orientation being that of the other two frames; however, it is rigidly attached to the vehicle and rotates with it.

Euler angles are used in this case to handle the 3D rotation of the Body Frame relative to the Vehicle Frame (see [13] for a detailed presentation). This is done with three subsequent rotations. First, a "Yaw" rotation about the z-axis of angle ψ followed by a "Pitch" rotation about the current y-axis of angle θ followed by a "Roll" rotation about the current x-axis of angle ϕ. The rotation matrix from the Vehicle Frame to the Body Frame using the discussed Euler convention is given by

$$R_v^b = \begin{bmatrix} c\theta c\psi & c\theta s\psi & -s\theta \\ s\phi s\theta c\psi - c\phi s\psi & s\phi s\theta s\psi + c\phi c\psi & s\phi c\theta \\ c\phi s\theta c\psi + s\phi s\psi & c\phi s\theta s\psi - s\phi c\psi & c\phi c\theta \end{bmatrix}. \quad (1)$$

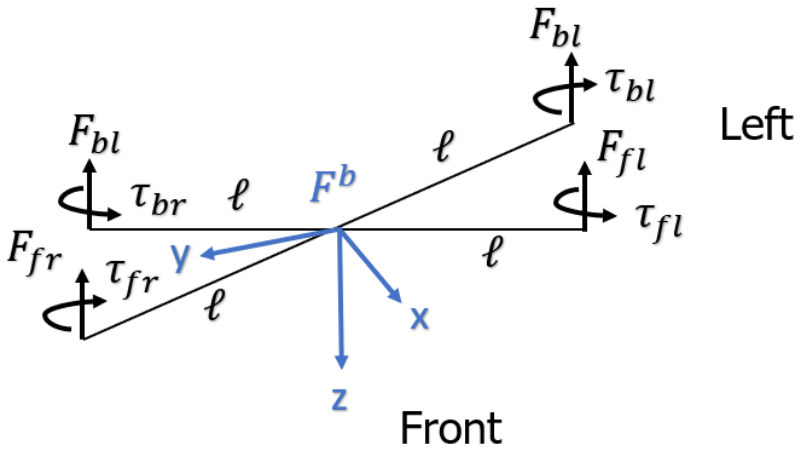

Figure 2.2. Propeller forces and reaction torques

Each rotating propeller imparts a thrust force and reaction torque onto the quadcopter, defined in the Body Frame, as seen in Figure 2.2. Here, "fl" stands for “front left” and "br" stands for “back right”, which help define the four propellors. It should be noted that the front left and back right propellor rotate in the clockwise direction and the other pair rotates in the counterclockwise direction. The total thrust generated by the propellors is given by

$$F = F_{fl} + F_{fr} + F_{bl} + F_{br} \quad (2)$$

and the toques along the Body Frame axes are

$$\tau_x = F_{fl}\ell + F_{bl}\ell - F_{fr}\ell - F_{bl}\ell \quad (3)$$

$$\tau_y = F_{fl}\ell + F_{fr}\ell - F_{br}\ell - F_{bl}\ell \quad (4)$$

$$\tau_z = \tau_{bl} + \tau_{fr} - \tau_{fl} - \tau_{br}. \quad (5)$$

Assuming the motor thrusts and reaction torques are proportional to pulse width modulation inputs, they can be written as

$$F_\star = k_1\delta_\star \quad (6)$$

$$\tau_\star = k_2\delta_\star \quad (7)$$

where $\star$ represents fl, bl, fr, and br and δ the motor pulse width modulation input. Then, the motor inputs can be solved for by

$$\begin{bmatrix}\delta_{fl}\\ \delta_{bl}\\ \delta_{fr}\\ \delta_{br}\end{bmatrix} = \begin{bmatrix}k_1 & k_1 & k_1 & k_1\\ \ell k_1 & \ell k_1 & -\ell k_1 & -\ell k_1\\ \ell k_1 & -\ell k_1 & \ell k_1 & -\ell k_1\\ -k_2 & k_2 & k_2 & -k_2\end{bmatrix}^{-1}\begin{bmatrix}F\\ \tau_x\\ \tau_y\\ \tau_z\end{bmatrix}. \tag{8}$$

From Newton's Second Law, the dynamic equations of motion for the UAV in the body frame can be derived as

$$\begin{bmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{bmatrix} = \begin{bmatrix}rv - qw\\ pw - ru\\ qu - pv\end{bmatrix} + \begin{bmatrix}-gsin(\theta)\\ gcos(\theta)\sin(\phi)\\ gcos(\theta)\cos(\phi)\end{bmatrix} + \frac{1}{m}\begin{bmatrix}0\\ 0\\ -F\end{bmatrix} \tag{9}$$

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix}\frac{J_y - J_z}{J_x}qr\\ \frac{J_z - J_x}{J_y}pr\\ \frac{J_x - J_y}{J_z}pq\end{bmatrix} + \begin{bmatrix}\frac{1}{J_x}\tau_x\\ \frac{1}{J_y}\tau_y\\ \frac{1}{J_z}\tau_z\end{bmatrix} \tag{10}$$

where (9) is for linear motion and (10) is for rotational motion. The variables in (9) and (10) are defined as:

- u, v, and w are the absolute linear velocities given in Body Frame coordinates along the x, y, and z axes, respectively
- p, q, and r are the angular velocities of the Body Frame relative to the Vehicle Frame given in Body Frame coordinates along the x, y, and z axes, respectively
- m represents the UAV's total mass and J_x, J_y, and J_z are the principal moments of inertia along the axes of the Body Frame
- g is the acceleration of due to gravity (9.81 $\frac{m}{s^2}$)

The following equations give the kinematic relationships between the Body Frame and Inertial Frame velocities

$$\begin{bmatrix}\dot{X}\\ \dot{Y}\\ \dot{Z}\end{bmatrix} = R_v^{b\,T}\begin{bmatrix}u\\ v\\ w\end{bmatrix} \tag{11}$$

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta)\\ 0 & \cos(\phi) & -\sin(\phi)\\ 0 & \dfrac{\sin(\phi)}{\cos(\theta)} & \dfrac{\cos(\phi)}{\cos(\theta)}\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix} \tag{12}$$

where X, Y, and Z are the position coordinates of the UAV center of mass in the Inertial Frame. Equations (9)-(12) completely define the nonlinear dynamics and kinematics for the quadcopter and can be used to design flight controllers as well as in simulation.

The following simplifications are made to the system model. Using small angle approximation for $\psi, \theta, \text{and } \phi$, (12) becomes

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix} \tag{13}$$

and therefore

$$\begin{bmatrix}\ddot{\phi}\\ \ddot{\theta}\\ \ddot{\psi}\end{bmatrix} = \begin{bmatrix}\dfrac{J_y - J_z}{J_x}qr\\ \dfrac{J_z - J_x}{J_y}pr\\ \dfrac{J_x - J_y}{J_z}pq\end{bmatrix} + \begin{bmatrix}\dfrac{1}{J_x}\tau_x\\ \dfrac{1}{J_y}\tau_y\\ \dfrac{1}{J_z}\tau_z\end{bmatrix}. \tag{14}$$

Furthermore, differentiating (11) and neglecting $\dot{R}_v^{b\,T}$ (assuming small angular velocities) gives

$$\begin{bmatrix}\ddot{X}\\ \ddot{Y}\\ \ddot{Z}\end{bmatrix} = R_v^{b\,T}\begin{bmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{bmatrix}. \tag{15}$$

Now, substituting (9) into (15), neglecting Coriolis terms (again assuming small angular velocities), and assuming Yaw angle $\psi = 0$ gives

$$\begin{bmatrix} \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \end{bmatrix} = \begin{bmatrix} -\frac{F\cos(\phi)\sin(\theta)}{m} \\ \frac{\mathrm{Fsin}(\phi)}{m} \\ g - \frac{Fcos(\phi)\cos(\theta)}{m} \end{bmatrix} \quad (16)$$

which will be useful later.

2.2. Overall Control Architecture

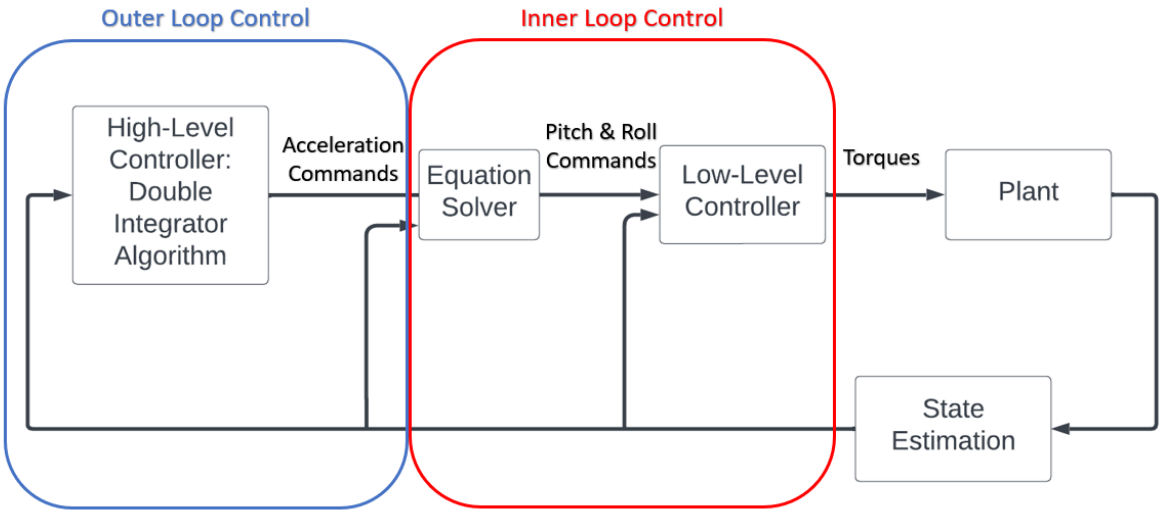

Figure 2.3. Control architecture

This section serves to present the control archeticture which consists of an outer and inner loop scheme as seen in Figure 2.3. The outer loop is governed by the double-integrator, distance based formation control algorithm which is discussed in the following section. The output of the outer loop is an acceleration command. The inner loop plays the role of actuating the quadcopter to achieve the desired acceleration, as to be discussed in Section 2.4.

2.3. Formation Control for Double-Integrator Model

2.3.1. Graph Theory

Discussion on the following topics in this section and the next follows the presentation in [1]. Graph theory is a useful tool for studying formation control. In general, graph theory studies

vertices that are connected by edges which we can imagine as robot agents and sensing, control, and/or communication links between agents, respectively. A graph G is a pair (V, E) where $V = \{1, 2, \dots, N\}$ is a set of vertices and $E = \{(i, j) | \, i, j \in V, i \neq j\}$ is a set of edges connecting the vertices. The set of neighboring vertices for a vertex i is defined as $\mathcal{N}_i(E) = \{(j \in V | (i, j) \in E)\}$. If $p_i \in \mathbb{R}^3$ gives the coordinates of vertex *i* of a 3D graph, then a framework *F* is the pair (*G*, $\boldsymbol{p}$) where $\boldsymbol{p} = [p_1, \dots, p_N] \in \mathbb{R}^{3N}$.

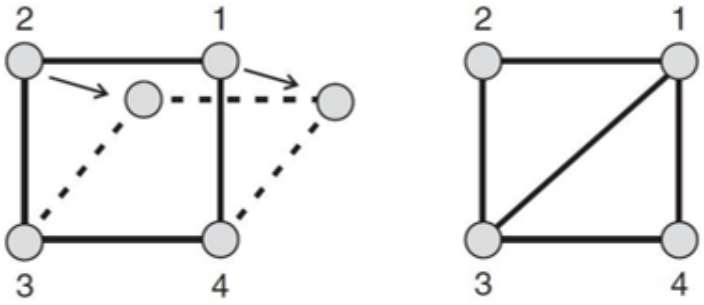

Figure 2.4. Example of simple graphs. Image taken from [1]

Rigidity is a property of frameworks which is highly relevant for formation control. We image a ball and joint framework which is a mechanical idea of rigid rods connected by ball joints. This framework is idealized in the fact that a ball joint can connect any number of rods and the rods are able to pass through one another freely. Using this idea, a framework is said to be rigid if all its continuous motions are rigid body motions. For a simple example, see Figure 2.4 where it is clear some vertices of the left graph can move and deform the structure, while on the right there are no possible motion of the vertices other than moving the whole structure as a rigid body. For simple graphs, human intuition may be enough to determine rigidity, however the related notion of *infinitesimal rigidity* is useful since infinitesimal rigidity implies rigidity and it is easy to check for via a matrix rank. First, an edge function $\gamma(\boldsymbol{p}) = \left[\dots, \|p_i - p_j\|^2, \dots\right]$, $(i, j) \in E$ can be defined which maps the relative position vectors of the vertices in E

to a vector of the squared edge lengths. To preserve the edge length between any vertices i and j, then the following must hold

$$\frac{d}{dt}\|p_i - p_j\|^2 = (p_i - p_j)^T(v_i - v_j) = 0 \tag{17}$$

where $v_i = \dot{p}_i$. By defining a *rigidity matrix* $\mathcal{R}(\boldsymbol{p}) = \frac{1}{2}\frac{\partial\gamma(\boldsymbol{p})}{\partial \boldsymbol{p}}$, (17) can rewritten to show the preservation of distances between all $(i,j) \in E$ as

$$\mathcal{R}(\boldsymbol{p})\boldsymbol{v} = \frac{1}{2}\frac{\partial\gamma(\boldsymbol{p})}{\partial \boldsymbol{p}}\boldsymbol{v} = \mathbf{0} \tag{18}$$

where $\boldsymbol{v} = [v_1, \dots, v_N] \in \mathbb{R}^{3N}$. Then, it can be shown that a framework in $\mathbb{R}^3$ is infinitesimally rigid if and only if $rank\big(\mathcal{R}(\boldsymbol{p})\big) = 3N - 6$. Also, a graph is *minimally rigid* if it is rigid, and the removal of a single edge causes it to lose its rigidity. A graph is minimally rigid in $\mathbb{R}^3$ if and only if $l = 3N - 6$ where l is the number of edge lengths. A graph is both minimally and infinitesimally rigid if its rigidity matrix has full row rank and if $\mathcal{R}(\boldsymbol{p})\mathcal{R}(\boldsymbol{p})^T$ is positive definite. For the case of three vertices, it is easy to use the above tools to show that a graph with three edges is minimally rigid.

It is now important to discuss the connection between formation control and graph theory. Through graph theory, we can create rigid structures based on the assumption that those structures are made of rigid bars connected by ball joints. In formation control, we want robotic agents to move through space as virtual rigid bodies. Therefore, if we have some sort of control over the agents to be able to acquire and maintain some defined inter-agent distances (the rigid bars), and we use graph theory to define the correct number of controlled inter-agent distances (from the ideas presented on rigidity), we can control the robot agents to move together as a rigid body (rigid graph).

2.3.2. High Level Control Law

The job of the high-level control is to regulate the defined inter-agent distances to their desired values. In this project, the so-called *double-integrator model* was used which seeks to control the accelerations of the agents. Let $\boldsymbol{u}_i$ be the control input to agent i and $\boldsymbol{q}_i = \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix}$ as defined in section 2.1, then the equations of motion for the double-integrator model are

$$\begin{aligned} \dot{\boldsymbol{q}}_i &= \boldsymbol{v}_i \\ \boldsymbol{v}_i &= \boldsymbol{u}_i = \begin{bmatrix} \ddot{X}_i \\ \ddot{Y}_i \\ \ddot{Z}_i \end{bmatrix}. \end{aligned} \tag{19}$$

Let $\tilde{\boldsymbol{q}}_{ij} = \boldsymbol{q}_i - \boldsymbol{q}_j$ be the position vector from agent j to agent i and d_{ij} be the desired inter-agent distance between i and j. Then, we can define the inter-agent distance error as

$$e_{ij} = \left\| \tilde{\boldsymbol{q}}_{ij} \right\| - d_{ij}. \tag{20}$$

It also useful to define

$$z_{ij} = \left\| \tilde{\boldsymbol{q}}_{ij} \right\|^{\mathbf{2}} - d_{ij}^2. \tag{21}$$

Then the so-called distance-based formation control law for each agent is given by

$$\boldsymbol{u}_i = -k_a \boldsymbol{v_i} - \sum_{j \in \mathcal{N}_i(E)} \left[(k_a k_v + 1) \tilde{\boldsymbol{q}}_{ij} \boldsymbol{z}_{ij} + k_v \left(\boldsymbol{z}_{ij} I_{\mathbf{3}} + 2 \tilde{\boldsymbol{q}}_{ij} \tilde{\boldsymbol{q}}_{ij}{}^T \right) \tilde{\boldsymbol{v}}_{ij} \right] \tag{22}$$

where $k_a, k_v > 0$ are user defined gains and I_3 is the 3x3 identity matrix. It has been shown in [1] that the control low in (22) renders $\boldsymbol{e} = \left[\dots, e_{ij}, \dots \right] = \mathbf{0}, (i,j) \in E$ to be locally exponentially stable.

2.4. Low Level Control

The low-level controller receives inputs of $\boldsymbol{u}_i$ from the double integrator model and tries

to track these desired accelerations. Note that (16) can be manipulated such that

$$\phi^d = \tan^{-1}\left(\frac{\ddot{Y}\cos(\theta)}{g - \ddot{X}}\right) \tag{23}$$

$$\theta^d = \tan^{-1}\left(\frac{\ddot{X}}{\ddot{Z} - g}\right) \tag{24}$$

$$F = m\frac{g - u_z}{\cos(\phi)\cos(\theta)} \tag{25}$$

where ϕ^d and θ^d are the desired pitch and rolls angles, respectively. Using Equations (22)-(24) one can solve for the correct pitch and roll angles and thrust to actuate the UAV to the desired acceleration from the double integrator model.

The next challenge is controlling the UAV to the desired and pitch and roll angles. This was done by using a sliding mode controller adapted from [9]. Let $e_\phi = \phi^d - \phi$ be the pitch tracking error and $s_\phi = \dot{e_\phi} + \omega_\phi e_\phi$, where ω_ϕ is a user-defined positive constant, define the error surface in the error state space. If s_ϕ can be driven to zero, then the error dynamics are defined by $\dot{e_\phi} = -\omega_\phi e_\phi$ which clearly implies e_ϕ is exponentially driven to zero given that $\omega_\phi > 0$. Differentiating the error surface gives $\dot{s_\phi} = \ddot{e_\phi} + \omega_\phi \dot{e_\phi}$. Using (14), this becomes

$$\dot{s_\phi} = \ddot{\phi}^d - \frac{J_y - J_z}{J_x} qr - \frac{1}{J_x}\tau_\phi + \omega_\phi \dot{e}_\phi \tag{26}.$$

The torque-level control law is given as

$$\tau_\phi = J_x\left(\ddot{\phi}^d + \omega_\phi \dot{e}_\phi - \frac{J_y - J_z}{J_x} qr + \varepsilon_\phi tanh(s_\phi) + k_\phi \int s\, dt\right), \tag{27}$$

where $\varepsilon_\phi, k_\phi > 0$ are user-defined gains, which transforms (26) to the form of

$$\dot{s}_\phi = -\varepsilon_\phi tanh(s_\phi) + V \tag{28}$$

$$\dot{V} = -k_\phi s. \tag{29}$$

The stability can be checked via the following Lyapunov function candidate

$$L(s_\phi, V) = \frac{1}{2}k_\phi s_\phi^2 + \frac{1}{2}V^2. \tag{30}$$

Taking the derivative of (30) and substituting in (28) and (29) gives

$$\dot{L} = -k_\phi \varepsilon_\phi s_\phi \tanh(s_\phi) \tag{31}$$

which is negative semidefinite and therefore $s_\phi = 0$ is rendered asymptotically stable [10].

Similarly, the control law for the roll angle is

$$\tau_\theta = J_y\left(\ddot{\theta}^d + \omega_\theta \dot{e}_\theta - \frac{J_z - J_x}{J_y} pr + \varepsilon_\theta tanh(s_\theta) + k_\theta \int s_\theta \, dt\right) \tag{32}.$$

PID control laws were used for the yaw and altitude variables. The yaw controller, which is trying to maintain a yaw angle of 0°, is given by

$$\tau_\psi = k_{\psi P} e_\psi + k_{\psi I} \int e_\psi \, dt + k_{\psi D} \dot{e}_\phi \tag{33}$$

where $e_\psi = \psi^d - \psi$ while the altitude controller is

$$F = k_{zP} e_z + k_{zI} \int e_z \, dt + k_{zD} \dot{e}_z \tag{34}$$

where $e_z = Z^d - Z$ and $k_{\psi P,}\, k_{\psi I}, k_{\psi D}, k_{zP}, k_{zI}$, and $k_{zD} > 0$ are used defined gains. The thrust control law of (34) was opted for as opposed to (25). This affects how well the UAVs track the acceleration commands but gives the ability to regulate their altitudes. This was done due to imperfect measurements of the UAV's mass.

Chapter 3. Experimental Platform and Results

3.1. Hardware

Figure 3.1. Parrot Mambo Fly

To reduce the development time of the project, it was decided to select a commercially available UAV as opposed to developing a DIY one. The quadcopter selected was the *Parrot Mambo Fly*, shown in Figure 3.1. This UAV is small (18 × 18 × 4 cm) and lightweight (~68 grams), making it perfectly suitable for indoor operation, and is relatively inexpensive (~$220). The Parrot Mambo Fly uses an "X" rotor configuration (see Figure 2.2) where the x-axis of the Body Frame is aligned between the two front motors as opposed to a "+" rotor configuration where the x-axis points directly to the front motor. Each rotor is driven by a brushless DC motor. The quadcopter is powered by a 660 mAh lithium polymer battery and has a flight time of 8-10 minutes. The battery can be recharged within 30 minutes through a micro-USB cable connected to a computer or USB power adaptor. The internal memory of the UAV can store up to 1GB of data. The internal processor has a processing speed of 200 Hz. The Parrot Mambo Fly is equipped with the following sensors for feedback control.

- Camera: The 0.3 megapixel camera is located under the UAV facing downwards and has a frame rate of 60 fps. It is used to measure the UAV's horizontal motion using optical flow, which estimates velocity based on relative motion between patterns in consecutive frames.

- Ultrasound: The ultrasonic sensor, which is also located under the UAV, measures its vertical position relative to the ground (altitude). The sensor range is 4 meters.
- Inertial measurement unit: This consists of a 3-axis accelerometer and a 3-axis gyroscope and is used to measure the UAV's translational acceleration and angular velocities.
- Air pressure sensor: This is used to measure the UAV's vertical position beyond the 4-meter range ("high" altitudes) using the fact that the air pressure decreases with increasing altitude.

3.2. Software

MathWorks has developed a MATLAB add-on called Simulink Support Package for Parrot Mambo (SSPPM), which allows a user to build and deploy flight control algorithms on the UAVs over Bluetooth from a host computer. Simulink is a MATLAB-based graphical programming environment for modeling, simulating, and analyzing dynamic systems. Its primary interface is a graphical block diagramming tool and has a vast library of blocks representing different mathematical, logic, and programming operations. This specific package allows users to interface directly with the Parrot Mambo's onboard processor by converting Simulink's block diagram language into C code, which is readable to the UAV. This package allows for access to logged data mid-flight for post-flight analysis, uploading custom flight controllers, uploading custom state estimation algorithms, and other high-level logic. This package also comes with a simulation environment that allows the user to simulate the performance of proposed flight controllers on the drone with the use of the nonlinear dynamic model of the quadcopter, the environment, and dynamic behavior from the sensors. Conveniently, there are custom Simulink blocks available with this package allowing for the live

sharing of data between a host computer and drone during flight. This ability will be leveraged to create a decentralized communication network. For more information on the SSPPM see [11]. Another convenient feature of the SSPPM is the built in logic for state estimation which uses the sensor data, signal processing methods, and filters (Kalman and complementary) to estimate the UAV's state. The parameters associated with the state estimation were left at the default values. This feature was very important for reducing the development time.

SSPPM requires several other MathWorks components to run: Aerospace Blockset, Aerospace Toolbox, UAV Toolbox, Control System Toolbox, Signal Processing Toolbox, Simulink 3-D Animation, and Simulink Coder. These components are either included with the standard MATLAB/Simulink license or can be downloaded from the MathWorks website. The ROS Toolbox in MATLAB is also used to facilitate communications between computers (to be discussed in next section), see [12] for detail on this toolbox. The full Simulink block diagram that is uploaded to the UAVs can be found in the appendix.

3.3. Communications

Some drawbacks of the Parrot Mambo and its supporting package are the lack of Wi-Fi capabilities and the disadvantage of only being able to establish a Bluetooth connection between one UAV and one computer at a time. This requires creativity in developing a decentralized communication network for the platform.

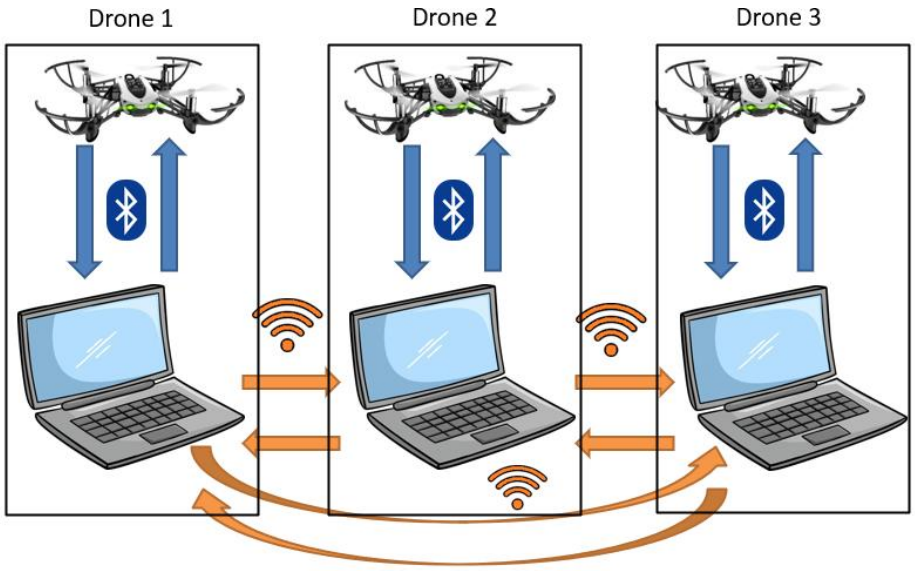

Figure 3.2. Communication network diagram

Figure 3.2 shows a diagram of the developed communication network. Each UAV establishes a Bluetooth connection with one computer using TCP/IP Bluetooth protocols. This is facilitated by the dedicated blocks in the SSPPM. Then, data is shared between computers using the ROS Toolbox via Wi-Fi. The ROS Toolbox allows for the sharing of data between agents or nodes. Data is sent to the network by nodes through “Publisher” blocks and received by other agents through “Subscription” blocks which subscribe to the published messages.

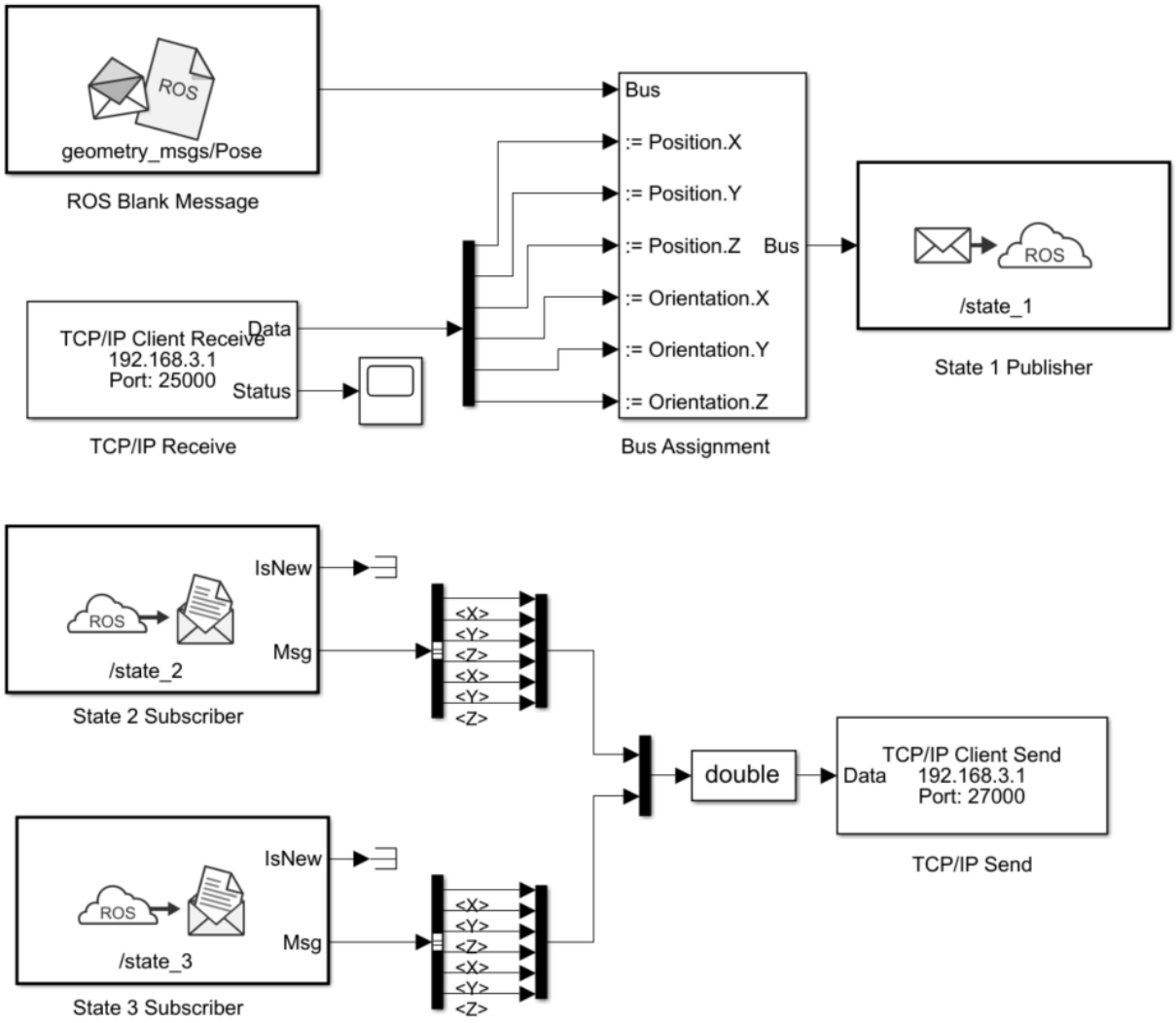

Figure 3.3. Simulink communication model

Shown in Figure 3.3 is the Simulink model which runs on each computer to facilitate communications between UAVs during flight; shown here is the model for UAV 1. The “TCP/IP Receiver” block receives the position and velocity of UAV 1, updates a blank ROS message, and then publishes this message to the network. Likewise, the other agents publish their states. The “State 2 Subscriber” and “State 3 Subscriber” blocks subscribe to the publications from the other agents and then send this data back to the drone via the “TCP/IP Send” block. In this way, each agent has access to the states of the other agents.

An important characteristic of such a network is the delay at which information is shared.

Quadcopters are inherently unstable and micro-UAVs such as these are prone to quick movements. Delays in a system like this can greatly affect performance. An experiment was conducted which sent a signal from a Parrot Mambo, to its host computer, then to another computer, back to the host computer, and then back to the Parrot Mambo. Both the original and echoed signals were logged and are shown in Figure 3.4. It was expirementally determined that there is about a quarter-second delay in the system from when one agent measures a signal to when another agent receives that signal.

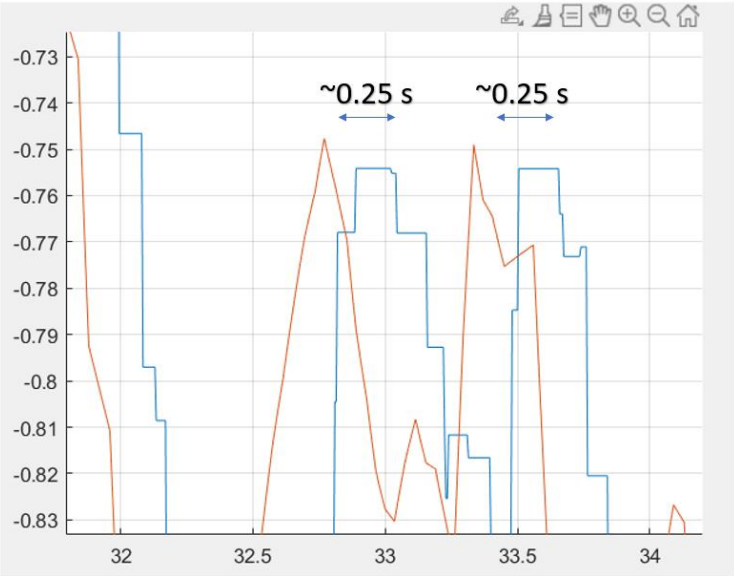

Figure 3.4. Communication delay

3.4. Simulation

As mentioned, the SSPPM comes with a very useful simulation environment, which was used to create a simulation of the formation control performance on the UAVs. Not included in the simulation are the communication delays nor the sensor dynamics and therefore it serves as a "best case scenario" simulation rather than one which is trying to emulate the real system as well as possible. The block diagram for the simulation is shown in Figure 3.5 below.

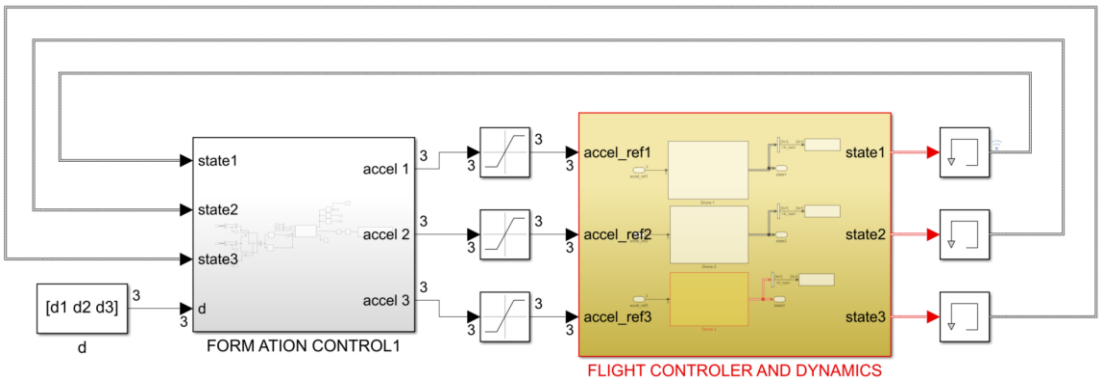

Figure 3.5. Formation control simulation model

The "Formation Control" block takes in inputs of the positions and velocities of each of the agents as well as the vector of desired edge lengths. Its outputs are the acceleration commands for each agent which are saturated at +/- 0.3 m/s, which is applied on the real hardware as well. The acceleration commands enter the "Flight Controller and Dynamics" block which passes these commands through the flight controller which outputs torques and forces on the agents. The dynamic model uses these external torques and forces to simulate the agent's states. These states are used directly, i.e., there are no sensor dynamics at play.

Three separate tests were run in simulation and on the real hardware. For each, the UAVs would takeoff and hover in some initial shape, and then begin running the formation control to acquire some final shape. The first test was starting in a large triangle and then converging to a smaller one. The second test was going from a smaller to a bigger triangle and the last test was going from a random starting shape to a medium sized triangle. The lengths of the initial and desired shapes are shown in Table 3.1.

Table 3.1. Desired edge lengths for tests

	Big to Small		Small to Big		Random Start	
	Initial	Desired	Initial	Desired	Initial	Desired
$\tilde{q}_{12}$	2.5 m	1.5 m	1.5 m	2.5 m	1 m	1.7 m
$\tilde{q}_{13}$	2.5 m	1.5 m	1.5 m	2.5 m	1.24 m	1.7 m
$\tilde{q}_{23}$	2.5 m	1.5 m	1.5 m	2.5 m	2 m	1.7 m

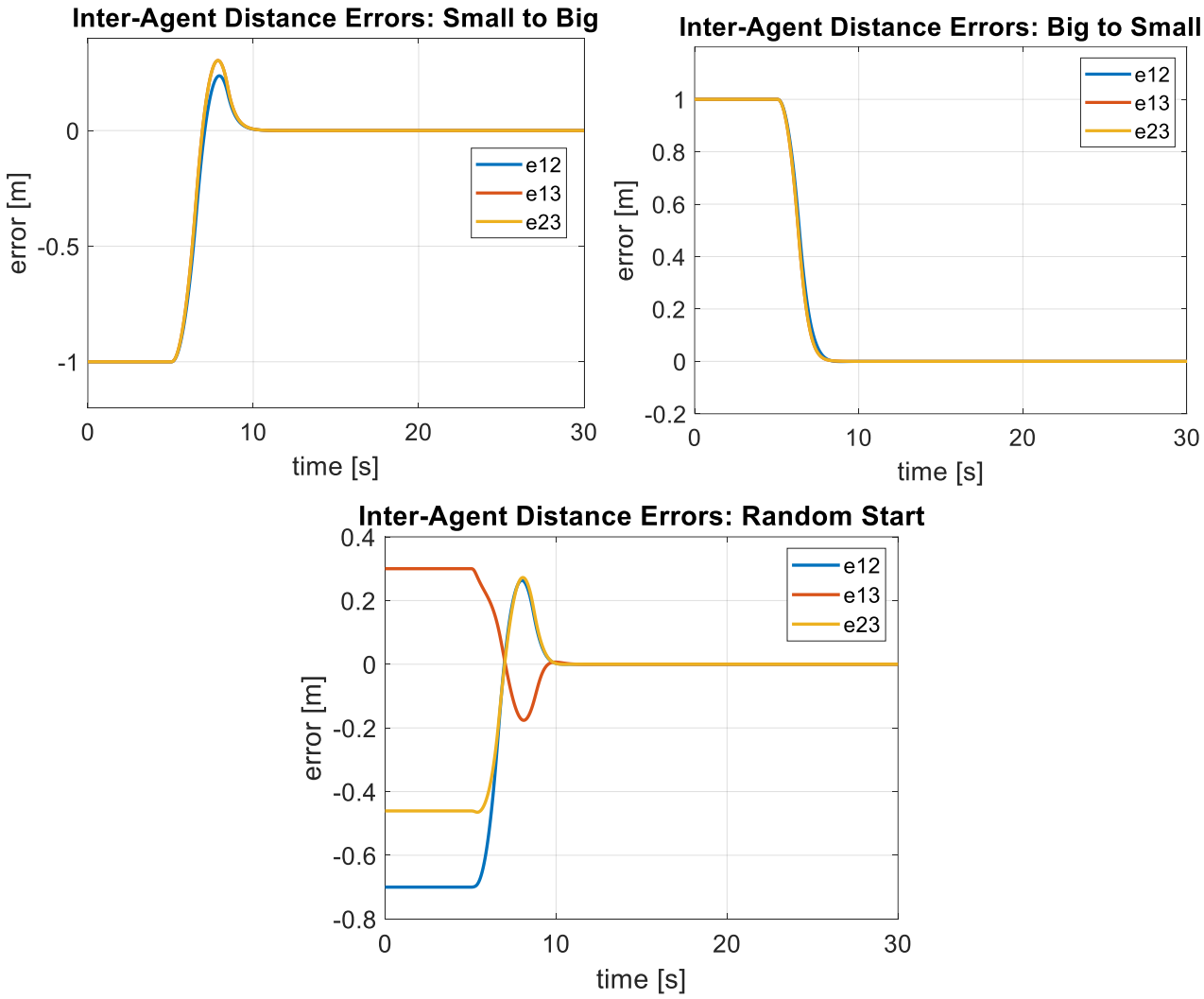

Figure 3.6. Formation control simulation results

Figure 3.6 shows the errors plots from the three separate tests done in simulation where the errors were defined in (20). In all cases the inter-agent distance errors converge to zero implying the desired shape was aquired. In each case, formation acquisition happened within five seconds of starting the formation control law, and there were no oscillations afterwards about the desired shape.

3.5. Testbed

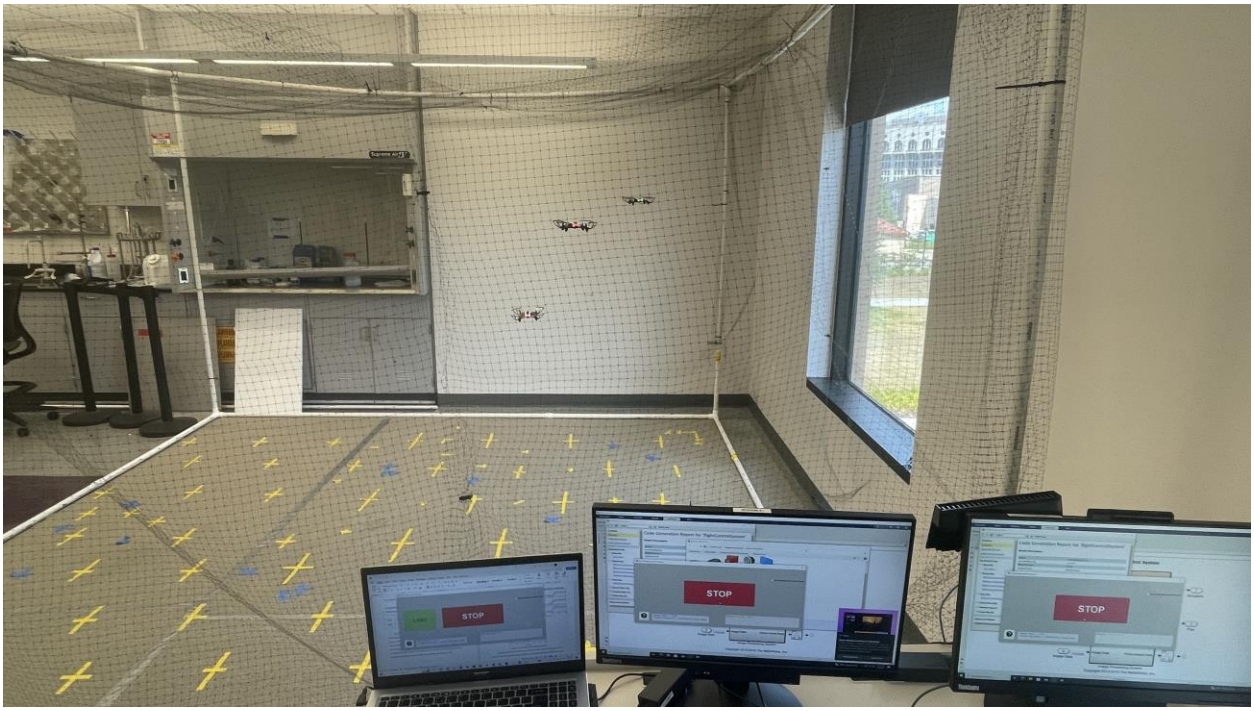

Figure 3.7. Experimental testbed

Figure 3.7 shows the testbed that was created. The three computers at bottom of the image are running the communication Simulink models in the background to facilitate the communications between the UAVs. The three UAVs can be seen in the middle of the figure. The testbed has an enclosure made of a PVC pipe frame that is 12 x 12 ft at its base and 8 ft in height. Surrounding the frame is a net which is there to safely catch the UAVs if they fly outside of the enclosure. The floor is carpet, which provides a soft landing should the UAVs fall out there air. On the carpet is tape which has been laid in crosses in a grid pattern. This is for the cameras at the bottom of the UAVs, to provide points of interest for the cameras for the optical flow process of estimating position and velocity.

3.6. Experimental Results

Table 3.2. Control gains

Sliding Mode Control Gains				PID Control Gains				Formation Control Gains	
	ω	ε	k		k_P	k_I	k_d	k_a	k_v
Pitch	12	30	100	Yaw	0.004	0	0.0012	0.001	0.5
Roll	12	30	100	Altitude	0.8	0.24	0.5		

The same three tests were done on the hardware using the Parrot Mambos, the Simulink communication models shown in Figure 3.3, and the models uploaded to the UAVs' internal processors which is shown in the appendix. The control gains used for the hardware tests are shown in Table 3.1. All of the gains were chosen through testing with a trial-and-error method. Note that these were the same gains used in simulation. For each case, five separate tests were run, and the data was averaged. The results are shown in Figures 3.8, 3,9, and 10 where the red error bars represent one standard deviation on either side of the mean across the five tests. Table 3.2 quantifies the performance of each experiment through three specifications. First, the settling time, T_s, indicates the time for the transients to die out. Second, we defined the average absolute value of the steady state error as

$$(e_{ij})_{ss} = \sum_{t=T_s}^{T_f} \frac{|e_{ij}|}{n}$$

where T_f is final time and n is the number of discrete data points between T_s and T_f. From now on, $(e_{ij})_{ss}$ will be referred to as just the average error. Last, it shows the percentage that $(e_{ij})_{ss}$ makes up of the corresponding desired distance, henceforth referred to as just the percentage error.

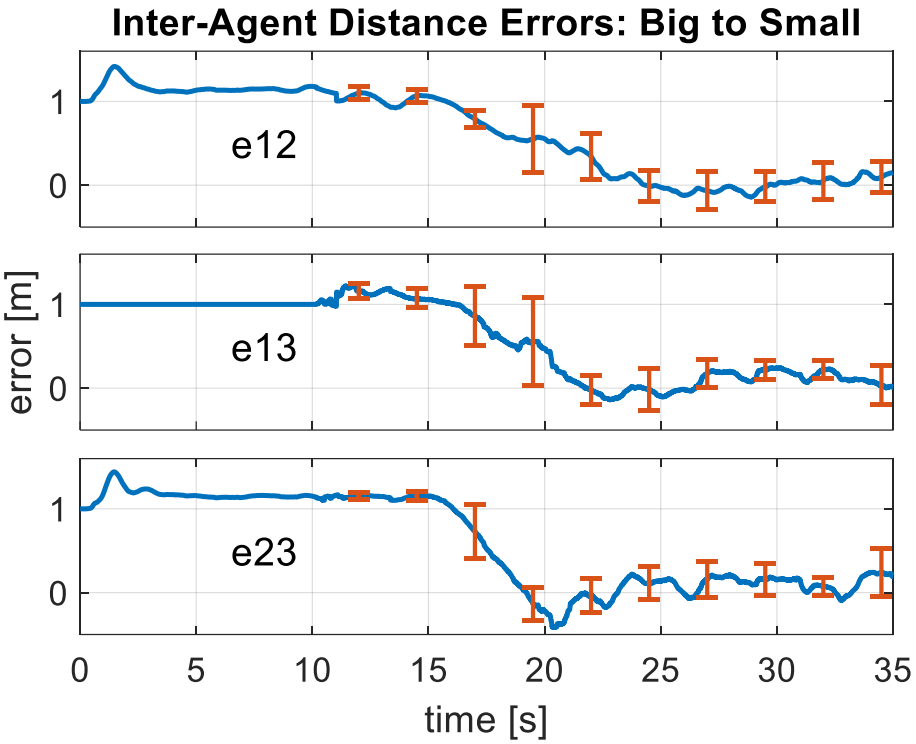

Figure 3.8. Formation control hardware results: Big to small

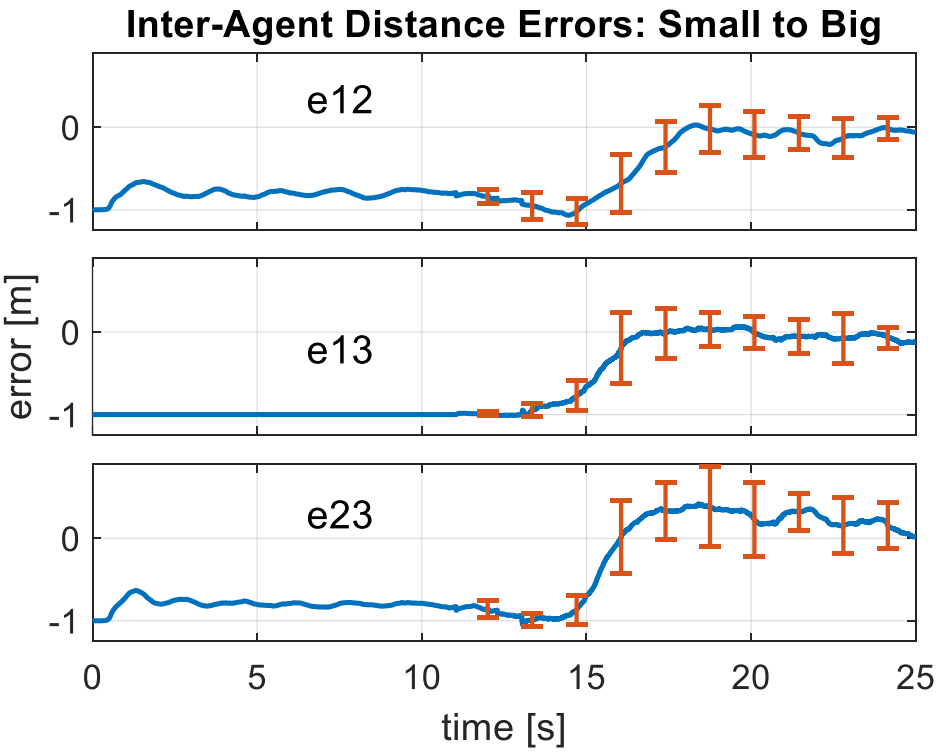

Figure 3.9. Formation control hardware results: Small to big

Figure 3.10. Formation control hardware results: Random start

Table 3.3. Experimental settling time and averaged absolute steady state error

	Big to Small			**Small to Big**			**Random Start**		
	T_s **[s]**	$(e_{ij})_{ss}$ **[m]**	**% of** d_{ij}	T_s **[s]**	$(e_{ij})_{ss}$ **[m]**	**% of** d_{ij}	T_s **[s]**	$(e_{ij})_{ss}$ **[m]**	**% of** d_{ij}
e_{12}	~11s	0.065	4.3%	~4s	0.069	2.8%	~8s	0.052	3.1%
e_{13}	~11s	0.129	8.6%	~3s	0.055	2.2%	~8s	0.102	6%
e_{23}	~11s	0.128	8%	~4s	0.249	10.0%	~8s	0.079	4.6%

Comparing the hardware results to the simulation results, a couple observations can be made. For one, the real results contain oscillations and unsmooth error plots. The distance errors also do not converge to zero like in simulation but oscillate in a range around zero. Both differences can be attributed to sensor noise/inaccuracies, communication delays, unmodeled

dynamics, and environment conditions which are not present in simulation. Table 3.2 displays that the “small to big” test showed the shortest settling time and “big to small” showed the largest. When looking at the average errors, there is not a clear difference between the three tests, implying that the size of the desired distances had little effect on the magnitude of the errors. The percentage errors show an inverse relationship to the size of the desired distances (outside of e_{23} in the “small to big” test). Since the errors are roughly the same across the tests, smaller desired distances seemed to result in higher percentage errors and vice versa. Lastly, one might notice that the error bars in the “big to small” test are smaller than in the others. This is likely due to randomness and a result of the small sample size used. Overall, the real-world testing demonstrates working but potentially improvable results.

Chapter 4. Conclusions and Recommendations for Future Work

The goal of this project was to create a 3D decentralized platform for testing formation controllers on mini quadcopters. The platform was successfully developed using Parrot Mambos as the UAVs, three computers, and with Simulink/MATLAB for the software development. This platform will serve later researchers in the iCORE lab at LSU for formation control education and research. In this project specifically, the double-integrator model was used for testing the distance-based formation control acquisition and was able show results with the hardware of a steady state error of up to 10% of the desired distance when averaging across several tests.

For future work, there are two paths of choice to try to improve the suboptimal performance. The first would be to spend more effort to try to improve the current setup. This could be done by trying to better tune the gains used in the high and low-level control loops. The communication network can be improved by reducing the delays, if such a solution is possible. There also might be other flight controllers that perform better than the sliding mode controller such as an adaptive controller. A motion tracking camera could be added for a ground truth which could help assess the performance of the system. The second path would be to build custom quadcopters, which would give the ability to have much better hardware for a similar or cheaper price. A UAV with a higher processing speed for control calculations, less noisy and more accurate sensors, and higher quality motors would perform much better than the Parrot Mambo. This is very feasible today because of the many opensource DIY quadcopters available online. The downside of this would be the increase and development time and the challenge of estimating position locally without the use of GPS or cameras. Integrating receivers and transmitters into the UAV would cut out the need for the three computers and give more potential for scalability. A DIY quadcopter would also not lock the project into using the

MATLAB/Simulink environment and the SSPPM, giving flexibility to use other software and programming languages.

Appendix A. Chapter 3 Supplementation

The following describes the Simulink model that is uploaded to Parrot Mambos. Since the model is too large to include in one image so it was broken up into several images moving from left to right.

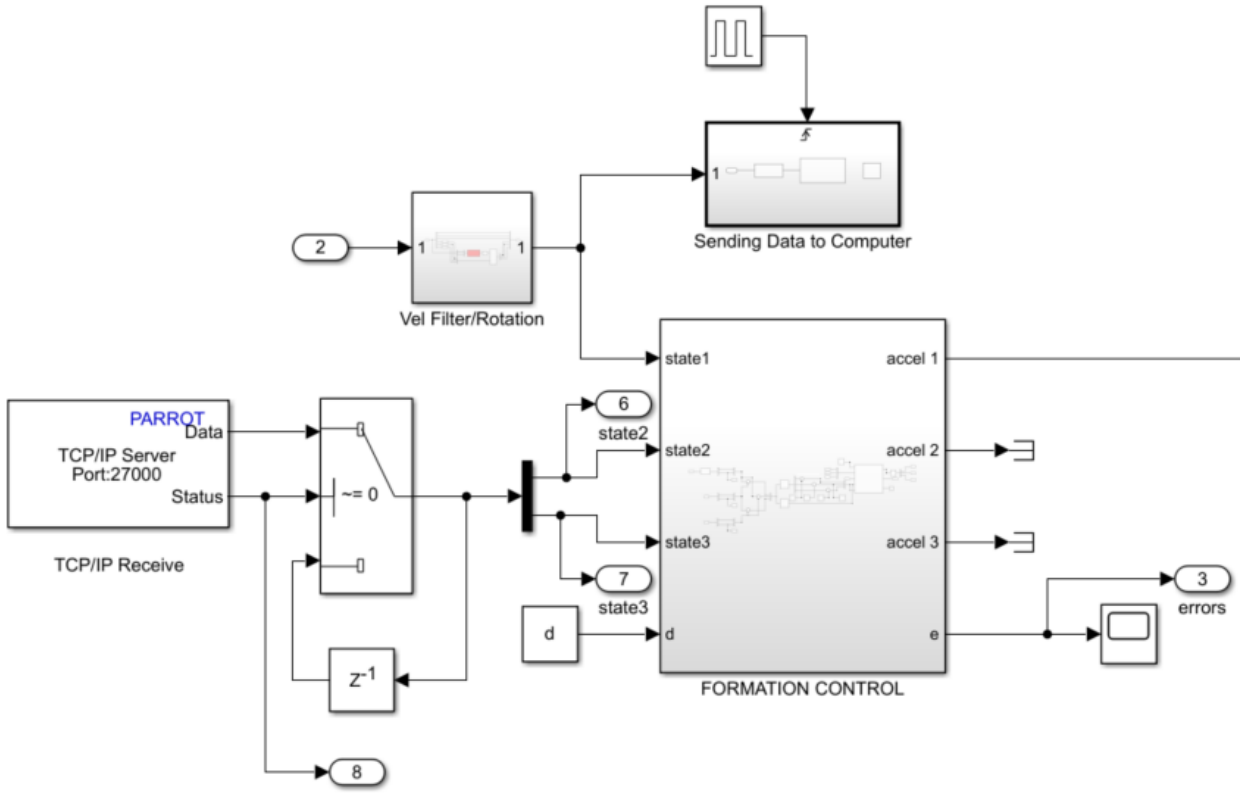

Figure A.1. UAV model (1)

In Figure A.1 the “TCP/IP Receive” block links to the “TCP/IP Send Block” shown in Figure 3.3. This means it receives the states of the other two UAVs from its host computer. There is a switch and delay logic following this block so that the previous data point is used if the current one is not received. The “Vel Filter/Rotation” block transforms the estimated velocities of the UAV into the inertial frame and filters the signals using an exponential weighting filter with a forgetting factor of 0.98 to smoothen them. This updated state is sent to

the “Sending Data to Computer Block”. Figure A.2 shows the inside of this block. The data is

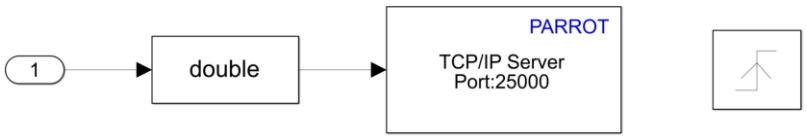

Figure A.2. UAV model (2)

converted to type double and is sent to the host computer using a TCP/IP Send block which links with its coupled TCP/IP Receive block in Figure 3.3. There is a trigger attached to this block controlled by a pulse generator with a period of 0.05 seconds and a pulse width of 10%. It was determined experimentally that sending data any faster than this causes the drones to fly erratically.

The “Formation Control” block takes as inputs the three states and the vector of desired edge lengths. One of the states comes directly from the state estimator of the host drone, and the other two states are sent from the Simulink Communication Model. The output is an acceleration command given by the double integrator model which is saturated at +/- 0.3 $\frac{m}{s^2}$.

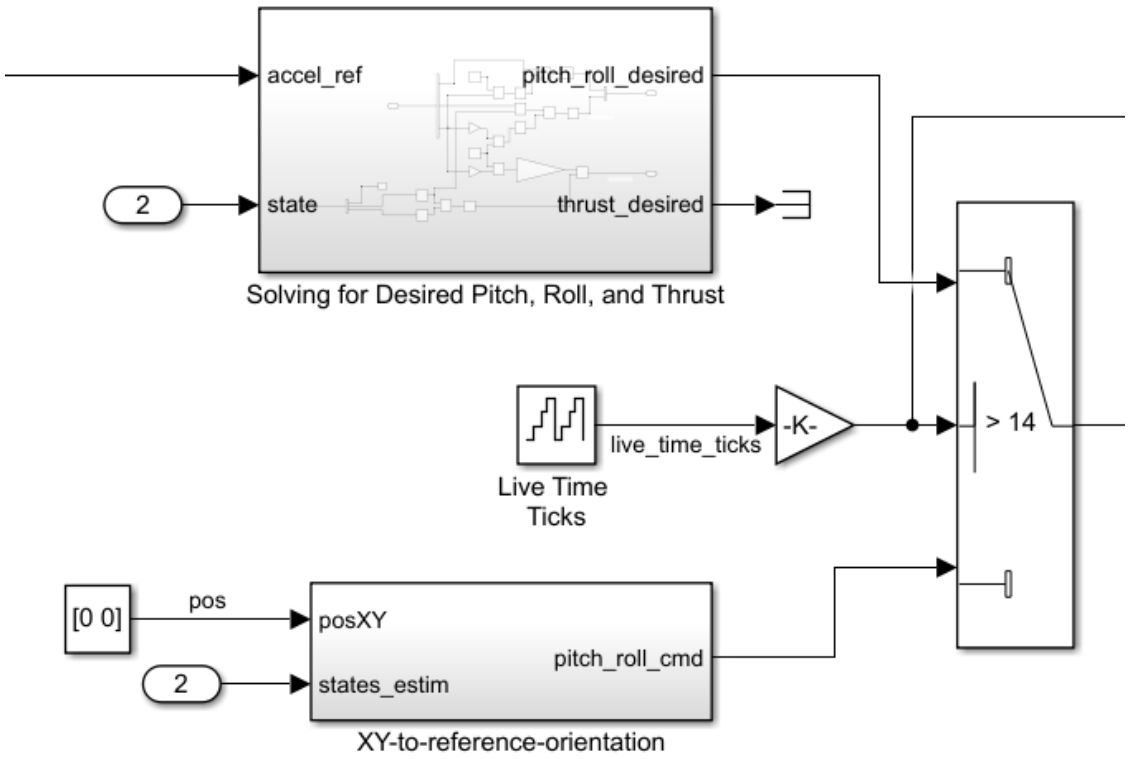

Figure A.3. UAV model (3)

The next part of the model, shown in Figure A.3, dictates the pitch and roll commands to be sent to the flight controller. For the first 14 seconds the UAV hovers (the switch controls this). During this period, the pitch and roll commands are dictated by PID control which tries to maintain initial position in the XY place. After 14 seconds, the pitch and roll commands come from the “Solving for Desired Pitch, Roll, and Thrust” block which takes the UAV’s state and the high-level acceleration command and outputs the pitch and roll angles given by Equations (23) and (24).

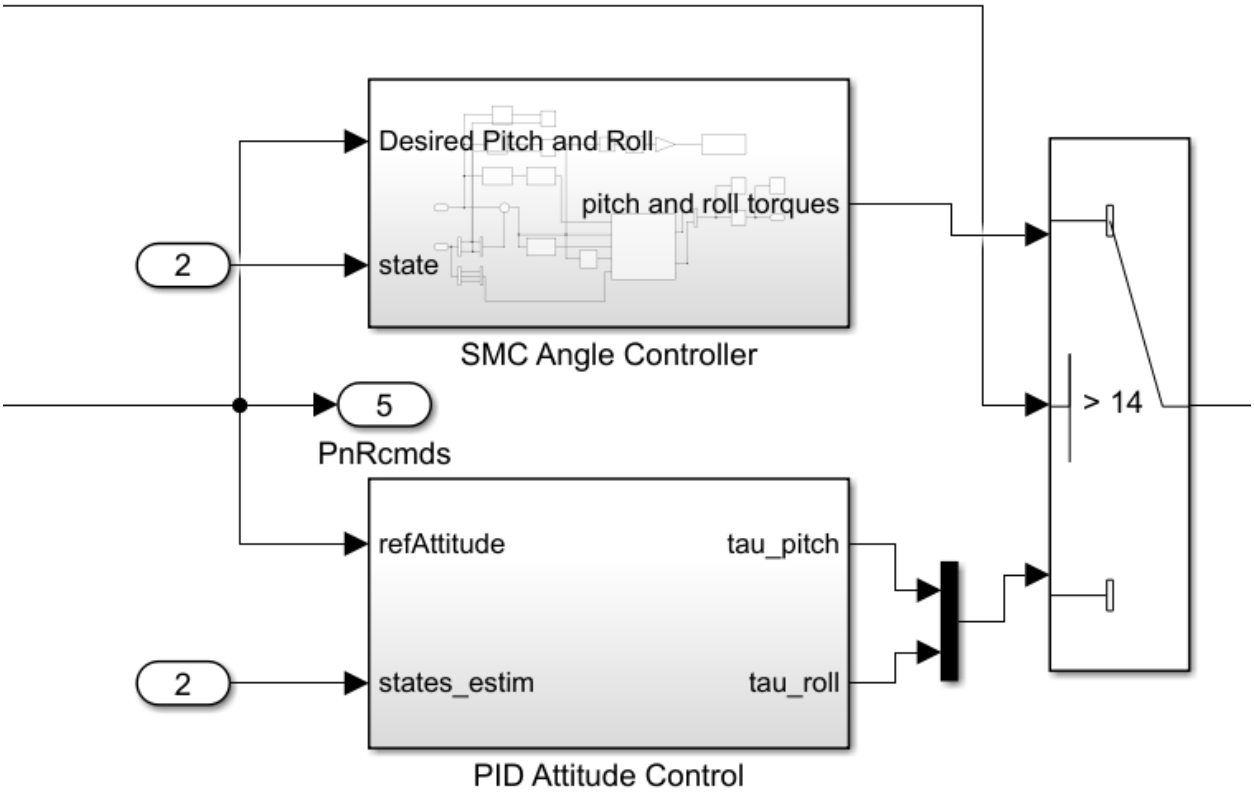

Figure A.4. UAV model (4)

Figure A.4 shows how the control torques are calculated. As mentioned above, during the 14-second hovering period, the UAV uses simple PID control. After 14 seconds the drone uses the adapted sliding mode controller given by Equations (27) and (32).

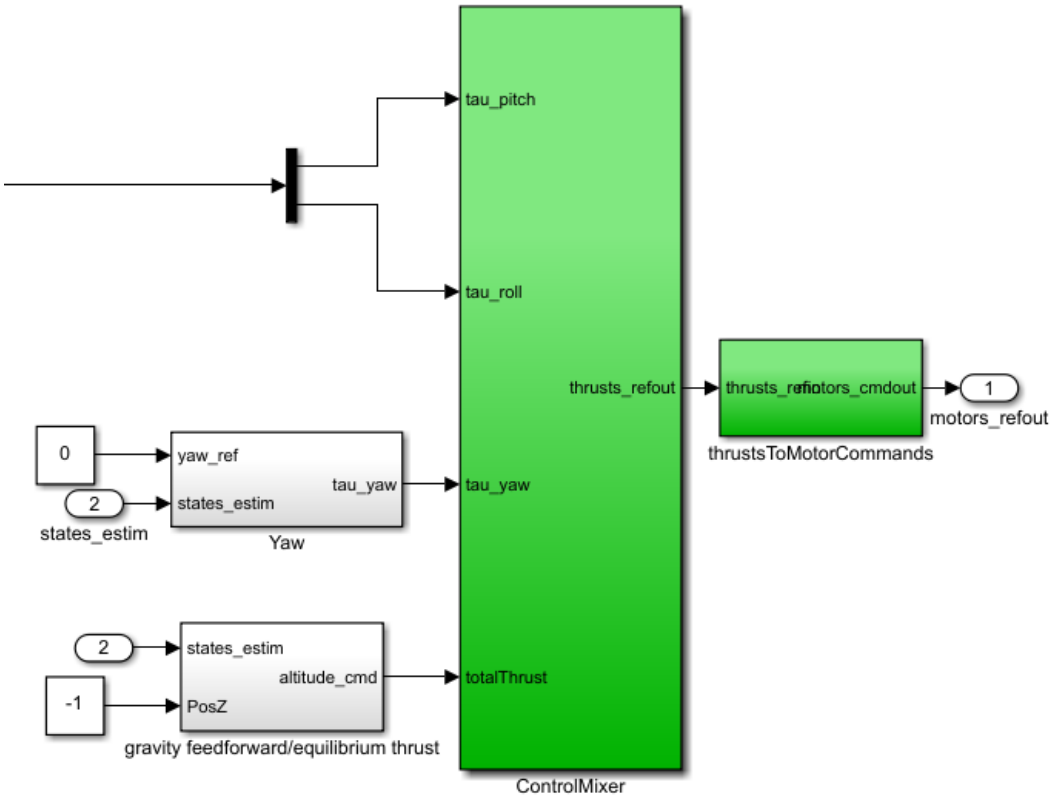

Figure A.5. UAV model (5)

The “ControlMixer” block in Figure A.5 receives the output from Figure A.4. as well as outputs from the yaw and attitude control blocks which use PID control given by Equations (33) and (34) that try to maintain the initial heading and a height of -1 meters. The “ControlMixer and “thrustToMotorCommands” use Equation (8) to calculate the motor pulse width modulated commands which is the final output of the UAV model.

Appendix B. Instructions for Testbed Use

This section serves as an instructional guide for using the developed testbed. As mentioned, there is one computer for each UAV. All three computers and UAVs need to be cooperating and running together for this testbed to work as intended.

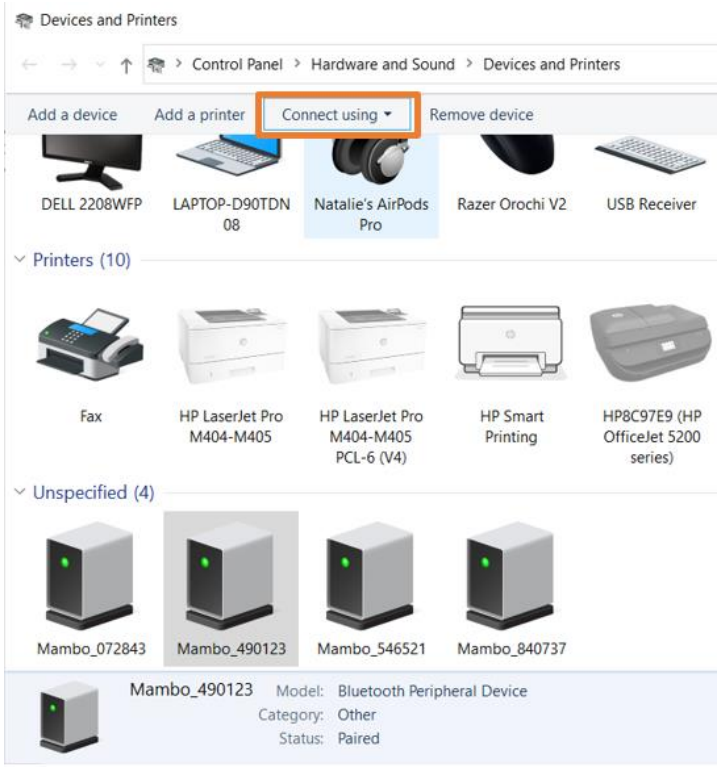

Figure B.1. Establishing Bluetooth connection

` The first step is to turn on all three computers and connect the three Parrot Mambos to their host computers. To do establish a new connection between Parrot Mambo and a computer see reference [11]. Once established, open Settings on Windows, then click Devices, then Device and Printers. Select the correct UAV then select “Connect using” as seen in Figure B.1. From the drop down menu, select “Access point” which will connect the UAV and computer. Do this for all three UAVs/computers.

The next step is to open all of the necessary files on each computer to operate the test bed. The files can be found in the following folders:

- Computer 1 - “C:\Users\12259\MATLAB\Projects\examples\asbQuadcopter23”
- Computer 2 – “C:\Users\msnell5\MATLAB\Projects\examples\asbQuadcopter2”
- Computer 3 - “C:\Users\msnell5\MATLAB\Projects\examples\asbQuadcopter2”

Then, on each computer there are four files which need to be opened, which are called: “StartupScript.m”, “FCcommunications.slx”, data_plotting.m”, and “asbQuadcopter.prj”.

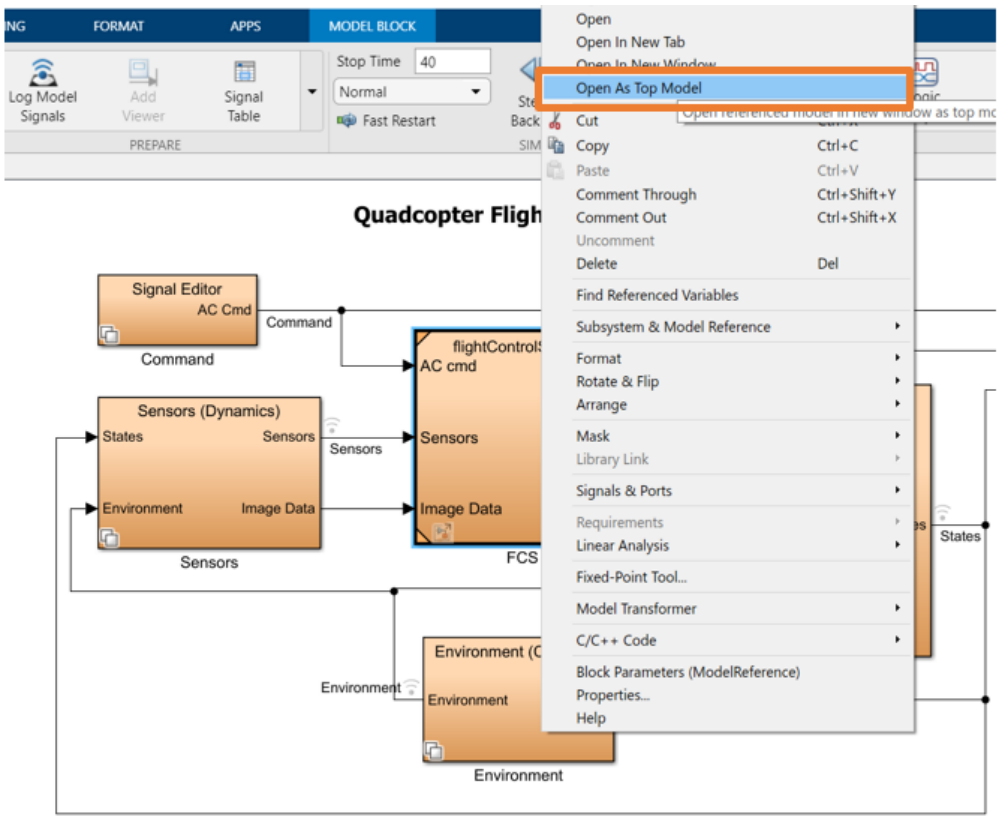

Figure B.2. Uplaoding flight controller

“asbQuadcopter.prj” is a project file that, when opened, will open the Simulink model for the flight controller model and automatically put key variables for operating the flight controller into the workspace. This model contains the state estimation, inner and outer control loops as discussed in Section 2.2, and the data logging blocks. To upload this model to a UAV, right click the block labelled, “flightControlSystem” and then select “Open As Top Model” as shown in Figure B.2. This will open a new window. Select “Build, Deploy, and Start” under the Hardware

tab, and the model will build and upload to the drone. This should be done for all three UAVs/computers.

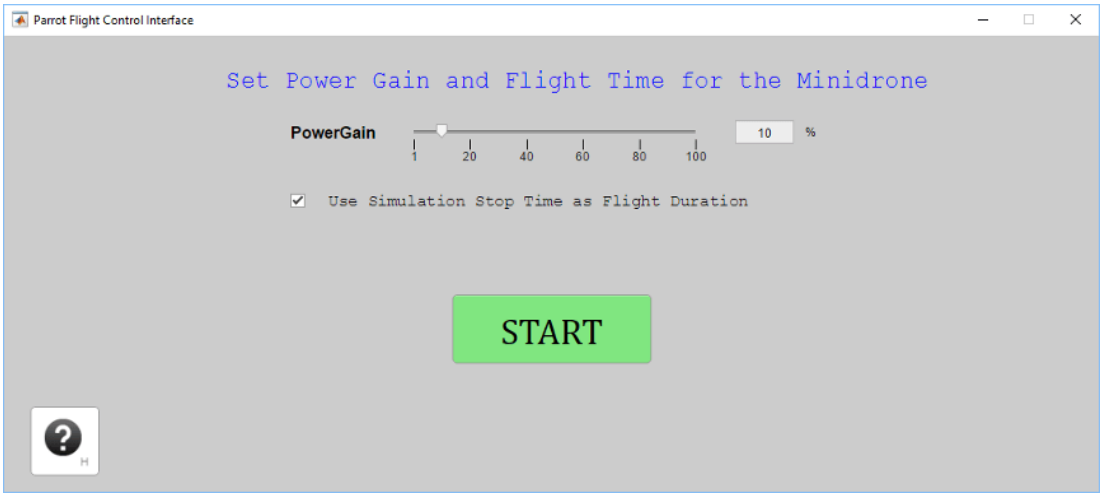

Figure B.3. Flight Control Interface

Figure B.3 shows the Flight Control Interface which will open after the model has been built and deployed onto a UAV. This interface is used to start and stop the drones. Set the “PowerGain” to 100%. After clicking start, the UAV will take off and follow the instructions from the flight control model. It will run for 40 seconds but can also be manually stopped by clicking stop.

```
%% Ros Setup
setenv('ROS_IP','167.96.77.103')
rosinit

%% Formation Control Variables
%Initial Position Vectors
q12i=[-0.75; 3*sqrt(3)/4; 0];
q13i=[ 0.75; 3*sqrt(3 )/4; 0];
q23i=[1.5; 0; 0];

%Desired Distances
d12=2.5;
d13=2.5;
d23=2.5;
d=[d12 d13 d23];

ka=0.01/10;
kv=0.5;
```

Figure B.4. Master StartupScript

```
%% Ros Setup
setenv('ROS_IP','167.96.140.183')
setenv('ROS_MASTER_URI','http://167.96.77.103:11311')
rosinit

%% Formation Control Variables
%Initial Position Vectors
q12i=[-0.75; 3*sqrt(3)/4; 0];
q13i=[ 0.75; 3*sqrt(3 )/4; 0];
q23i=[1.5; 0; 0];

%Desired Distances
d12=2.5;
d13=2.5;
d23=2.5;
d=[d12 d13 d23];

ka=0.01/10;
kv=0.5;
```

Figure B.5. StartupScript

The purpose of the “StartupScript” file for each computer is to establish the ROS network and define several key variables which are used in the Simulink flight control model uploaded to the drone. One of the computers will serve as the ROS master (see [12]). Figure B.3. shows the

master script. Ensure that the specified IP address is the correct one corresponding with the Eduroam network. To do so, go to the Command Prompt and type in the command “ipconfig”. The correct IP address will be located next to “IPv4 Address”. Run the script, which will open the “roscore” and will establish a node in the network for the computer serving as the master. The same script should be run on the other two computers, but will be slightly different, see Figure B.4. Ensure that the IP address next “ROS_IP” is the address for the computer the script is being run on and that the IP address next to “ROS_MASTER_URI” is the address for the computer serving as the master.

Also included in this script are several crucial variables which can be customized. The general variable “qhji” is the relative position vector point from agent j to agent h in the initial starting position as defined by the frame of reference taped to the floor of the testbed. It is important that these vectors are the same on each computer when running the scripts. The general variable “dhj” is the desired distance between agent h and agent j. The variables “ka” and “kv” are the user-defined formation control gains mentioned in Section 2.3. After running this script on all three computers, the ROS network should be completely set up.

The file named “FCcommunications” serves to facilitate the communications between the computers and distributes the states of the agents around the system. To operate the test bed, a user should (in quick succession!) click “start” on the Flight Control Interface on each computer and then quickly run the “FCcommunications” Simulink model on each computer. Ideally, all of the UAVs should take off, and the three communication Simulink models will be running in the background connecting the system of agents.

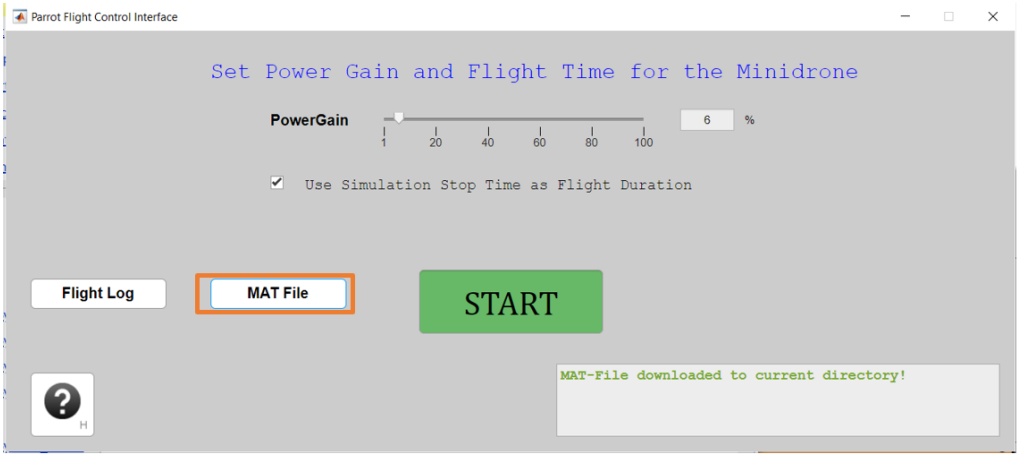

Figure B.6. Downloading flight data

After the experiment is over, the logged flight data can be downloaded and used for analysis. To do so, click “MAT File” on the Flight Control Interface as seen in Figure B.6. Then double click on “RSdata.mat” in MATLAB under Current Folder. This will download the logged data into the workspace. Then the “data_plotting” script can be run with this data to see the inter-agent distance errors, the tracking of pitch and roll from the sliding mode controller, and the acceleration commands from the double-integrator law.

References

1. M. de Queiroz, X. Cai and M. Feemster, Formation control of multi-agent Systems: A graph rigidity approach, Hoboken, NJ:Wiley, 2016.

2. Jiménez-González, A., Martinez-de Dios, J. R., & Ollero A. Testbeds for Ubiquitous Robotics: A survey. *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1487-1501, 2013.

3. Y. Zou, Z. Zhou, X. Dong, and Z. Meng, "Distributed Formation Control for Multiple Vertical Takeoff and Landing UAVs With Switching Topologies," *IEEE/ASME Trans. Mechatr.*, Vol. 23, No. 4, pp. 1750-1761, 2018.

3. Y. Zou, Z. Zhou, X. Dong, and Z. Meng, "Distributed Formation Control for Multiple Vertical Takeoff and Landing UAVs With Switching Topologies," *IEEE/ASME Trans. Mechatr.*, vol. 23, no. 4, pp. 1750-1761, 2018.

4. K. Guo, X. Li, and L. Xie, "Ultra-Wideband and Odometry-Based Cooperative Relative Localization With Application to Multi-UAV Formation Control," *IEEE Trans. Cybern.*, Vol. 50, No. 6, pp. 2590-2603, 2020.

5. J.F. Guerrero-Castellanos, A. Vega-Alonzo, S. Durand, N. Marchand, V.R. Gonzalez-Diaz, J. Castañeda-Camacho, and W.F. Guerrero-Sánchez, "Leader-Following Consensus and Formation Control of VTOL-UAVs with Event-Triggered Communications," *Sensors*, Vol. 19, No. 24, 5498, 2019.

6. S. Kang, M. Park, and H. Ahn, "Distance-Based Cycle-Free Persistent Formation: Global Convergence and Experimental Test with a Group of Quadcopters," *IEEE Trans. Ind. Electr.*, Vol. 64, No. 1, pp. 380-389, 2017.

7. V.P. Tran, M. Garratt, and I.R. Petersen, “Switching Time-Invariant Formation Control of a Collaborative Multi-Agent System using Negative Imaginary Systems Theory,” Control Eng. Prac., Vol. 95, 104245, 2020.

8. B. Randal, "Quadrotor Dynamics and Control Rev 0.1," *All Faculty Publications*, Paper 1325, 2008.

9. N. P. Nguyen, N. X. Mung, H. L. N. N. Thanh, T. T. Huynh, N. T. Lam and S. K. Hong, "Adaptive Sliding Mode Control for Attitude and Altitude System of a Quadcopter UAV via Neural Network," *IEEE Access*, Vol. 9, pp. 40076-40085, 2021.

10. H. K. Khalil, “Nonlinear Systems,” 3rd Edition, Prentice Hall, Upper Saddle River, 2002.

11. Parrot Minidrones Support from Simulink, https://www.mathworks.com/hardware-support/parrot-minidrones.html (accessed on 3/21/2002).

12. *Ros Toolbox*. ROS Toolbox Documentation. (n.d.). https://www.mathworks.com/help/ros/

13. Greenwood, D. T. (2002b). *Principles of dynamics*. Prentice Hall.

Vita

Matthew Snellgrove was born and raised in Baton Rouge, Louisiana. He graduated from LSU with a Bachelor of Science in Mechanical Engineering in May of 2022. He plans to receive his Master of Science degree in December 2023. His research interest is in mechanical systems.