MicroCART 2016-2017
Design Document

Group: May1716

Team Members

Brendan Bartels	–	*Controls Software Key Concept Holder*
Kristopher Burney	–	*Ground Station Key Concept Holder*
Joseph Bush	–	*Quadcopter Software Key Concept Holder*
Jacob Drahos	–	*Team Webmaster*
Eric Middleton	–	*Hardware Maintainer*
Tara Mina	–	*Team Communications Leader*
Andrew Snawerdt	–	*Control Systems Key Concept Holder*
David Wehr	–	*Team Leader*

Advisors

Dr. Phillip Jones
Dr. Nicola Elia

Team Email: *may1716@iastate.edu*

Team Website: *http://may1716.sd.ece.iastate.edu*

Revision Date: *Monday, November 28th 2016*
Version: 2.0

Contents

Figures

Tables

1 Definition of Terms

Below are some of the terms we use regularly in this design document:

Term	Description
CLI	Command Line Interface, will be used to take in user input to get data from the quadcopter and give it flight commands
ESC	Electronic Speed Controller, a component between the Zybo board and the motor, which takes in a PWM signal from the board and converts this to an amplified current to control the speed of the motor
GUI	Graphical User Interface, will be used to take in user input to get data from the quadcopter and give it flight commands
LiDAR	Light Detection and Ranging, a laser used to detect and measure distances (ranging)
MicroCART	Microprocessor-Controlled Aerial Robotics Team, our senior design team
PCB	Printed Circuit Board, will be designed to include a battery voltage regulator and power distributor for improved power management on the quadcopter
PID	Proportional-Integral-Derivative, a commonly used type of feedback controller
PWM	Pulse-Width-Modulation, a digital signal with varying duty cycle but constant period, which is received by each of the ESCs to control the speed of each motor

Table 1: Definition of Terms

2 Introduction

The MicroCART (Micro-processor Controlled Aerial Robotics Team) senior design project has been passed down from team to team since 2006, developing a quadcopter as a research platform. Up to now, the MicroCART quadcopter (**Figure 1**) has been flying in the Distributed Sensing and Decision Making Laboratory, using the twelve-camera infrared tracking system for navigation. The quadcopter has been gradually improved by the work of each senior design team, and it currently has much of the hardware necessary to accomplish our goals for this academic year.

Figure 1: MicroCART Quadcopter

2.1 Project Statement

We intend to create a modular platform for research in controls and embedded systems, building on the current system developed by previous teams. In addition to improving the modularity of the quadcopter system, we intend to advance its abilities and functions, including developing the ability to fly autonomously through a sequence of user-specified waypoints and to fly outdoors, without the help of the infrared tracking camera system in lab.

2.2 Purpose

By creating a modular platform, our advisors wish for any controls student to be able to design their own controller, including new types of controllers and test it with our system easily. Additionally, our advisors want our project to demonstrate the capabilities of the Electrical and Computer Engineering

Department, so we hope to develop a system that can execute more impressive abilities, including autonomous flight, flying outdoors using GPS navigation, and also potentially performing advanced flight maneuvers. Having these more impressive capabilities to demonstrate will help excite new students, give them a friendly, hands-on platform to learn about controllers, and better represent the talents of our department to visitors, other interested students and faculty members. Furthermore, if we develop a platform that works well and is easy to use, this system will likely be used and added into the second semester course for controls systems, EE 476, where it can help students gain a more intuitive understanding of PID controllers and give them a platform to test their designs.

2.3 Goals

We plan to design several PID controllers for each direction of movement. Our plan will differ from approaches used in the past, deriving the controller from a systematically developed mathematical model of quadrotor dynamics, in accordance with the approach developed in a thesis written by one of the advising graduate students. Thus, we will initially conduct system identification of our quadcopter by taking measurements and performing data analysis, and afterwards utilize these parameters to characterize and implement the linear control systems necessary for controlling the movement of the quadcopter. We will also create a user interface that can be used to select waypoints which the quadcopter will follow while in autonomous flight. Eventually, we also want the quadcopter to be able to fly outdoors, without the infrared camera tracking system in the lab, by using GPS for navigation instead and a LIDAR sensor to detect the presence of obstacles located underneath the quadcopter.

Our advisors, Dr. Jones and Dr. Elia, had some high-level goals for what he wanted from our project, which can be summarized by the following points:

- Build a modular system for the quadcopter, so that each component can be easily removed and replaced by another of the same functionality, without breaking the operation of the entire system
- Design a control system for the quadcopter from a mathematical model representation of the quadcopter system, rather than from iterative testing procedures as done from previous teams
- Decrease the communication latency between the quadcopter and the ground station to be less than 10 milliseconds on average, to increase the stability of the system overall, which will most likely require WiFi communication rather than the current Bluetooth communication system
- Ensure that the communication system and controls system work together robustly, such that if any data packets are dropped, the quadcopter does not lose control and potentially damage itself
- Create a user-friendly GUI with nice multi-panel layout and click navigation features for the user to easily specify waypoints for the quadcopter to travel between
- Throughout the development process, create detailed and user-friendly documentation and tutorials for future MicroCART teams to follow and learn from, including video tutorials as well, and keeping the two-decade-long Wiki page up-to-date

3 Deliverables

The quadcopter system can be compartmentalized into three major subsystems or sub-projects: the development of the quadcopter software, the ground station, and the PID controls systems on the quadcopter. We will need to work on each of these components to transform it to a particular, useful state which will be essential to meet our objectives. In addition, there are overall, all-encompassing aspects of the project that we need to address, including improving on the modularity of the system.

3.1 Current Quadcopter Software

3.1.1 Autonomous Flight Capability

The quadcopter has an autonomous mode in which it attempts to remain at a fixed point in space using the infrared camera tracking system in the lab. It is currently unstable and unreliable. Our team plans to:

- Update the controllers to improve the performance of this autonomous ability
- Add the ability to receive waypoints from the ground station and fly towards the given point, regardless of the current location (within the Coover 3050 IR tracking system)

Figure 2: Tracking System Camera

3.1.2 Improved Communication System

Communication between the ground station and quadcopter is currently done using bluetooth, which limits the range, bandwidth, and latency of our communication. On the quadcopter, we will:

- Replace the bluetooth receiver with a WiFi receiver, which will improve these three metrics [1]

- Design a communication system that will support both low-latency transfer of real-time data over UDP, as well as reliable data transfer for TCP
  - This will require us to update the command structure on the quadcopter to differentiate between UDP and TCP

3.1.3 Ability to Support Flight without Ground Station

The current quadcopter software requires that the ground station be connected in order for any flight to start, which is unnecessary for flight modes that don't require data from the camera system. We plan to:

- Remove requirement for any flight mode not needing data from the camera system, such as:
  - Manual flight mode
  - GPS-based autonomous flight modes
- Add a watchdog timer that will monitor communication link in modes requiring camera data
  - Example mode: indoor autonomous mode
  - Will force the system into manual mode if communication with the base station is lost

3.1.4 LiDAR Altitude Control

Once we have basic stabilized flight using the camera system, we will:

- Implement a LiDAR system which will allow autonomous control of altitude
  - This system will involve a single LiDAR pointing downward
  - LiDAR will be used to measure the distance to the ground
- The LiDAR distance reading will then be added into the PID control loop as the Z position

Figure 3: LiDAR Sensor

3.1.5 GPS Navigation Capability

The quadcopter currently receives position information for the camera system in 3050, which limits our autonomous capabilities to that specific area. Once we have basic stable flight using the camera system, and LiDAR control for altitude, we plan to:

- Integrate GPS into the system

- Use the GPS data to add the ability to navigate autonomously outside of the camera system.
  - Using the GPS for the horizontal position and LIDAR for Z position the quadcopter will be able to navigate outside

3.1.6 Running Linux on second core

The Zynq-7000 SoC, as shown in **Figure 4** below, runs the software on the quadcopter, and it contains two ARM processor cores [6]. Currently, all of the quadcopter software runs on only one of the cores, while the other core is unused. To better support research, we will:

- Configure the unused core to run linux, which will expose higher level functionality to the programmer
- Create a set of libraries that will allow programs to communicate with the real-time control program running on the other core.
  - These will allow researchers to use higher-level functionality, such as the computer vision library OpenCV, on the quadcopter without much effort
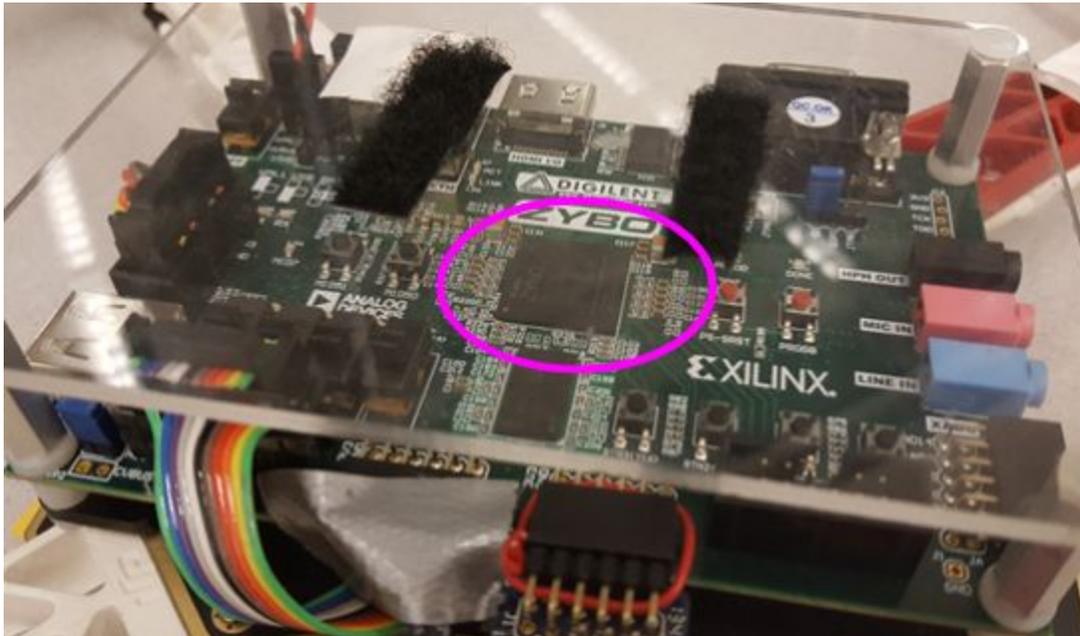
Figure 4: Zync-7000 SoC on Zybo Board

3.2 Ground Station

3.2.1 Real Time Communication

The current system has a limited amount of communication between the quadcopter and the ground station. We want to have feedback from the quadcopter at all times in order to verify that we are performing the correct functions.

- Constant real time communication between quadcopter and ground station
- Ability to log data at all times from quadcopter, in both manual and autonomous mode

3.2.2 Back End

The back end will be designed to be a modular piece of the ground station. Any front end, whether it be a command line interface or a graphical user interface, will connect to the back end via a socket connection. This backend will accept input through this socket and forward it through another socket connected to the quadcopter.

- Complete connection to quadcopter through a back end
- Server-Client relationship between back and and front end
- Command pass through from front end to quadcopter

3.2.3 Command Line Interface (CLI)

A command line interface (CLI) will provide the lowest level of front end ground station control. Users will be able to set and request any relevant variables such as PID constants and pitch, roll and yaw set points. This low level bare bones control will allow for a fast testing environment as well as the ability to quickly check our understanding of the system as a whole. The CLI will connect via a socket to the modular back end below it.

The following command abilities for the user should be implemented into the command line interface:

- Disable printing on backend process
- Display the menu of commands available to the user
- Get current quadcopter information
- Set PID constants for the different control systems on the quadcopter
- Choose waypoints for quadcopter to follow

3.2.4 Graphical User Interface (GUI)

The current GUI system has a rather limiting set of options. If connection issues occur between the GUI and the quad, the only thing we know is that the connection was lost. While utilizing our new real time control, we will implement a more descriptive error detection and recognition. Continuing with the real time control, we will have constant updates of VRPN data as well as the current and desired position of the quad. We intend for the GUI to extend the usability of the CLI and provide easier control over high level aspects of the system.

- Visual command passing to backend through socket connection
- Graphical map waypoint selection
- Use the constant communication that the back end has with the quad to provide constant useful updates to the user

3.3 Hardware Improvements

Because this project has been passed down over many semesters with the focus of each team being to add functionality to the quadcopter, the actual system hardware is currently in a non-ideal state. Wire connections are not very secure, some wire connections are being held together with tape, and some plugs are very tight. Along with the rewiring of the quadcopter in general, we would like to

design our own PCB which will regulate power on the quadcopter system. The old system utilizes 4 AA batteries which is unnecessary weight. These batteries also could not be fully charged as 4 AA batteries when fully charged is about 6 volts, when the board is powered off of 5 volts.
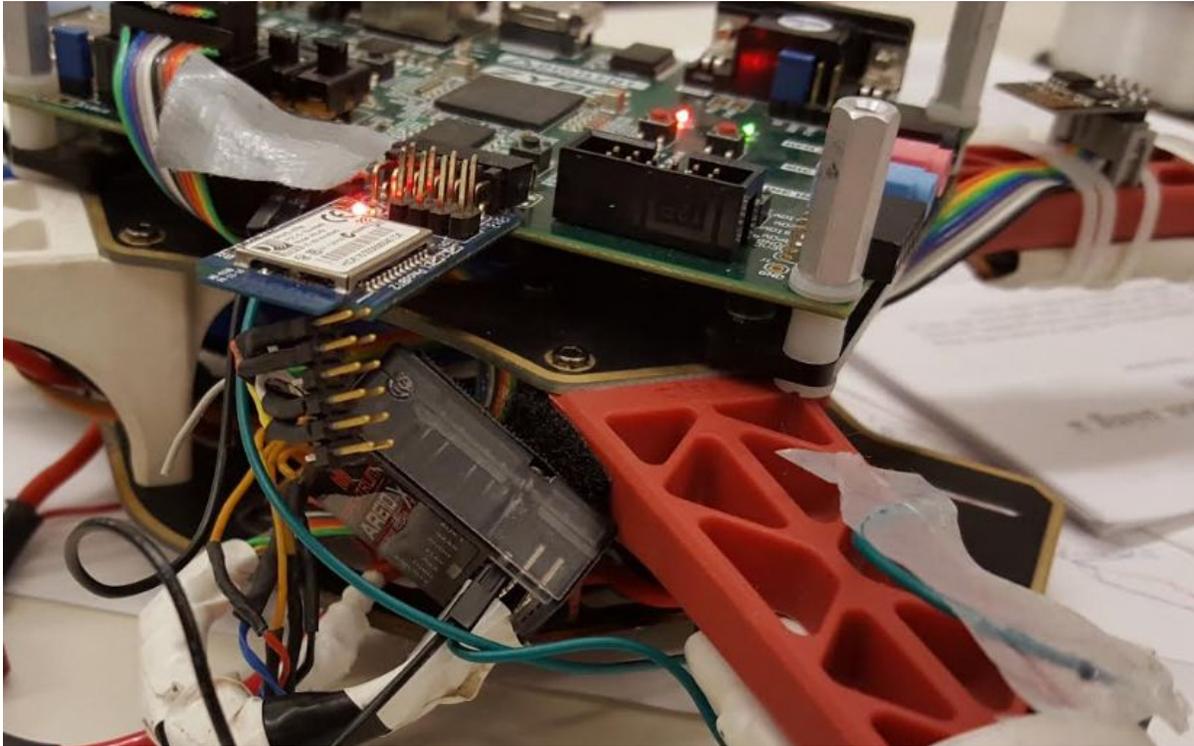
Figure 5: Current Quadcopter Hardware State

Creating our own board will also allow us to add additional quality of life components. We intend to explore improvements to these issues by doing the following:

- Replace all tape-maintained wire connections with secure wire connections
- Replace the plug connecting the battery with ones that are easier to open and close
- Design a PCB to have the following qualities to regulate power on the quadcopter:
    - Regulate the LiPO battery to power the Zybo board:
        - From 11.1 volts to 5 volts
        - At 3 amperes
    - Monitor the current and voltage of the LiPO battery
    - Include a motor cutoff switch
    - Allow for easy access to all of the signal lines

3.4 WiFi Communication

The system is currently configured to use Bluetooth as the communication method between the ground station and the quadcopter. Bluetooth has high latency, which presents difficulties for autonomous flight and running control algorithms on the base station. To improve this, we will:

- Replace Bluetooth with WiFi, using an ESP8266 microcontroller with integrated WiFi.

- Test different configuration parameters to identify the fastest way to send data.
- Write code to implement the communication.
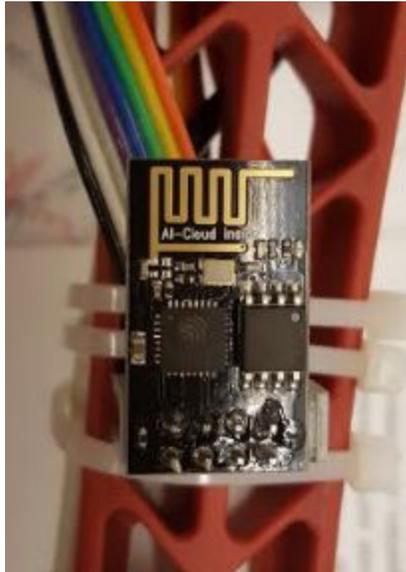
Figure 6: WiFi Module

Our goal is to be able to send data between the ground station and the quadcopter with less than 5ms latency, which would result in 10ms round-trip, a small enough delay to allow the entire control algorithm to execute on the ground station.

3.5 Documentation

In order for subsequent teams to continue the project effectively, critical procedures need to be documented. In terms of developing good tutorials and instructions for future teams, we plan to:

- Write testing procedures for performing measurements for system identification including:
  - Taking moment of inertia measurements about the pitch, roll, and yaw axes
  - Measuring the thrust and drag constants
  - Determining the motor resistance
- Our documentation for test procedures will also include the following:
  - MATLAB scripts used to perform data analysis
  - Pictures of our setup for each test procedure, accompanying a written description and set of instructions for setting up
  - Video tutorials, also will be made available on our website, of our test procedures

3.6 PID Controller Design and Physical Model of the Quadcopter

The quadcopter is stabilized by multiple proportional-integral-derivative (PID) controllers. Historically, the coefficients of the PID controller have been determined through iterative optimization. Since this approach has a couple of downsides, which will be explained in more detail later, our team plans to

characterize the quadcopter with a mathematical model and to then derive the PID coefficients from that model. More specifically, our team intends to:

- Create a mathematical model representation of the quadcopter system, by following the generic quadcopter model described and characterized in [5].
    - Perform system identification - take measurements of quadcopter parameters, including:
        - Moment of inertia about X, Y, and Z axes
        - Thrust and drag constants of quadcopter
        - Motor parameters (i.e. motor winding resistance)
        - ESC parameters: (i.e. "turn-on" percent duty-cycle, maximum duty-cycle)
        - Data communication parameters: (i.e. round-trip latency for camera data)
    - Design our model in Simulink using these parameters and the model in [5]
        - Include 4 main blocks: actuator, sensor, communications, and controller
        - Implement in each block all of the calculations and equations in [5] for a physics model of the quadcopter and how it should fly
        - Characterize the sensor noise and implement this in that block as well
        - Follow code structure, create a data flow diagram, and implement this logic in the controller and sensor block
- Simulate the entire quadcopter system in Simulink from our mathematical model
- Design PID controllers from this mathematical model

4 Design

Below are the design specifications for our approach to the project. We enumerate specific characteristics we need, as specified by our advisor. Additionally, we include the design requirements we believe will best help us meet the end vision for our system by the end of the Spring semester, as described by our advisor.

4.1 System Specifications

Our advisor, Dr. Jones, had some high-level specifications for what he wanted from our project by the end of the year. Besides a few exceptions, these specifications were not very low-level or detailed, so we have come up with some sub-system level specifications as well to help guide our design to meet the specifications of Dr. Jones's high-level vision. The specifications for this project and what he wants for its final version are the following:

- The quadcopter system should be modular, meaning that each component should be easily removed and replaced by another of the same functionality, without breaking the operation of the entire system
- The control system of the quadcopter should be designed from a mathematical model representation of the quadcopter system, rather than from iterative testing procedures as done from previous teams
- The communication latency between the quadcopter and the ground station should be less than 10 milliseconds on average, which will most likely require WiFi communication rather than the current Bluetooth communication system
- The communication system and controls system should work together robustly, such that if any packets happen to be dropped, the quadcopter does not get wildly out of control and potentially damage itself
- Throughout the development process, detailed and user-friendly documentation and tutorials should be created for future MicroCART teams to follow and learn from, including video tutorials as well, and keeping the two-decade-long Wiki page up-to-date

4.2 Non-functional Requirements

4.2.1 Quadcopter Software Non-functional Requirements

The general requirements for the quadcopter software involve autonomous flight, improved communication, integration of external sensors supporting outdoor flight, and running Linux on the second core that can communicate with the main control algorithm.

4.2.1.1 Autonomous Flight Capability Non-functional Requirements

- *The autonomous flight mode should be stable and without oscillations*
- *Autonomous flight will be implemented using PID controllers on the X, Y, and Z axes.*

4.2.1.2 Improved Communication System Non-functional Requirements

- *Sending messages over the guaranteed and non-guaranteed methods should not require excessive burden on the programmer.*
- *Non-guaranteed delivery should be implemented using UDP for low latency*

4.2.1.3 Ability to Support Flight without Ground Station Non-functional Requirements

- *The quadcopter should be able to fly without requiring a ground station or camera tracking system to be initialized.*
- *The quadcopter should store flight logs while not connected to a ground station and be capable of sending these logs to a ground station that connects at a later time*

4.2.1.4 LiDAR Altitude Control Non-functional Requirements

- *LiDAR should provide accurate enough altitude to enable autonomous takeoff and landing for future control systems.*
- *LiDAR system should work whether the quad is inside the camera tracking system or not*

4.2.1.5 GPS Navigation Capability Non-functional Requirements

- *Using a GPS module the quadcopter should be able to navigate positionally outside of the camera tracking system*
- *The position data from the GPS should be able to drop into the quad system and replace the camera system data*

4.2.1.4 Running Linux on Second Core Non-functional Requirements

- *The linux system shall be capable of running high-level, non-realtime applications, such as OpenCV-based camera processing applications*
- *The message passing system to realtime core shall be simple for ECpE students to develop for*

4.2.2 Ground Station Non-functional Requirements

4.2.2.1 Real Time Communication Non-functional Requirements

- *Constant connection with quadcopter*
- *Know all parameters of the quad at all times*
- *Handle graceful failing when connection breaks*

4.2.2.2 Back End Non-functional Requirements

- *Back end should maximize stability and consistency of behavior*
- *Back end should have Properly structured C code and isolated compilation units for encapsulation*
- *Back end should guarantee thread-safety by using encapsulation to prevent unsafe access to protected data*

4.2.1.3 Command Line Interface (CLI) Non-functional Requirements

- *CLI implementation should maximize code reuse between CLI operations*
- *CLI should have common syntax between operations*
- *CLI should run quickly and return human- and machine-usable status codes*

4.2.1.4 Graphical User Interface (GUI) Non-functional Requirements

- *GUI should have good visual appearance*
- *GUI should have an easy-to-use interface*
- *GUI should have Multi-Pane to remove cluttering*
- *GUI should support point and click navigation of the Quad*

4.2.3 Hardware Improvements Non-functional Requirements

4.2.3.1 Power Distribution Board with Integrated Regulator Non-functional Requirements

- *The PCB shall be small enough to fit on the quadcopter without impacting flight performance*
- *All connections between the ESCs and Zybo Board shall be routed through the PCB to allow for custom signal injection*
- *The WiFi module shall be placed on the board in an orientation that does not negatively affect signal strength*
- *All wiring shall be neatly wired to and from the PCB*

4.2.4 WiFi Bridge Non-functional Requirements

From the addition of the WiFi module, the system should have increased flexibility to deliver data, improved management of connectivity, reduced power demand, and increased communication speed.

- *The WiFi module will support both guaranteed and non-guaranteed delivery of data.*
- *The WiFi communication will gracefully handle dropped connections from the ground station, without losing data*
- *The WiFi module should be lightweight and low power*
- *The communication should be fast enough to enable running the control algorithm on the ground station.*

4.2.5 PID Controller Non-functional Requirements

- *The control system shall be implemented with a PID controller*
- *The quadcopter shall be controlled with 9 PID controllers, consisting of the following:*
  - *3 PID controllers controlling the angular position of the quadcopter*
  - *3 PID controllers controlling the angular velocity of the quadcopter*
  - *3 PID controllers controlling the linear position of the quadcopter*
- *Each PID controller shall be implemented as a PID_controller object within the quadcopter software, having the following fields:*
  - *Ki constant, the integral constant of that particular PID controller object*
  - *Kp constant, the proportional constant of that particular PID controller object*

- ○ *Kd constant, the derivative constant of that particular PID controller object*
- ● *Our Simulink model will have the following high-level components*
    - ○ *Communication System*
    - ○ *Control System*
    - ○ *Actuation*
    - ○ *Sensor System*

4.3 Functional Requirements

4.3.1 Quadcopter Software Functional Requirements

4.3.1.1 Autonomous Flight Capability Functional Requirements

- ● *The quadcopter will be able to stay at a stationary point in the Coover 3050 lab autonomously, and deviate less than 1 foot*
- ● *The quadcopter will be able to receive waypoints from the ground station, and autonomously navigate to them, then remain there until further waypoints are given*

4.3.1.2 Improved Communication System Functional Requirements

- ● *The quadcopter will be able to send messages to the ground station with two methods: A guaranteed delivery method, and a non-guaranteed method*
- ● *The non-guaranteed delivery method must be lower latency than the guaranteed method*

4.3.1.3 Ability to Support Flight without Ground Station Functional Requirements

- ● *The quadcopter shall be able to fly in manual mode without a connection to the ground station*

4.3.1.4 LiDAR Altitude Control Functional Requirements

- ● *The quadcopter shall be able to connect to the Sparkfun LiDAR Lite module, take measurements from it, and use those measurements as the altitude information for autonomous flight*
- ● *The LiDAR should provide measurements with precision of at least 3 inches when below 3 feet, and within 1 foot above that*

4.3.1.5 GPS Navigation Capability Functional Requirements

- ● *The quadcopter shall be able to connect to a GPS module, take measurements from it, and use the GPS coordinates as information about the horizontal location of the quadcopter*
- ● *The GPS module will provide location accuracy within 10 feet after obtaining a satellite lock*

4.3.1.4 Running Linux on Second Core Functional Requirements

- ● *The linux operating system shall be capable of communicating with the real-time processor with less than 1ms round-trip latency*
- ● *The linux system shall be capable of connecting to USB wireless cards and routing communication between them and the realtime core*

4.3.2 Ground Station Functional Requirements

4.3.2.1 Real Time Communication Functional Requirements

- *The communication system shall have updated position information about the quadcopter every 500 milliseconds.*
- *The communication system shall pass VRPN data in less than 50 milliseconds, round trip.*
- *The communication will be able to send any and all commands from the user, no matter when the command is made.*

4.3.2.2 Back End Functional Requirements

- Backend socket must accept simultaneous connections from many clients
- Backend daemon must handle connection failure
- Backend socket must handle data errors from quad and provide only valid data to clients
- Backend must reject invalid or unsafe user input

4.3.2.3 Command Line Interface (CLI) Functional Requirements

- Clients must validate user input before forwarding to backend
- Clients must indicate error conditions to user
- Clients shall allow waypoints to be sent to the quad
- Clients shall allow individual-access setpoints to be set, displayed, and unset
- Clients shall allow logs to be retrieved from the quad

4.3.2.4 Graphical User Interface (GUI) Functional Requirements

- *The GUI shall constantly provide a visual representation of Quad*
- *The GUI shall map points and allow for click navigation.*
- *Status and Error notifications should be present*
- *Graceful connection and disconnection with the Quad*
- *All communication to backend done in one process*

4.3.3 Hardware Improvements Functional Requirements

4.3.3.1Power Distribution Board with Integrated Regulator Functional Requirements

- *The voltage regulator will output 5v +/- 2%*
- *The voltage regulator will provide up to 3A continuous and stay within voltage spec*
- *The current monitor will measure up to 30A at 12-bit accuracy*
- *The voltage monitor will measure up to 13v at 12-bit accuracy*
- *The solid state power switch will be capable of handling up to 30A continuous current*

4.3.4 WiFi Bridge Functional Requirements

Overall, the requirements for the WiFi module are to be able to send and receive relatively large quantities of data with low latencies. The functional requirements below quantify this.

- *The ground station will be able to send 32 bytes to the quadcopter in less than 5ms, and the quadcopter will be able to send 32 bytes to the ground station in less than 5ms.*
- *The quadcopter and ground station will be able to communicate for up to at least 100 ft without obstruction.*
- *The WiFi module will be able to send data over a protocol that guarantees delivery.*

4.3.5 PID Controllers Functional Requirements

In general, the PID controller should provide stability to the quadcopter under various flight conditions.

- *The quadcopter, while in autonomous mode and while in the Lab, shall be able to hover and remain at a fixed point.*
- *The quadcopter, while in autonomous mode and while in the Lab, shall be able to travel from a point in space to another point in space autonomously, when given a position input from the user in the GUI interface.*
- *The quadcopter, while in autonomous mode and while outside, shall be able to hover and remain at a fixed point.*

4.4 Proposed Design Method

4.4.1 Quadcopter Model Design

The overall system is composed of four main components including the communication system, control system, actuation, and sensor system as shown below in **Figure 7**.
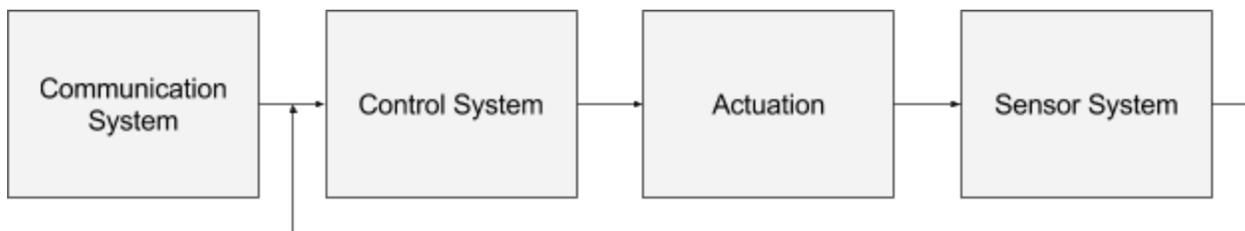
Figure 7: High-Level System Block Diagram

4.4.1.1 Communication System

The input to the control system from the communication system is dependant upon indoor or outdoor flight. When flying in the distributed autonomous and networked control lab, the x, y, z position of the quadcopter in space is determined from an OptiTrack camera system. This information must then be passed to the quadcopter through the ground station. However, during outside flight, the x, y position will be determined from an onboard GPS module, and the z position will be determined with a LiDAR solution. This GPS module and LiDAR solution will be a part of the sensor subsystem described in Section 3.2.4. Alongside this, the communication system will also provide any user input from the command line interface (CLI), or controller during manual flight. This overall communication process is represented below in **Figure 8**.
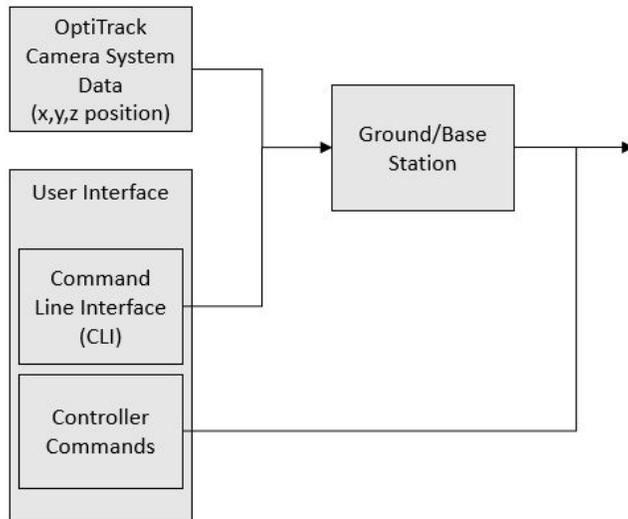
Figure 8: Communication System Block Diagram

4.4.1.2 Control System

As described in Section 4.4.1 and Section 4.4.4, the control system inputs come from both the communication system and sensor system. The entire control system is composed of multiple nested PID controllers. The composition of a PID controller in a feedback loop is shown below in **Figure 5**, where r(t) is the set point value, and y(t) is the measured output. The primary components of the PID controller include the K_p , K_i , and K_d terms, which denote the coefficients for proportional, integral, and derivative terms, respectively. These coefficients will be determined using a mathematical model of the quadcopter as described in the introduction of Section 3 and in Section 2.7.
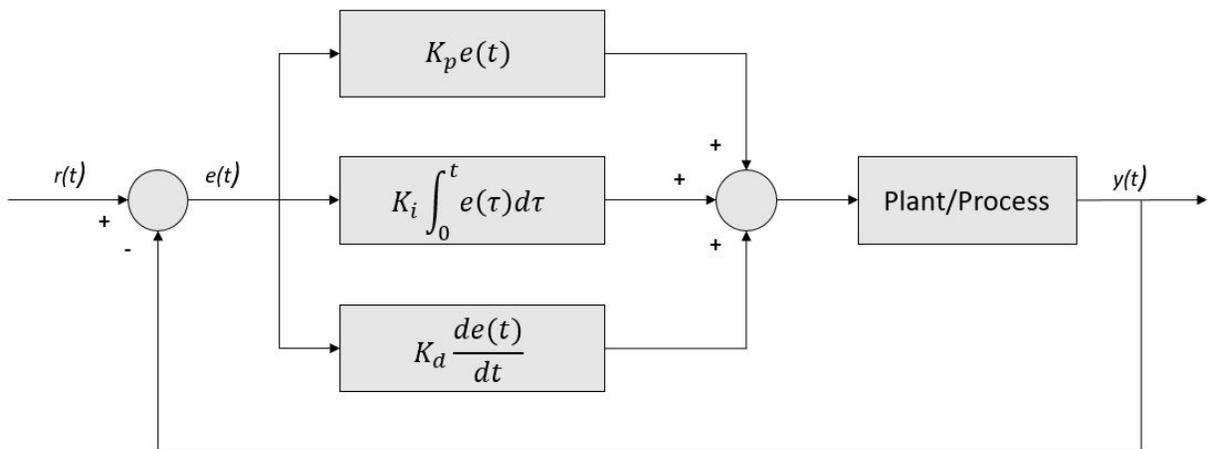
Figure 9: PID Controller Block Diagram

The overall control system is broken down into four major subsections including the height controller (z-axis), longitudinal controller (y-axis), lateral controller (x-axis), and a dedicated yaw controller. Most of these controllers have nested controllers associated with them, as shown below in **Figure 10**.
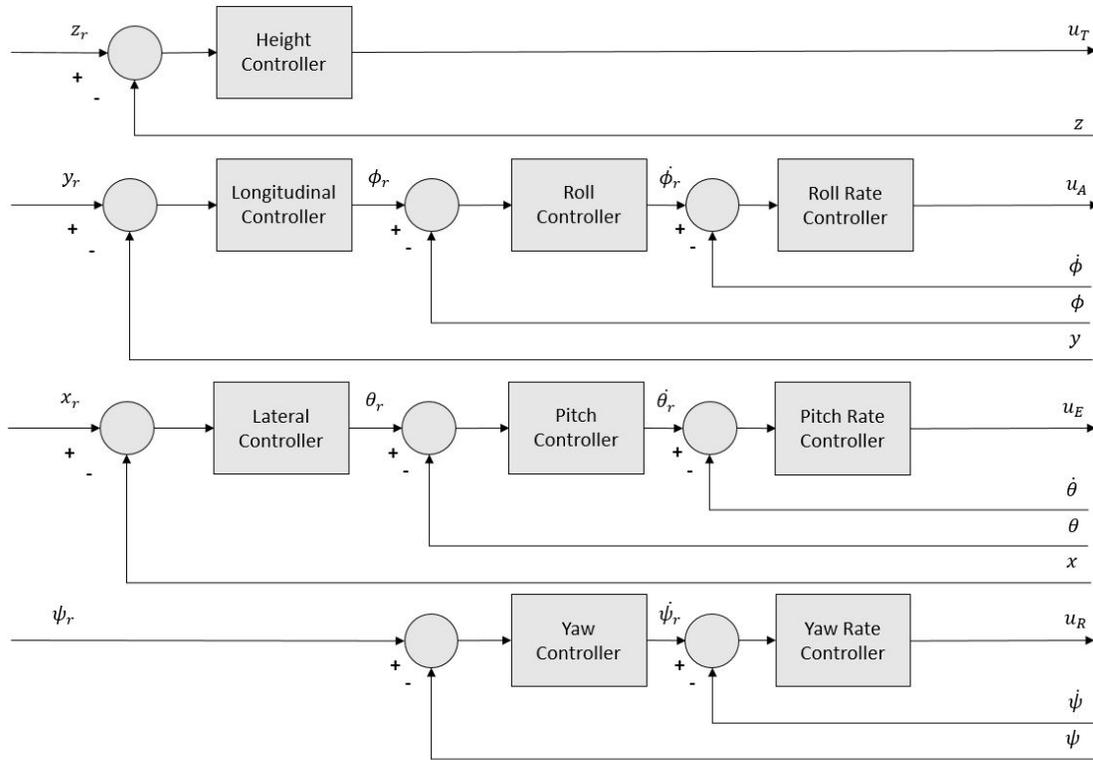
Figure 10: Nested Loop PID Architecture

This architecture allows us to not only control the body frame position of the quad, but it's velocity as well. When describing things in terms of body frame we are referring to the frame of reference of the body of the quadcopter. This assumes that the origin of this axis lies at the center of mass of the body. The variables utilized in Figure 6 are defined in **Table 1** below. Note that in the above image variables denoted with the subscript "r" represent setpoint values.

Variable	Definition
x	Body frame x position
y	Body frame y position
z	Body frame z position
ϕ	Body frame roll angle
θ	Body frame pitch angle

ψ	Body frame yaw angle
$\dot{\phi}$	Body frame roll angular velocity
$\dot{\theta}$	Body frame pitch angular velocity
$\dot{\psi}$	Body frame yaw angular velocity
u_T	Throttle command
u_A	Aileron command
u_E	Elevator command
u_R	Rudder command

Table 2: PID Architecture Variable Definitions

The last portion of the control system is converting the actual output commands of the controllers to equivalent input commands for each of the four individual electronic speed controllers (ESCs). To do this we utilize a signal mixer, defined by the following matrix:

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

This addition is shown in **Figure 7** below:
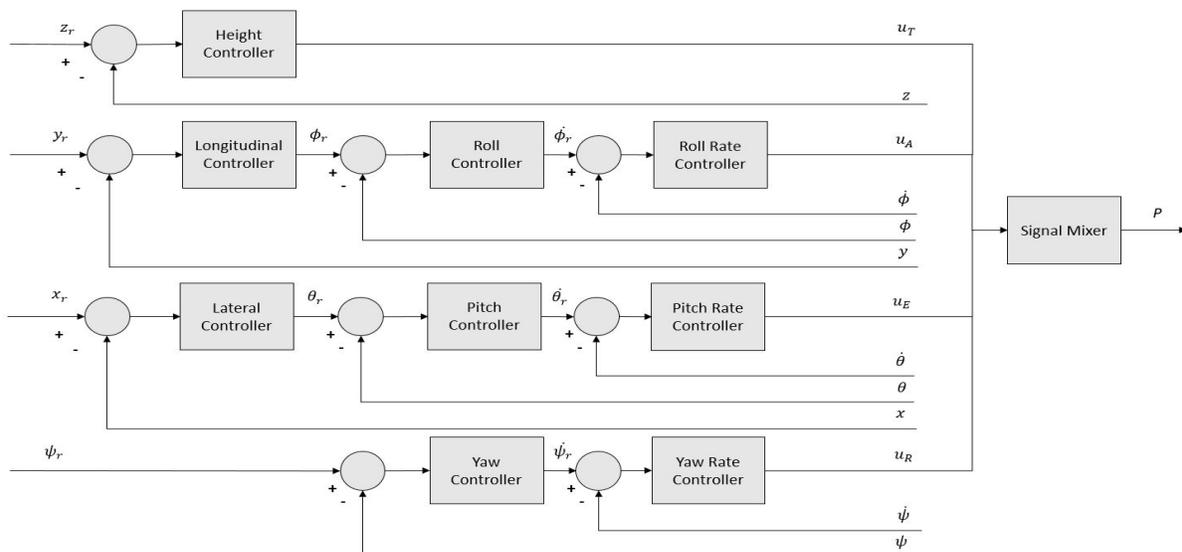
Figure 11: Nested Loop PID Architecture with Signal Mixer

Note that P in the above system diagram represents the vector of all four ESC duty cycle percentages.

4.4.1.3 Actuation

The actual actuation, or mechanical movement of the quad occurs through the driving of each motor/rotor combination. As stated previously the output from the control system provides ESC duty cycle percentages, these percentages are used by the ESC to coordinate what voltage to apply to each motor, represented as the vector V in **Figure 8**. The ESCs themselves are powered from a 11.1V LiPO battery (nominal voltage). From there the we can determine the actual angular velocity and acceleration defined below:

$$M = \begin{bmatrix} \omega \\ \alpha \end{bmatrix}$$

where ω and α represent the angular velocity and acceleration respectively. From this we are able to derive the overall block diagram for the actuation of the quad.
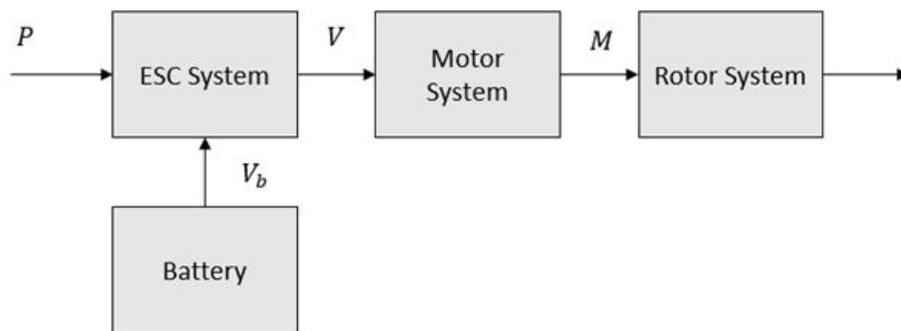
Figure 12: Actuation System Block Diagram

4.4.1.4 Sensor System

The sensor system is composed of the MPU-9150 IMU (Inertial Measurement Unit) which provides gyroscope and accelerometer data and is also an input to the control system. Alongside this, during outside flight the OptiTrack camera system will be replaced with a dedicated GPS module and LiDAR solution for determining the x, y, z position of the quadcopter. Data from the gyroscope and accelerometer can be used to find the yaw, pitch, roll angles, and angular velocities. With this and either the OptiTrack camera system or GPS module and LIDAR solution, we are able to provide all the required inputs to the control system.
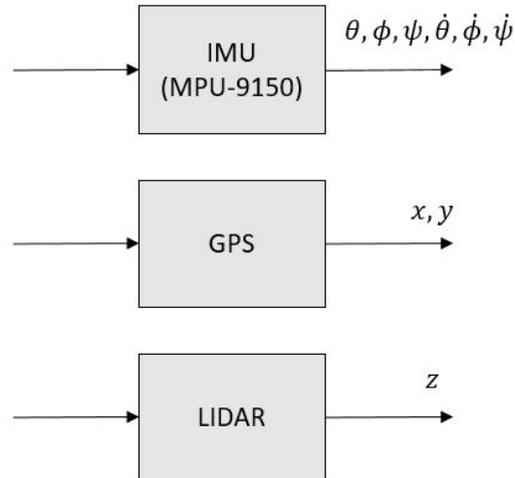
Figure 13: Sensor System Block Diagram

We have recently been looking into representing these systems in our Simulink model of our quadcopter in the best possible way. Currently our software code that takes the readings from the IMU to determine the pitch and roll angle of the position of the quadcopter is based on the a few equations. And though this makes intuitive sense, since we were able to rederive these equations using basic trigonometry, our graduate student advisors, Matt and Ian, have been explaining to us that some researchers may have developed a better way to represent these angles of the quadcopter [2]. We will look into this to see if this is indeed a better method that will be worth switching to depending on how much more accurate it is as well as how much more effort it requires and complication it adds to the project.

4.4.2 Ground Station Design

4.4.2.1 Backend
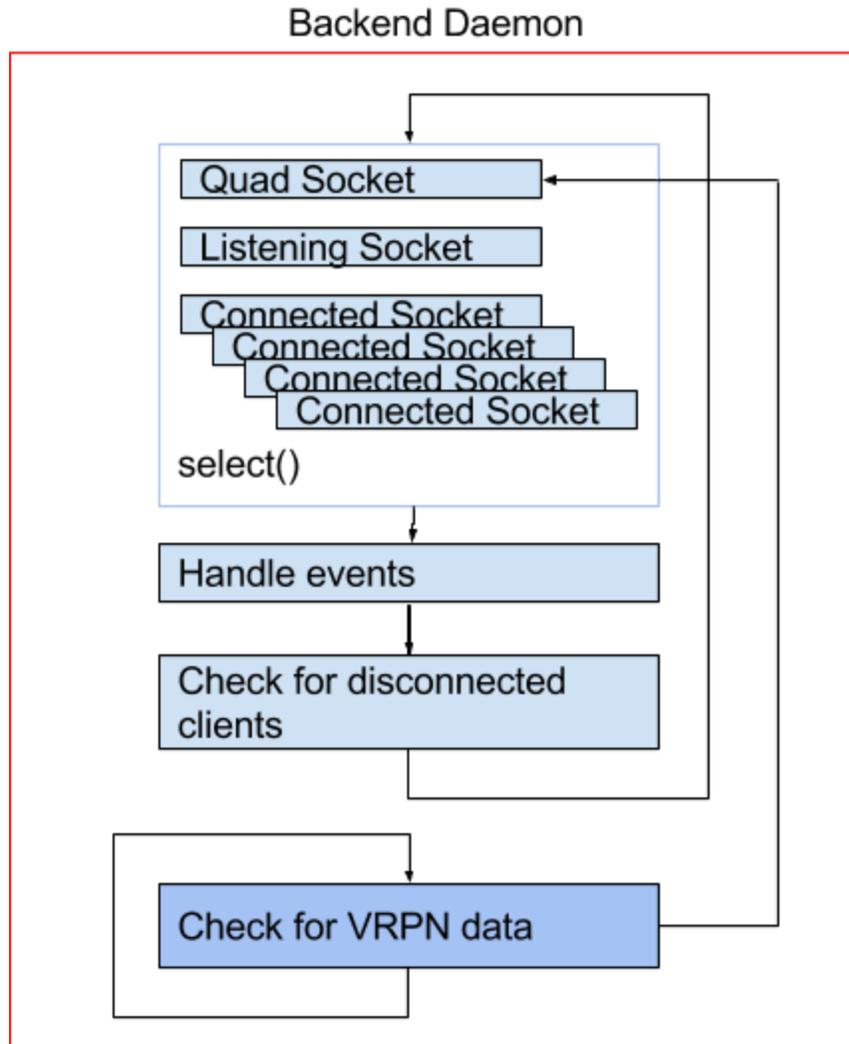
Figure 14: Backend Daemon

The backend daemon runs and manages connections with the quad. Communication with the quad uses a low-overhead binary protocol. It also connects to the tracking system and forwards tracking information to the quad. Frontends, CLI or GUI, connect to the backend and issue commands in an easy-to-parse ASCII protocol. These commands manipulate the state of the quad.

4.4.2.2 Command Line Interface (CLI)
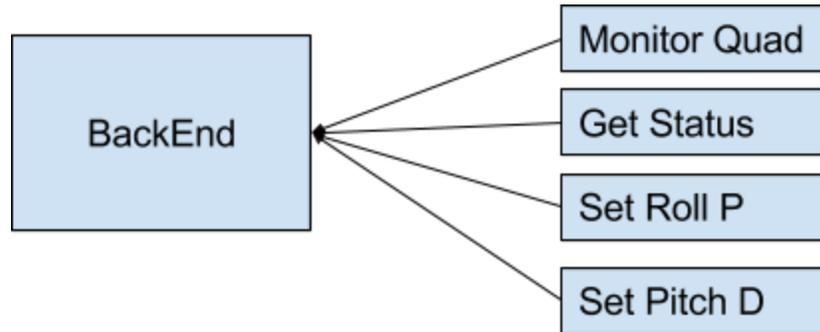
Figure 15: Command Line Interface Block Diagram

The design of the CLI is to create a Client-Server relationship with the BackEnd. The CLI will be split up into separate programs. These programs will either create a connection and perform the task, or create a connection and hold it until the user is done with the task. This will allow tab completion in a shell, bash scripting to allow test scripts as well as a history of commands with BASH_HISTORY.

4.4.3 Quadcopter Software Design

The quadcopter software from last year's team is already modular, so we will not be making large changes to their software. We will have to add functionality for two major components: GPS and LiDAR.

In order to add LiDAR we will add a LiDAR module into the existing quadcopter hardware. This LiDAR module will optionally replace the Z position measurement that is currently coming from the camera system. Adding this LiDAR will be necessary for the quad to be able to fly outside of the camera system.

Another addition to the quadcopter hardware is a GPS module. Adding this new hardware will allow the X, Y position of the quad to be tracked while the quad is outside of the camera tracking zone.

The combination of the LiDAR and GPS systems will allow tracking of the X, Y, Z positions, replacing the camera system functionality. We will need alter the quad software sensor data collection and data structures to allow use of the new hardware.

We will also update the code that performs communication. To do this, we will pull the message sending out into a function. Currently, when a message is to be sent, a function formats the packet, but we manually call write() over the whole packet. This makes it difficult to manage memory, and resulted in last year's code having memory leaks when sending/receiving messages.

4.4.4 WiFi Bridge Design

The WiFi communication will be implemented with the ESP8266 microcontroller with integrated WiFi. The WiFi module will act as an access point, so the ground station can connect to the WiFi network

hosted on the module without any extra configuration. To program the ESP8266, we will use features provided by the Arduino library for the ESP8266, as well as functionality provided by the Espressif SDK. The module will support both TCP and UDP for communication. To simplify the integration of WiFi, it will forward data from UART to WiFI, and forward the data received over WiFi to UART. This way, no changes will be required in the quadcopter software to use WiFi as the communication method.

4.5 Design Analysis

4.5.1 Quadcopter Model

The quadcopter model method of designing our PID controllers is different from how the past two teams have approached the control system design problem. In the past two years the controllers have been designed by hand-tuning the PID controllers, or in other words, guessing and checking the PID constants for each of the controllers. Although this method is usually still a good way of getting adequate performance for the purposes of a given application, for the quadcopter system, this method will not work very well, because a quadcopter is inherently an unstable system.

Thus, this year, our design method for the PID controllers will be done in a more systematic, more rigorous manner. So we will be trying to create a mathematical model of the quadcopter and design the control system using this representation of our quadcopter. We are quite certain this will be a better method than in the past two years, because we have seen from Matt Rich, who is a PhD student working closely with the controls team who helped develop a mathematical model for MicroCART almost 5 years ago, that using this model when doing a PID controller design can give you a robust system for traveling between waypoints.

Because this has been accomplished before by Matt's senior design team, we will try to apply the same method for the PID controller design of our quadcopter. And indeed, we have been following Matt's proposed method of designing the PID controllers from a mathematical representation of the quadcopter, by following these methods proposed in his thesis. And so far, after doing preliminary testing, which has its results included in the last section entitled "Results", we have found that our mathematical representation of the quadcopter system, such as its motors, is proving to be quite similar to how the actual system is behaving.

4.5.2 Ground Station

The ground station will be separated into two roles: backend and frontend. This will increase stability and allow for more consistent state of the ground station as it runs.

The backend will consist of a headless daemon running at all times. This daemon will manage connection lifecycle with the quad and tracking system, as well as forward tracker data to the quad. The backend will control whether or not the quad is active, and the control authority (autonomous/manual) of the quad. The daemon will provide a Unix-domain socket for clients (front-ends) to connect and issue commands.

Front-ends may exist in several forms. The first front-end will be a simple non-interactive command line control program. This program, supporting several subcommands, will be able to manipulate the state of the quad (active, manual, automatic, set coordinates, command waypoints), retrieve state, and download logs. This command will run, issue the command to the backend, and return any information before exiting. The source code for this utility will be structured to lend itself to reuse in later front-end implementations: the connection to the backend as well as the actual functionality to issue commands will be isolated from the argument parsing logic.

Later front-ends may be implemented as interactive GUI (or possibly CUI) applications. The backend daemon will allow one or many front-ends to operate simultaneously, and reusable command and communication code will make the creation of further front-ends a relatively trivial matter of acquiring user input, then calling the proper function to perform the desired operation. UX development work can be completely isolated from the business logic of controlling the quad.

4.5.3 Quadcopter Software

The design of the quadcopter software was completed by previous teams. Their design is very modular, so we do not expect to change the design very much. Our plan to move message sending into a function of its own will eliminate unnecessary malloc() and free() calls, improving performance, and decreasing the likelihood of memory leaks.

Because of the modularity of last year's system, it will not require substantial changes to add GPS and LiDAR to the system. Sensor reading and sensor filtering are in separate files. We will add the LiDAR and GPS to the sensor reading, and add a field to the sensor data structure to pass the sensor data to the sensor processing file. Within the sensor processing file, we will add a function that is able to convert GPS data and LiDAR data into X, Y, Z data that is used elsewhere in the system. Because the sensor processing function allows us to convert data, it will not require changes elsewhere in the code to integrate GPS and LiDAR.

4.5.4 WiFi Bridge

We believe that the design for the WiFi bridge should work well. Because the ESP8266 chip includes a microcontroller, we can add logic to automatically reconnect, buffer writes that failed, etc. This allows us to make a more robust system, especially when it is used outdoors.

5 Testing and Development

5.1 Interface Specifications

5.1.1 Ground Station Interface Specifications

The ground station software has two main components:

- Backend Component
- Frontend Component

The backend communicates directly with the VRPN system and the quadcopter. It also allows frontend clients to connect it to it via a socket interface, which allow the clients to call functionality available in the backend, and the backend to send updates to the frontend.

The frontend can be a simple command-line interface, or a more sophisticated graphical user interface.

5.1.2 Graphical User Interface (GUI) Specifications

The GUI will have separate tabs to facilitate several modes of operation. It will have a tab for managing waypoints in full-autonomous mode, as well as separate tabs for different semi-autonomous modes. This includes a tab for setting fixed altitude, or fixing other coordinates. Additional tabs will have functionality to monitor connection status and download log data.

5.1.3 WiFi Bridge hardware Interface

The WiFi bridge will communicate with the Zybo board via UART. Each byte received over WiFi will be sent out on UART, and each byte received on UART will be sent out over WiFi.

5.2 Software Specifications

Here we will indicate any software specifications used during the testing phase. We should include brief, simple introductions for each sub-section to explain the purpose of each component within the overlying "Software" category

5.2.1 PID Controller

The current tests we have performed with regards to the PID controllers are primarily error calculations from the mathematical model of the quadcopter. To obtain accurate PID constants we need to model the quadcopter as best as we can to simulate a real scenario. One portion of this model involves the relationship between the electronic speed controllers (ESCs) and their relationship to the rotor speed. This relationship is as follows:

$$\omega_i = \frac{-1 + \sqrt{1 - 4R_m K_v K_q K_d (K_v R_m i_f - K_v u_{p_i} V_b)}}{2R_m K_v K_q K_d}$$

(5.1)

Variable	Definition
ω_i	Rotor Speed
R_m	Internal motor resistance
K_q	Motor torque constant
K_v	Motor back-emf constant
i_f	Motor internal friction current
μ_{pi}	ESC input duty cycle ratio
V_b	Nominal battery voltage

Table 3: Rotor Speed Equation Variable Definitions

For our testing procedure, we need to take experimental data of the angular speed of each rotor and then use the above equation to find our estimated rotor speed based on ESC input duty cycle percentage. With these data, we represent a relationship with our experimental data of the motor speed with respect to the percent duty cycle of the input PWM signal, and we can compare this with our predicted motor speed using the expression derived above. This test will be done for each of the 4 motors.

Additionally, we will do testing in Simulink of our quadcopter system on each of the four blocks to determine it has been simulated properly. These tests will include simple motions including the following, along with the corresponding expectations we should see in Simulink:

Test 1 for Simulink Model: Turning all quadcopter motors at the approximate hovering percent duty cycle - the percent duty cycle that directly counters gravity's downward force
- Should be approximately hovering:
- X, Y, and Z position should all be constants
- X, Y, and Z velocities should be zero
- Roll, pitch, and yaw angles and angle rates should all be zero

Test 2 for Simulink Model: Turn quadcopter motors on maximum percent duty cycle per motor
- Should accelerate upwards
- Z position should be a negative parabolic function (Z is positive in the downward direction)
- X and Y positions and velocities should be at zero
- Z velocity should be a negatively-sloped line (Z is positive going down)
- Roll, pitch, and yaw angles and angle rates should all be zero

Test 3 for Simulink Model: Turning motors to right at a slightly smaller percent duty cycle than those to left - while setting the gravity vector to zero for simpler analysis

- ○ Should roll in positive direction, flipping about its center, while going to the right
- ○ X positions and velocities should all be zero
- ○ Y should be a positively-biased sine function that diminishes in amplitude (starts at zero, goes positive when quadcopter circles right, then goes back to zero as the quadcopter circles around)
- ○ Z should be a negative sine function (starts at zero, goes negative when quadcopter goes up, back to zero, goes positive when quadcopter goes down)
- ○ Pitch and yaw angles and angle rates should be zero
- ○ Roll angle should be a positive and increasing at a parabolic rate
- ○ Roll angle rate should be a positively-sloped line

Test 4 for Simulink Model: Repeat Test 3 but set the motors to the left at a slightly smaller percent duty cycle than those to the right - should get the same results, except:
- ○ Y position and velocity should be negative of what they were in Test 3
- ○ Roll angle and angle rate should be negative if what they were in Test 3

Test 5 for Simulink Model: Repeat Test 3 but set the motors to the back at a slightly smaller percent duty cycle than those to the back - should similar results, but it will be analogous for the pitch rotation
- ○ Should pitch in positive direction, flipping about its center, while going to the backward
- ○ Y positions and velocities should all be zero
- ○ X should be a negatively-biased negative sine function that diminishes in amplitude (starts at zero, goes negative when quadcopter circles back, then goes back to zero as the quadcopter circles around)
- ○ Z should be a negative sine function (starts at zero, goes negative when quadcopter goes up, back to zero, goes positive when quadcopter goes down)
- ○ Roll and yaw angles and angle rates should be zero
- ○ Pitch angle should be a positive and increasing at a parabolic rate
- ○ Pitch angle rate should be a positively-sloped line

Test 6 for Simulink Model: Repeat Test 5 but set the motors to the top at a slightly smaller percent duty cycle than those to the bottom - should get the same results, except:
- ○ X position and velocity should be positive, not negative, of what they were in Test 5
- ○ Pitch angle and angle rate should be negative if what they were in Test 5

Additional Tests: Try different combinations to predict what the quadcopter will do and if it makes sense intuitively. Not only will this be good as it is additional testing for the quadcopter model system, but it also helps the tester gain a better understanding for how the quadcopter system is supposed to work

5.2.2 WiFi Bridge

The testing of the WiFi bridge should separately test the latency introduced by the network protocol (TCP, UDP) from the latency added by serial communication. This is important to distinguish, so we can compare latencies across different platforms.

The UART interface for the WiFi bridge can be tested by directly connecting the RX/TX lines on the WiFi bridge, and connecting to the WiFi module with a simple terminal(PuTTY on Windows, telnet on Linux). Every command sent over the terminal should be echoed back. This is a simple test to verify that we can send large and small amounts of data without loss.

For more robust testing, we have created a test script in Python that will send arbitrary sized packets, with different delays. It verifies the integrity of the packet, and records if any data was corrupted or lost.

5.3 Hardware Specifications

5.3.1 Power Distribution Board

The initial testing of the Power Distribution Board will be done in software simulation. Many companies that provide switch-mode power supply controllers also provide free software simulation tools, so our first simulation will be using those tools. Then, after our schematic has been fully implemented in Multisim, we will use the simulation functionality in that program to verify our design before we layout the PCB. Now, we will be confident enough in our design to complete the board design and have it manufactured.

The final testing of the voltage regulator will be after the board has been manufactured. We will start by assembling only the voltage regulation portion of the board and performing specific tests on this subsystem. First, we'll verify that the output voltage is within our required specifications. Then, we will increase the load on the system using high-power resistors until our maximum current rating is reached. At that point, we will verify that the output is still within the specifications.

After the voltage regulator has been fully tested, the last system to test are the connections for the ESCs and the WiFi module. For this, we will simply perform continuity tests between all connections to make sure that they are properly connected. Then, we will connect all of the devices and verify that all signals are being properly routed without being distorted. If all signals are correct, the board is functional.

6 Test Results

Here we will list and explain any and all results obtained during testing we have done so far. We should include both the results from our failed tests and our successful tests. After presenting these results, we should elaborate on what we learned from our results and how we will use our results to change and optimize our "plan of attack" as we progress with the project.

6.1 WiFi Bridge

We currently have implemented and tested the TCP portion of the WiFi bridge. We have run the test described in 5.2.2 with various packet sizes and delays, to verify that the WiFi bridge is able to sustain the data rates that we will be sending. During regular flight, we will be sending roughly 32 bytes every 10ms. To verify that we are able to reliably send sufficient data, we have simulated 30 minutes of flight time, sending 350 bytes every 10ms, without losing any data.

We have done timings for round-trip between ground station and quadcopter response. Figure x shows the probability that a message will have a round-trip latency lower than a particular time.
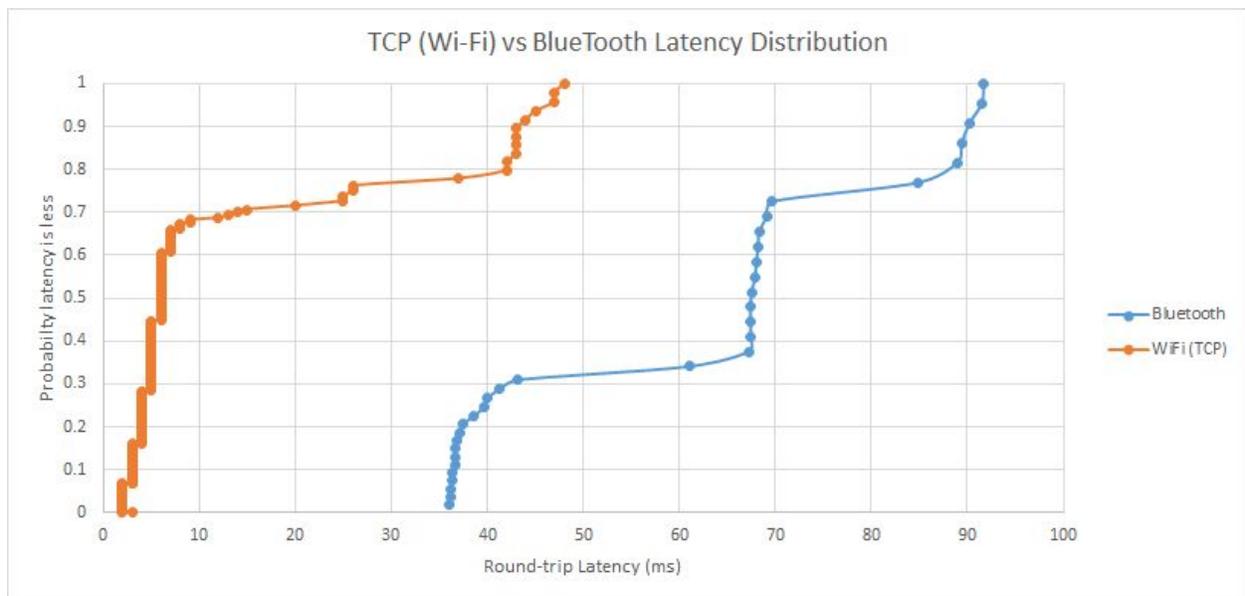

Figure 16: TCP vs BlueTooth Latency Distribution

We have also done some preliminary tests with UDP, and expect the median latency to decrease from ~7ms to ~4ms.

6.2 Quadcopter Model

By taking experimental data of the angular speed of each rotor and then using equation 5.1 to find our estimated rotor speed based on ESC input duty cycle percentage, we were able to produce the following graph:
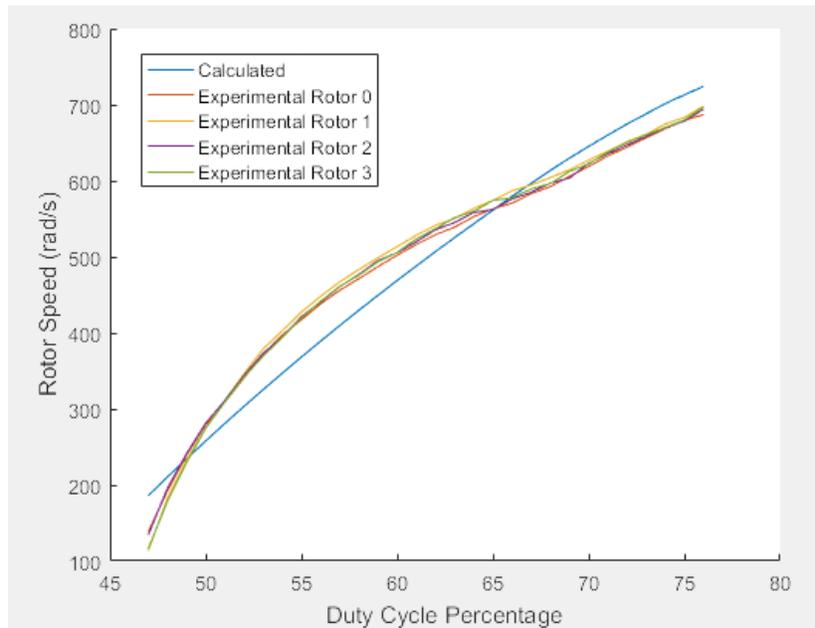
Figure 17: Calculated and Experimental Duty Cycle vs Rotor Speed

In the above graph the blue line represents our calculated rotor speed based off of equation 5.1, and the other lines represent the experimentally determined rotor speeds.

From this we can calculate the error associated with our calculation:
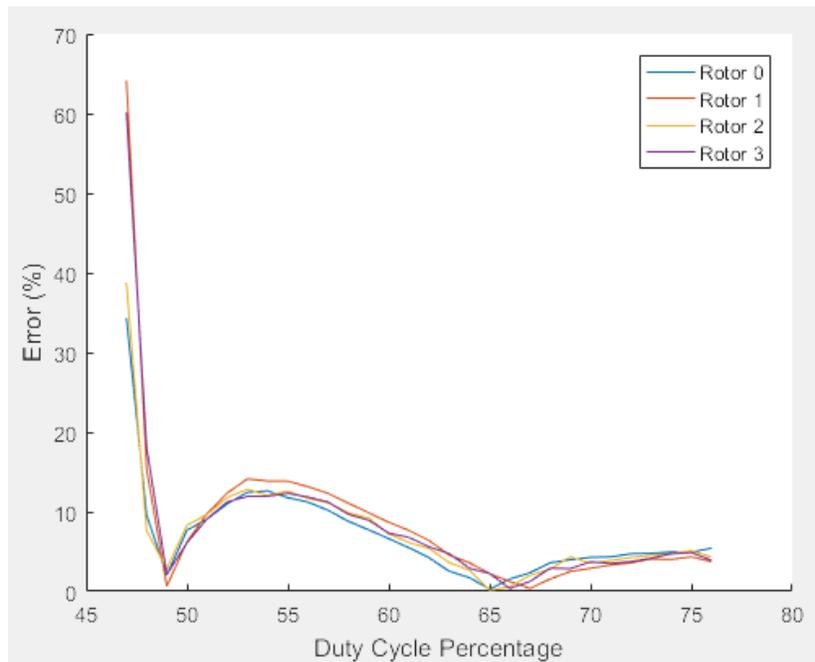
Figure 18: Error Between Predicted and Experimental Rotor Speed Data

From this we note that at low duty cycle percentages we have a very higher error, however, this is outside of our normal operating range. The typical error we would see within our operating range is between 5% and 15%, which we will neglect until further experimentation proves it to be an issue.

Additionally, we tested our Simulink model with the tests mentioned above in section 5, and tried applying an upward thrust with equal percent duty cycles for each motor. We expected to see the quadcopter going straight upward, which would be in the negative z-direction since z is positive downward. Then here were the graphs we got as a result:
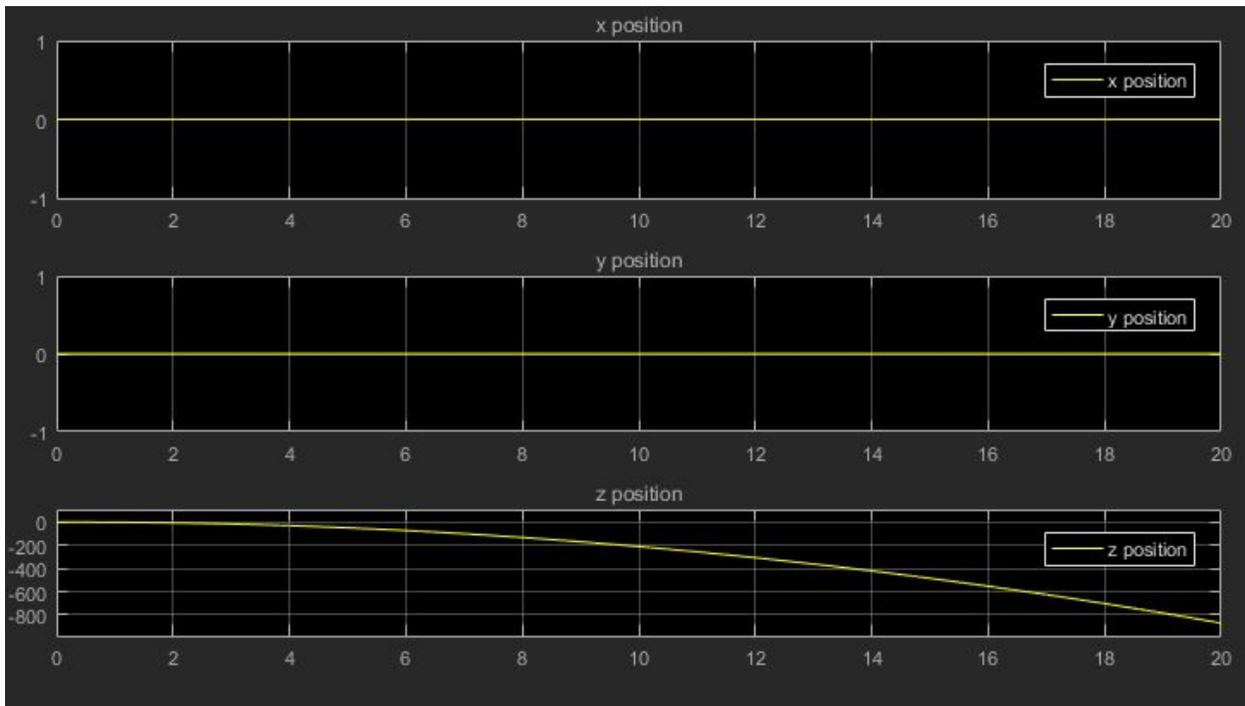

*Figure 19: X, Y, Z Position Calculated by Simulink Model*
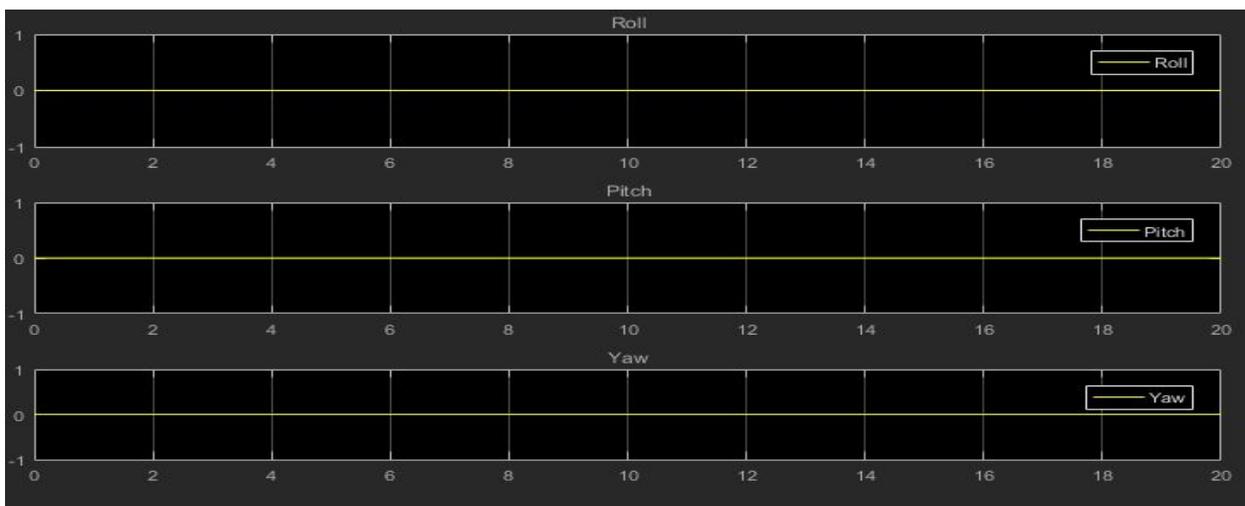
*Figure 20: Roll, Pitch, Yaw Angles Calculated by Simulink Model*
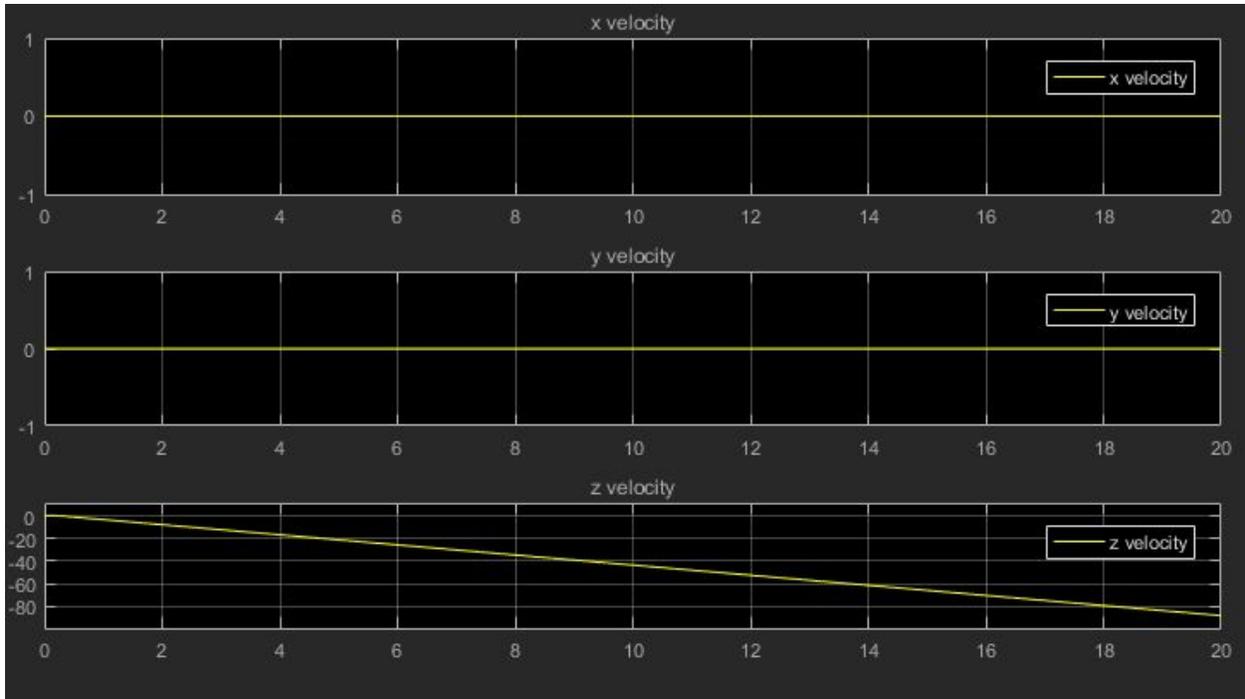
Figure 21: X, Y, Z Velocities Calculated by Simulink Model

As you can see, these calculate results make sense for the linear and angular position and velocities for our Simulink model. The X, Y, Z position make sense, because the quadcopter model should be going directly upward, which would make the X and Y position zero and the Z position a negative parabolic function with respect to time, since it should be accelerating upwards for there is a constant net force upward, and the positive z-position is pointed downward. Thus, this first graph makes sense.

Additionally, because the quadcopter is not tilting or rotating in any direction, it makes sense that the roll, pitch, and yaw angles are all zero, and that their angular velocities are also zero. Thus, this aspect of our Simulink model also makes sense. Finally, the X, Y, and Z velocities also make sense, since the quadcopter should not be moving in any direction in the X and Y directions, thus these velocity values should be zero, however it is accelerating upward, thus the Z-directed velocity should be increasing in magnitude, but negative in sign, since the Z axis is positive in the downward direction. Thus, these results make sense for our predictions and expectations for the behavior of our Simulink model. We ran many tests like this on the Simulink physics model, as enumerated and described in detail in the previous section of this document, and examine the results in Simulink in the process of verifying the accuracy of our physics model for the quadcopter system.

6.3 Ground Station Tests

To verify that the backend can receive socket connections from a frontend, we used a simple UNIX-local socket terminal to act as a command line interface. This communication between the frontend and backend can be seen in the screenshots below.

Figure 22: Simple Unix Socket Terminal Acting as CLI

Figure 23: Backend Running with Frontend Connected

7 Conclusions

Summarize the work we have done so far. Here, we will briefly re-iterate our goals. Then, re-iterate the plan of action we have determined is best for achieving our goals. We should also briefly indicate why this plan surpasses all other possible solutions, based on our results from testing.

So far, we have tested the communication module with WiFi, and believe this is a promising route for our communication method between the quadcopter and the ground station, since we have less latency than we did with Bluetooth. However, we need to continue to work on this module to decrease the latency still. We have tested our representation for the motor speed with respect to the input duty cycle, as well, and have found that this error is reasonable for our PID controller, thus we believe that the mathematical model is indeed a good way of representing the quadcopter and building a PID controller for.

This year (the 2016 to 2017 academic year), MicroCART will improve the current system by increasing modularity and designing PID controllers to manage the movement of the quadcopter in each axis of rotation and linear motion. Eventually, starting next semester, we will begin the process of enabling the quadcopter to fly outdoors using GPS navigation, along with a LIDAR sensor attached underneath to detect its height above ground or obstacles located beneath it, which will be important for achieving successful landings. Finally, after accomplishing these main goals listed, we would potentially like to enable the quadcopter to perform advanced flying maneuvers, as well.

We believe this is the best plan of attack, because of our preliminary results obtained and shown in this document in the "Results" section. Also, from the results obtained from MicroCART team of five years ago, which was the first team that followed Matt Rich's thesis, which discusses a good mathematical model of the quadcopter that is not overly rigorous, and this is the current thesis we are following to develop a mathematical model of the quadcopter system and design our PID controllers from it. From these indications, as well as the experience of our advisors, and their suggestions for being able to make our goals feasible ones, and the fact that our method has currently been helping us meet the details of their suggestions for our system, this also is indicative that our current method will be a better plan of attack than that of the past two years.

8 References

[1] *Bluetooth vs Wi-Fi. (n.d.). Retrieved November 19, 2016*, from

http://www.diffen.com/difference/Bluetooth_vs_Wifi

[2] Cavallo, A., A. Cirillo, P. Cirillo, G. De Maria, P. Falco, C. Natale, and S. Pirozzi. *Experimental Comparison of Sensor Fusion Algorithms for Attitude Estimation*. Thesis. Second University of Nepales, 2014. Aversa: ScienceDirect, 2016. Print.

[3] Ogata, Katsuhiko. *Modern Control Engineering*. 5th ed. Englewood Cliffs, NJ: Prentice-Hall, 1970. Print.

[4] "Products." *DJI Store*. DJI, 2016. Web. 12 Oct. 2016. <http://store.dji.com/>.

[5] Rich, Matthew. *Model Development, System Identification, and Control of a Quadcopter Helicopter*.

Thesis. Iowa State University, 2012. Ames: Graduate Theses and Dissertations, 2012. Web.

[6] *Zynq-7000 All Programmable SoC Overview*. DS190 (v1.10). Xilinx. September 27, 2016