Microprocessor-Controlled Aerial Research Team Final Report

Group May15-28

Joe Boldrey, Adam Campbell, Paul Gerver, Tyler Kurtz,
Ravi Nagaraju, Matt Post, Jacob Rigdon, Matt Vitale

04/28/2015

Contents

List of Figures

List of Tables

1 Definition of Terms

I^2C - A two-wire serial bus protocol invented by Philips
RC Transmitter - Device that controls thrust, roll, pitch, and yaw through positional pulse modulation
microSD - A small SD card used to hold the quad's hardware configuration and software program
JTAG - An interface for FPGA devices to be programmed and debugged

2 Abbreviations

ADC:	Analog-to-digital converter
BT:	Bluetooth
CAD:	Computer-aided design
DSP:	Digital Signal Processing
ESCs:	Electronic Speed Controllers
FPGA:	Field Programmable Gate Array
FSBL:	First Stage Bootloader
Gyro:	Gyroscope
I/O:	Input/Output
I2C:	Inter-Integrated Circuit
IP:	Intellectual Property
IR:	Infra-red
JTAG:	Joint Test Action Group
LED:	Light-Emitting Diode
MicroCART:	Micro-Controller Aerial Research Team
PID:	Proportional-Integral-Derivative
PMOD:	Peripheral Module
PPM:	Positional pulse modulation
PWM:	Pulse width modulation
quad:	quadcopter
RAM:	Random Access Memory
RC:	Radio Controller
RPM:	Revolutions Per Minute
SD:	Secure Digital
SPI:	Serial Peripheral Interface
UART:	Universal Asynchronous Receiver Transmitter
USB:	Universal Serial Bus
XPS:	Xilinx Platform Studio
XSDK:	Xilinx Software Development Kit

3 Executive Summary

The goal of this project is to design, implement, and integrate a new quadcopter into the old quadcopter system for the Distributed Sensing and Decision Making Laboratory at Iowa State University. The old system contains black box components that cannot be tweaked to improve quad stability and limits the room for further system improvement. Additionally, the old system has slow feedback response due to its high coherence with a camera system, ground station, and RC transmitter. To counter many of the issues with the old system, the new quad is reconfigurable, expandable, and more independent.

4 System Design

The overall project contains several components, and each component requires multiple functions necessary for quadcopter flight. Besides the required functionality of the system, several non-functional requirements like high performance, hardware and personal safety, and accessibility were discussed.

4.1 High-Level System Description

This project is a continuation of multiple years of work within the Distributed Sensing and Decision Making lab and is the foundation of a new platform to be used in aerial research or applications for the lab. The main decision to revamp the project was to expand the quad's capabilities and reliability. Figure 1 shows the old quadcopter on the left which has limited and inconsistent peripherals and on the right, the new quadcopter built for this project.

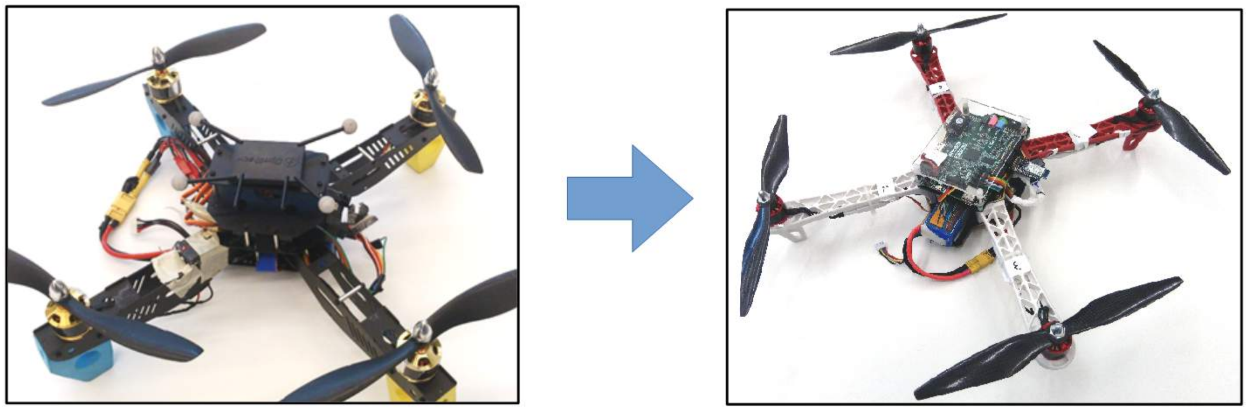

Figure 1: Project concept diagram

Because of its versatility, the new system that is in place allows for multiple projects to be developed at the same time. Rather than simply upgrade the old system's peripherals, the system has several swappable peripherals and reconfigurable hardware allowing for expandability and customization.

4.1.1 Quadcopter Vehicle

The quadcopter is the mobile unit in the group that performs most of the actuations in the system. Specifically, the quad is equipped with four motors and propellers that allow it to move in three dimensional space. To control the quadcopter, a user can use an RC transmitter to send throttle, roll, pitch, and yaw commands to the quad through its RC receiver. Additionally, the quad is equipped with a Bluetooth module to receive and send data from and to the ground station and internal controllers to make flying easier for pilots.

The quad system consists of several hardware pieces including the main chassis which contains the red- and white-colored physical frame, motors, ESCs, and propellers. To handle software computing, the body also contains a microprocessing board with reconfigurable logic and a sensor board equipped with a gyroscope, accelerometer, and magnetometer.

4.1.2 Ground Station

The ground station is a desktop computer that has a couple of critical roles to play in the system. The two roles of the ground station are to be the main communication base for the quad and to collect logs of flight data both from the quad itself and any operating environment sensors. To speak with the quad, the ground station communicates through its own Bluetooth module and mostly communicates commands from a user, data from an external sensor, or received log data. Once the ground station has successfully logged all the data, it uses a custom built MATLAB tool to plot the data for users to visualize the quad's measurements and calculations as well as the camera system data that is sent to the quad.

4.1.3 RC Transmitter

The RC transmitter is a controller that allows pilots to provide the quadcopter with direction and throttle commands. The controller allows for different configurations and trimming settings for pilots to improve direction biases. For this project, a pilot manually flies the quadcopter using the transmitter and is able to stop the flight program by disarming the quad, allowing it to finish logging data to the ground station.

4.1.4 Camera Environment

The Distributed Sensing and Decision Making Lab is equipped with 12 high-speed IR cameras that can be used to track a constellation of IR reflective balls. For this project, the camera environment is configured to track the X,Y, and Z position of the quadcopter as well as provide roll, pitch, and yaw angles. The camera data is sent to a desktop computer with special tracking software that collects the data. The camera system desktop then relays the directional and position information to the ground station that is further passed to the quad.

4.2 Functional Decomposition

The functional decomposition of project can be seen in Figure 2 and indicates what key functions each of the components perform—distinguishable by the color of the box: yellow for the camera system, purple for the ground station, red for the RC transmitter, and green for the quadcopter.

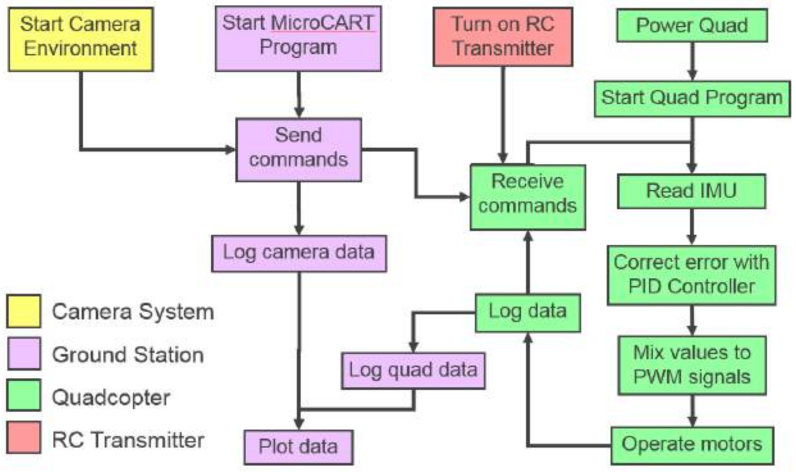

Figure 2: Diagram of the system's functional decomposition

The first few actions that need to be performed in the system are to start the camera system and ground station software and turn on the RC transmitter. At the same time, the quadcopter can be powered and turned on to start its on-board program. Once initialized, the quad goes through its main loop consisting of reading sensor data, calculating error based off its desired orientation and measured orientation, adding the corrections to the pilots RC commands, and then outputting the result to the motors to move the quad toward its goal. Afterwards, the quad logs its measured and calculated data, reads in the pilots commands and any from the ground station and continues to repeat the loop. During this process, the ground station is logging the camera data it sends to the quadcopter and receives the quad's logged data after the pilot has terminated the quad's flight program to be logged and eventually plotted.

4.3 System Requirements

4.3.1 Functional

Board and Peripheral Mounting
For the quad to operate, the Zybo board, among its peripherals, need to be mounted and placed on the new quad chassis so the microprocessor can operate the motors. Additionally, the main sensor board that measures gyro and accelerometer data needs to be placed at the center of gravity on the quad to receive useful data. In regards to how the board will be powered, external, mobile, and replaceable power supplies are required to be placed on the quad to power the microprocessor board, ESCs, and motors so the quad can operate in a wide environment and not be constrained by wires.

Power Supplies
There are a few requirements for the power supply batteries, both to the quad motors and the microprocessor board, including the need for proper voltage regulation, detecting reverse polarity, and not allowing for over-drain so the batteries do not get damaged if left unattended. To fit these requirements, special circuits were created to step down the voltage and let users know when batteries are getting too low.

Quadcopter

The software for the new quad system is to be able to collect several readings from its main sensor peripheral through an I2C protocol, communicate data through Bluetooth to the ground station, receive PPMs through an RC receiver or Bluetooth, and appropriately mix incoming position signals to PWM signals for each of the four motors on the quad.

Additionally, the processing board does contain volatile memory and a reconfigurable hardware, so the board needs to be configured without having to plug it into a computer.

Ground Station
The ground station is required to communicate data from all devices in the system including the camera system and the quadcopter. The ground station is also required to have data analytic tools to interpret the diverse range of data being received and transmitted. To communicate with the system, it needs to have the necessary hardware including, but not limited to, Bluetooth and Ethernet.

Camera Environment
In order for the camera system to function, power needs to be supplied to the cameras as well as any communication devices. Each camera needs to have a data line in order to be communicated back to the computer and processed. Lastly, the computer needs to be able to communicate large amounts of data to the ground station using Ethernet.

In regards to software, the tracking software needs to be on a desktop computer in order to specify IR constellations and settings as well as control the power and operation of the cameras.

4.3.2 Non-Functional

Board and Peripheral Mounting
The most important aspect of the new system is that it does not injure any of its components when testing. For this reason, the placement of the Zybo board and its peripherals on the quad chassis is critical to not damage equipment which can cost time and money to replace. Also, the mounted board should not be the tallest point on the quad since the quad will tip over upside down, and it should be protected and stable during test flights. Additionally, accessibility should be considered when mounting the board and peripherals so they are easy to access and remove, if needed.

Power Supplies
The power supply batteries used in the system need to be monitored and handled properly when their power is too low to operate the quad. For maintainability, the quad needs to safely land rather than stop altogether when in a flight, and the batteries need to not be damaged in the process. There is some protection from this already in that the power supplied to the motors will gradually descend when the battery is low instead of a digital fashion where it would automatically switch off. Nevertheless, battery information needs to be accessible for developers to allow for better quad's flights.

Quad System
There are many timing constraints that the quad should meet and the data bottleneck should be minimal when communicating data back to the ground station so the quad can have a smooth flight. The system should also be reliable and shouldn't need to be reset or configured after each flight, or behave strangely when launching the main program. Overall, users shouldn't have to touch the software aspects of the quad when starting the basic flying program.

Ground Station
The software for collecting data from the IR cameras and sending the correct instructions through the Bluetooth should not be of any concern to users wanting to fly the quad. These components should also be reliable and not need to be tampered with unless absolutely necessary. The data analytic tools also need to be automated so users do not need to dig through log files and look for scripts to run in order to view results from a flight. Lastly, the PID controllers on the system need to be optimal to make flying the quad easy and safe for users.

4.4 System Analysis

The largest difference between the new and old system is the shift of more computing on the quadcopter from the ground station. Before, the ground station was responsible for the control of the system and required a clunky Nexsys2 Spartan-3E FPGA board used to connect the ground station to an RC transmitter to allow the computer to send commands. While the specifications of the camera system will be discussed later, this method is slow due to the camera system's update rate and latency in combination with the RC trainers refresh rate of 50Hz. With the new system, the extra hardware is removed and also reduces some coherence in the system. While the new system currently uses a 50Hz RC transmitter as well, the quad has a much faster internal update rate and is able to obtain commands faster just through Bluetooth alone. Since the new quad is expandable, Wi-Fi can be added to the quad and have even faster communication.

Another difference between the two systems is how they handle PID controls. As previously mentioned, the old ground station used PID controllers for the camera data which were calculated and then sent to the quad through RC commands. On the other hand, the new system has all PID calculations and corrections on-board. This shift allows for the quad to become independent and becomes the focus of the system rather than an actuation of the system.

A main attraction of the new system is the reconfigurable logic on the microprocessor in the form of a Xilinx FPGA. While the new system has most of its logic for communicating with peripherals, configurable logic can be created to offload processing to speed up software processes on-board or make it possible to include compute intensive tasks like video.

4.5 Operating Environment

The new system was developed and tested within the Distributed Sensing and Decision Making Lab located in room 3050 of Coover Hall, as seen in Figure 3. The lab has different equipment for signal debugging, physical component characterization, single or multiple axis testing, and a station for charging batteries. The lab also has houses the IR camera system for directional and position measurements and has plenty of desktop computers for programming the quadcopter, collecting camera data, and acting as the ground station.

Although there are a few exceptions, this operating environment was ideal for quad tuning, testing, and flying as there are no natural forces like wind that disrupt conditions.

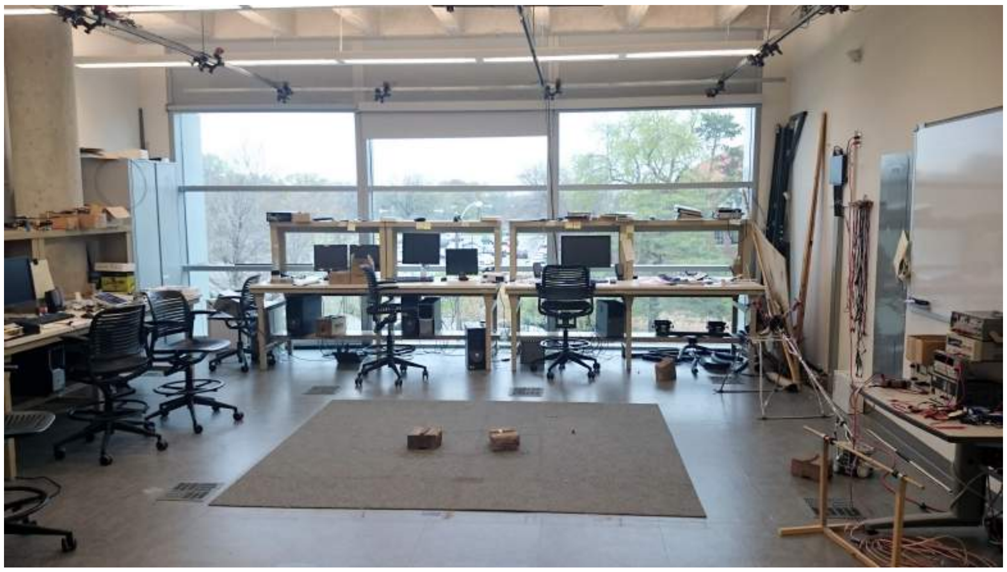

Figure 3: Coover 3050 Laboratory: MicroCART Operating Environment

5 Detailed Design

Inside the high-level components are specific products, items, and tools that are used to make them up. This section will dive into specifics of each item and module and explore the the use of the items within the component.

5.1 Hardware Specifications

5.1.1 DJI FlameWheel F450

The new quad system utilizes a four arm DJI FlameWheel to house and carry the proper components in order to fly the quadcopter in the system. The chassis has two platforms: one that sits on the intersection of the four arms, and one below that acts as a base. Both of the platforms are conductive and they share a single pair of wire leads for powering the ESCs and motors tightly constrained to the chassis' arms. The exact specifications and recommendations for the FlameWheel can be seen in Table 1.

Spec	**Description**
Frame Weight	282g
Diagonal Wheelbase	450mm
Takeoff Weight	800g ∼ 1200g
Recommended Propeller	10 x 3.8in; 8 x 4.5in
Recommended Batter	3S ∼ 4S LiPo
Recommended Motor	2212 ∼ 2216 (Stator Size)
Recommended ESC	15A ∼ 25A

Table 1: Specifications of the DJI FlameWheel F450

The FlameWheel carries the Zybo board on its top platform with batteries, additional peripherals, and the RC receiver mounted on the lower platform since there is more room. Lastly, the main sensor board is mounted on the top platform near the center of gravity for the most accurate measurements. A picture of the fully assembled quad can be seen in Figure 4.

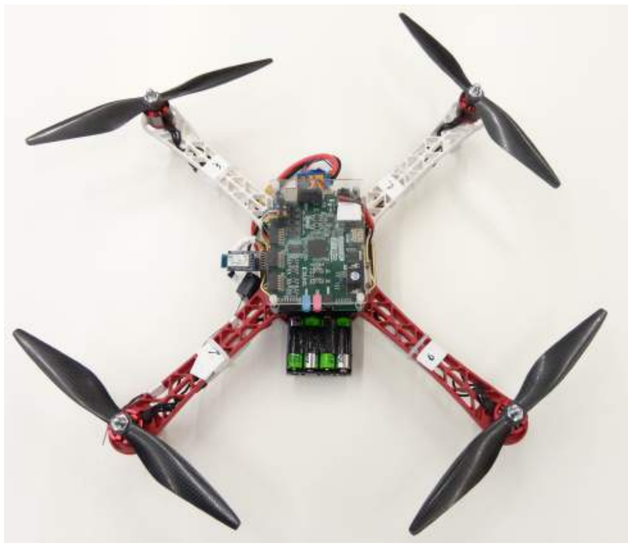

Figure 4: Picture of quadcopter with distinguishable white and red arms

5.1.2 ESCs

The ESCs act as the middle-man between the Zybo board and the motors. The ESCs are responsible for translating the PWM signals sent by the Zybo board into the amount of power to give the motors in order to spin. The ESCs are attached to the quad frame by zip ties and are powered through the main quadcopter battery plug-in on the frame. The ESCs the team are using are specified in Table 2 and a photograph of one of them can be seen in Figure 5.

Spec	**Description**
Electric Current	30A
Compatible Signal Frequency	30Hz - 450Hz
Battery	3S $\sim$ 4S LiPo

Table 2: Specifications for the DJI ESCs

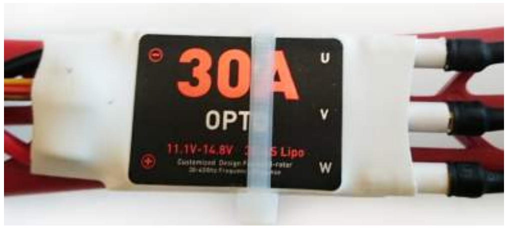

Figure 5: Picture of an Electronic Speed Controller (ESC) that controls the speed of the motors.

5.1.3 DJI Brushless Motors

The motors are the key actuators in the system that spin and eventually move the quad with the help of propellers. The DJI motors are brushless stepper motors that are controlled by magnets inside the motor that generate precise magnetic fields to make the motors rotate up to 10,000 RPM. With four total motors on the quad, proper mixing of signals is required for the motors to change their number of RPM so the quad can turn or rotate correctly. The specifications for the DJI motors can be seen in Table 3 along with a photo of one of the motors in Figure 6.

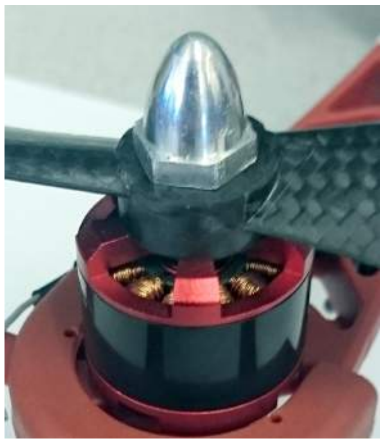
Figure 6: Picture of a motor with a propeller and screw cap on

Spec	**Description**
Stator Size	22 x 12mm
Brushless Motor KV	920rpm/V

Table 3: Specifications of the DJI Brushless Motors

5.1.4 Propeller Blades

For this project, the quad utilizes a 10 inch long propeller with a 3.8 inch pitch. Generally, the longer the propeller blades, the more stable flights can be and similarly the lower the pitch, the more stable and less vibration is caused.

Two sets of propellers were used for this project with the main difference being the material which the blades are made of. A picture of different propeller blades used can be seen in Figure7. Specifically, the original set of propellers were made of plastic (top and bottom propellers in the figure) while the majority of propellers used currently are made of carbon fiber (middle propeller). For a set of propellers, two blades are pitched clockwise and the other two are pitched counter-clockwise to provide spin balancing.

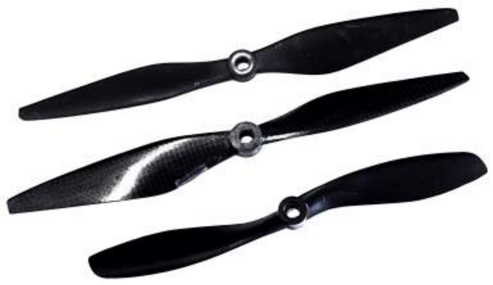
Figure 7: Picture of different propellers with varying length and pitch

For the quad, clockwise propellers are placed on arms 0 and 3 while counter-clockwise propellers are used on arms 1 and 2. If users switch the configuration, the quad will push itself into the ground rather than lift off.

5.1.5 Diligent ZYBO Zynq-7000 Development Board

The Zybo board is the core processing unit on the quadcopter and essentially the entire system. It is responsible for mixing controls from the ground station, handling gyro and accelerometer data for its sensors, and eventually handling PID controllers. Additionally, the board is responsible for communicating data appropriately to the ground station through the use of Bluetooth. Of course, the board has many more capabilities and a full list of features the Zybo board offers can be seen in Table 4.

Component	**Description**
Processor	650MHz dual-core Cortex-A9 Processr
Memory	DDR3 Memory Controller with 8 DMA channels
High-bandwith peripheral controllers	1G Ethernet, USB 2.0, SDIO
Low-bandwith peripheral controllers	SPI, UART, I2c
FPGA	28K logic cells 240KB Block RAM 80 DSP slices On-chip dual channel, 12-bit ADC
Misc.	MicroSD slot 6 pushbuttons, 4 slide switches, 5 LEDs 6 PMOD connectors On-board JTAG programming and UART to USB converter

Table 4: Features of the Diligent ZyBo Board

The Zybo board is used on the new system because it has 6 Diligent PMOD connectors and a Xilinx Zynq FPGA that can programmed to be utilize different peripherals including the 9 Degrees of Freedom sensor board, Bluetooth module, and the outputs signals for controlling the motors. Another reason the board was selected is due to its many internal controllers for UART, I2C, and SPI so developers can further remove load from the software.

This board is ideal for the new system because of its unique reconfigurability and amount of external peripheral support. Conveniently through the use of the Zybo board's coherent AXI-Lite bus, the CPU of the board can access Xilinx IP and custom cores that contain registers to read or write values that further expands usability. A photograph of the Zybo board can be seen in Figure 8 with several of its PMOD connectors in use.

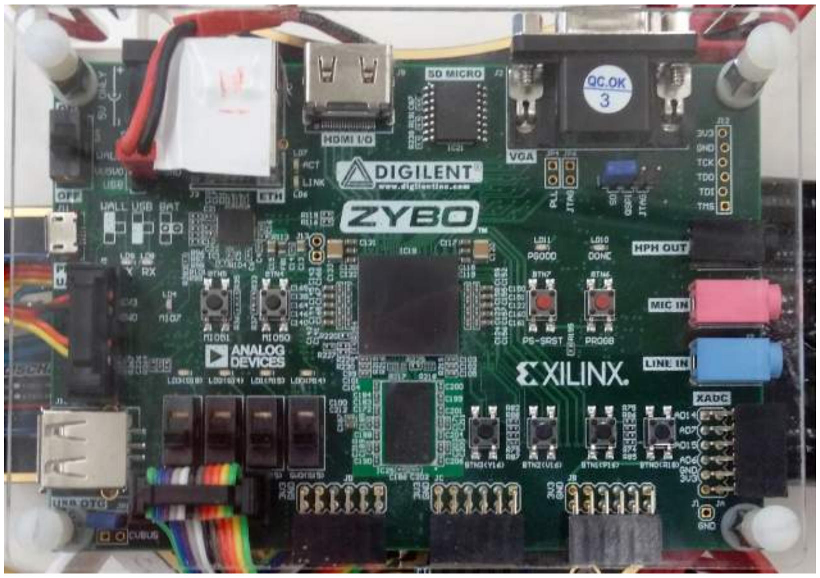

Figure 8: Picture of the quadcopter's dual-core microprocessor: Diligent ZYBO Board

For programming the Zybo board, users can develop a bitstream file through XPS to be flashed onto the FPGA that can configure the processing system and programming logic portions of the chip on the board. The actual process of programming is done either through UART JTAG if connected to a computer in the XSDK or through a microSD card if set up properly. The latter method is heavily used to allow users to run the quad. As long as the appropriate program is on the SD card, a user only needs to turn on the board to program it.

5.1.6 Diligent PMOD-BT2 - Bluetooth Interface

For communication, a Diligent Bluetooth module is used on the Zybo board to send data to the ground station for analysis or receive external sensory information from the camera system. The reason why the team chose to utilize a Diligent PMOD-BT2 device was due to its PMOD connection type and easy to use configurations both in hardware and software.

Internally, a Roving Network IC is used for actual Bluetooth communication while Diligent has expanded the interface to work for their products. For the project, the Bluetooth module operates at 921,600 baud rate for high data throughput. A photo of the Bluetooth module can be seen in Figure 9 with its specifications in Table 5.

Figure 9: Picture of the quad's Bluetooth module

Spec	**Description**
BT Compatibility	2.1/2.0/1.2/1.1
Size	0.8" x 1.5"
IC	Roving Networks RN-42
Baud Rate	1200 - 921K
Encryption	AES128

Table 5: Specifications for the Diligent PmodBT2

5.1.7 SparkFun MPU9150: 9 Degrees of Freedom

The 9 Degrees of Freedom board contains several useful sensors including a gyroscope, accelerometer, temperature sensor, and a magnetometer. These sensors are used to find roll, pitch, and yaw rotations and angles (X,Y, and Z) to tell if the quad tilted to one side or another.

The specifications for the board can be seen in Table 6 and a photograph of the board can be seen in Figure 10 which shows the different pins and the directions of the X, Y and Z for the gyroscope, accelerometer, and magnetometer. For this project, the maximum sensitivities are used to not limit the quad's capabilities.

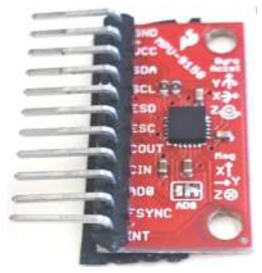

Figure 10: Picture of SparkFun MPU9150

Spec	Description
VDD	2.375 - 3.465V
Gyro Sensitivity	$\pm$250, 500, 1k, 2k ^{o}/sec
Accel Sensitivity	$\pm$2, 4, 8, 16g
Mag Sensitivity	$\pm$1200 μT

Table 6: Specifications for the SparkFun MPU9150

To communicate with the board, an I2C protocol master needs to be present in order to read and write to the board. Luckily, the Zybo board contains two of these types of controllers and makes working with the board easier.

5.1.8 SpekTrum DX6i RC Controller

The RC transmitter is the device for communicating the thrust, roll, pitch, and yaw values to the quad system though it is slow and can only operate at 50Hz. The RC receiver located on the quad receives the 6 channel PPM and splits the channels to individual PWMs for each flight dynamic. The specifications for the SpekTrum RC trainer can be seen in Table 7.

Spec	Description
Modulation	DSM2
Band	2.4GHz
Receiver	AR6200
Programming Features	Helicopter & Airplane
Model Memory	10
Modes	Mode 2
Transmitter (Tx) Battery Type	AA NiMH 1500mAh batteries
Charger	4-cell 150mAh wall charger

Table 7: Specification of the SpekTrum DX6i RC Controller

Besides providing aerial directions to the quad, the RC transmitter also contains a switch to stop the quadcopter's flight program. A picture of the RC transmitter and receiver can be seen in Figure 11.

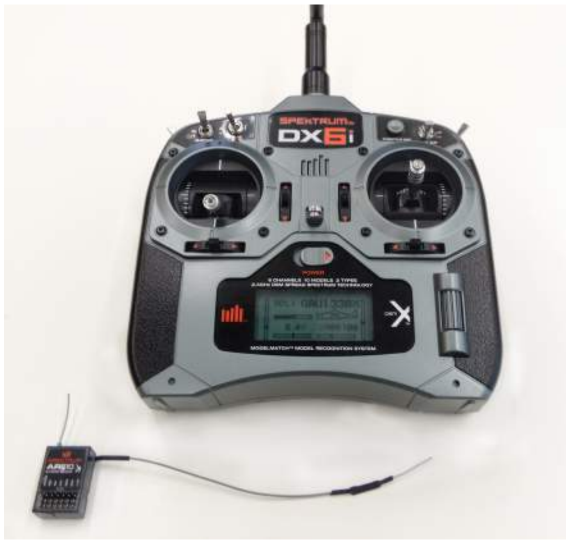

Figure 11: Picture of the Spektrum DX6i Transmitter and AR6200 receiver

5.1.9 Voltage Regulator and Power Management

Voltage regulation and power management is key to powering the Zybo board and the quad motors. To manage the different aspects, two separate circuits were created to power the Zybo board, identify the motor battery's level, detect reverse polarity, and stop power from being drawn from the battery after a certain level. The specifications for the voltage regulator for the Zybo board can be seen in Table 8, the circuit schematic can be seen in Figure 12 while the motor power board can be seen in Figure 13 with important parts labeled.

Spec	Description
Input Voltage Range	6.0V - 8.4V
Output Voltage	5.0V
Output Current	2.5A
Low Battery Threshold	6.7
Normalized Low Battery Threshold	0.799V
Size	2.01" x 1.23" (51x31mm)
Fabrication Cost	$4.10/board

Table 8: Specifications for the Zybo Battery Regulator

Spec	Description
Input Voltage Range	9.0V - 12.6V
Low Battery Threshold	9.8V
Normalized Low Battery Threshold	0.777V
Size	1.06"x1.15" (27x29mm)
Fabrication Cost	$2.04/board

Table 9: Specifications for the Motor Battery Regulator

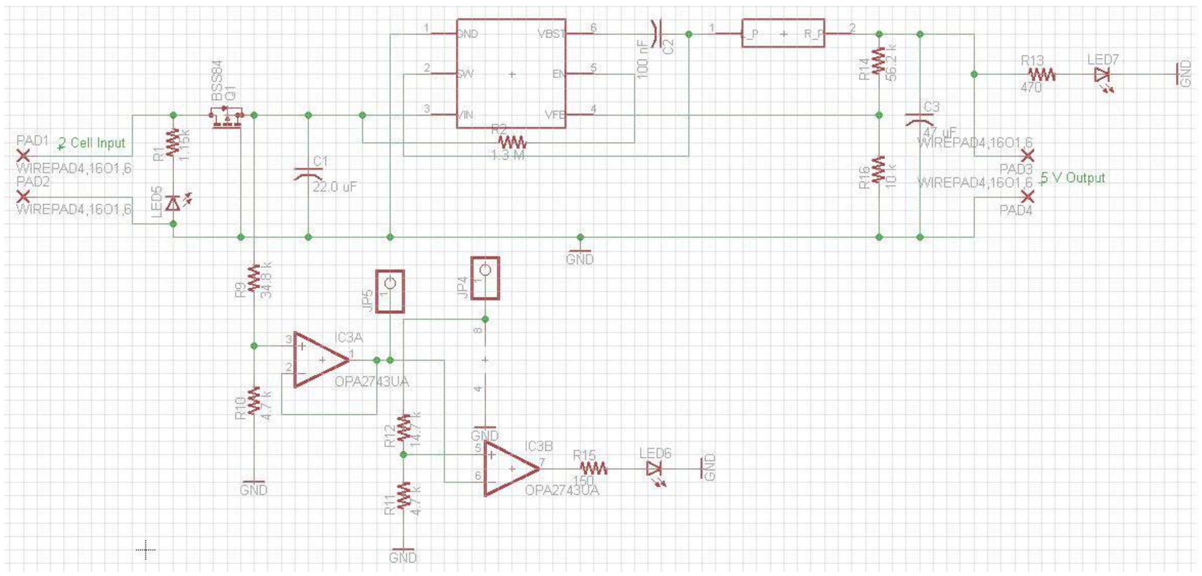

Figure 12: Circuit design for the Zybo's voltage regulator and power management board

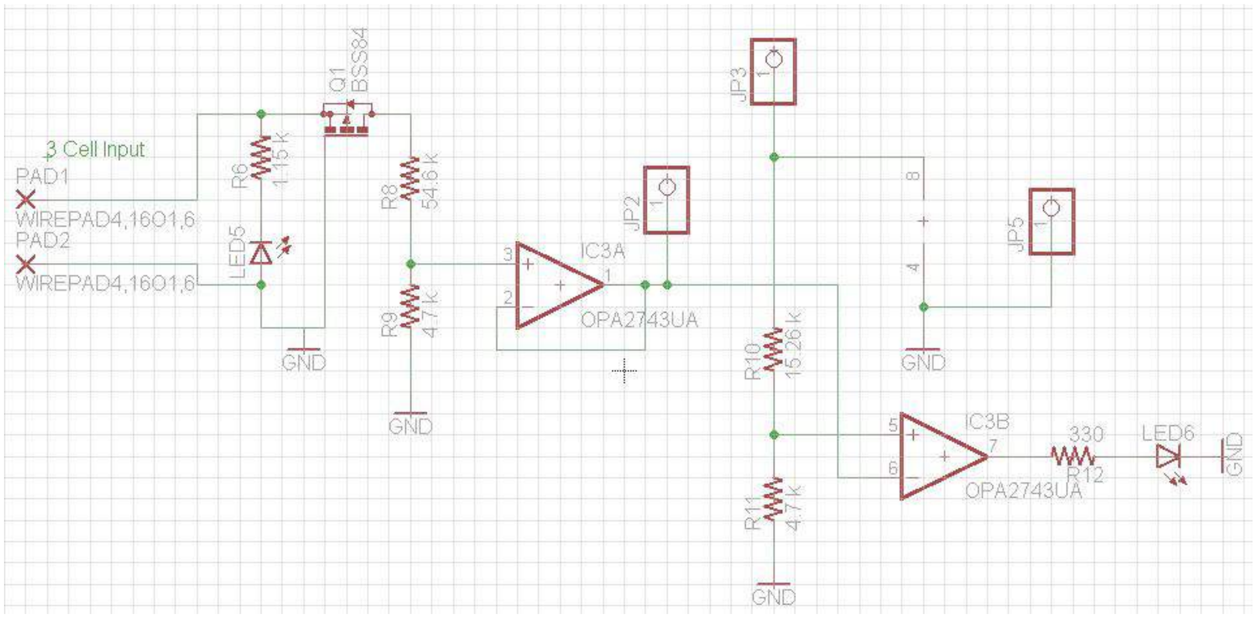

Figure 13: Circuit design for the quad motor's power management board

For the actual PCB routing for the Zybo and motor power management boards, the diagrams can be seen in Figures 14 and 15

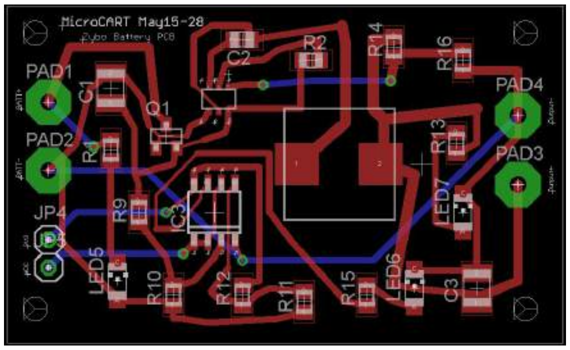

Figure 14: PCB routing diagram for the Zybo power management board

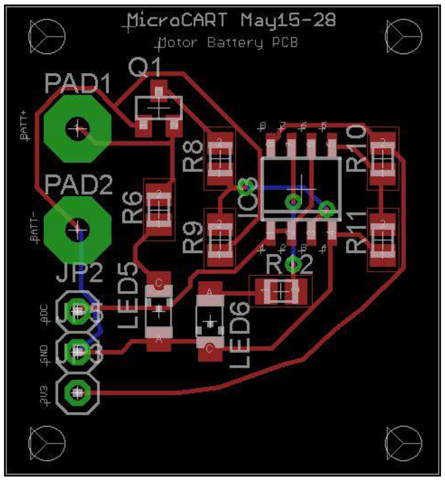

Figure 15: PCB routing diagram for the motor batteries

5.1.10 OptiTrack IR Cameras

The OptiTrack cameras in the lab are set to capture reflected IR waves from the quad with the proper pieces of equipment. In order to track the quad's location, the quad frame carries a set of reflective IR balls that the cameras can track. Again, the data produced from the cameras are sent to an intermediate desktop through USB 2.0 where it is then sent to the ground station through Ethernet. More specifications for the camera system can be seen in Table 10 and a photo of a camera can be seen in Figure 16.

Figure 16: Picture of an OptiTrack IR Camera

Spec	Description
Resolution	0.3MP (640 x 480)
Frame Rate	100 FPS
Horizontal FOV	46^o
Filter Switcher	Optional
Interface	USB 2.0
No. of LEDs	26
Latency	10ms

Table 10: Specifications for the OptiTrack IR Cameras

For this project the camera system is only used for yaw (direction) data to be used in a yaw PID controller to keep the quad directed in a northward direction. While incorporating the quadcopter into the camera system would provide the possibility for positional data and have precise location feeds, a goal for the project is to remove the need for the camera system.

5.2 CAD Drawings

The quad frame requires several mounts for the Zybo board, peripherals, and testing equipment. Physical support pieces were created using CAD software for each including a reflective IR constellation mount, a base platform for securing the quad to a testing platform, and support beams for mounting the Zybo board to the chassis; all of these pieces can be seen in Figures 17, 18, and 19.

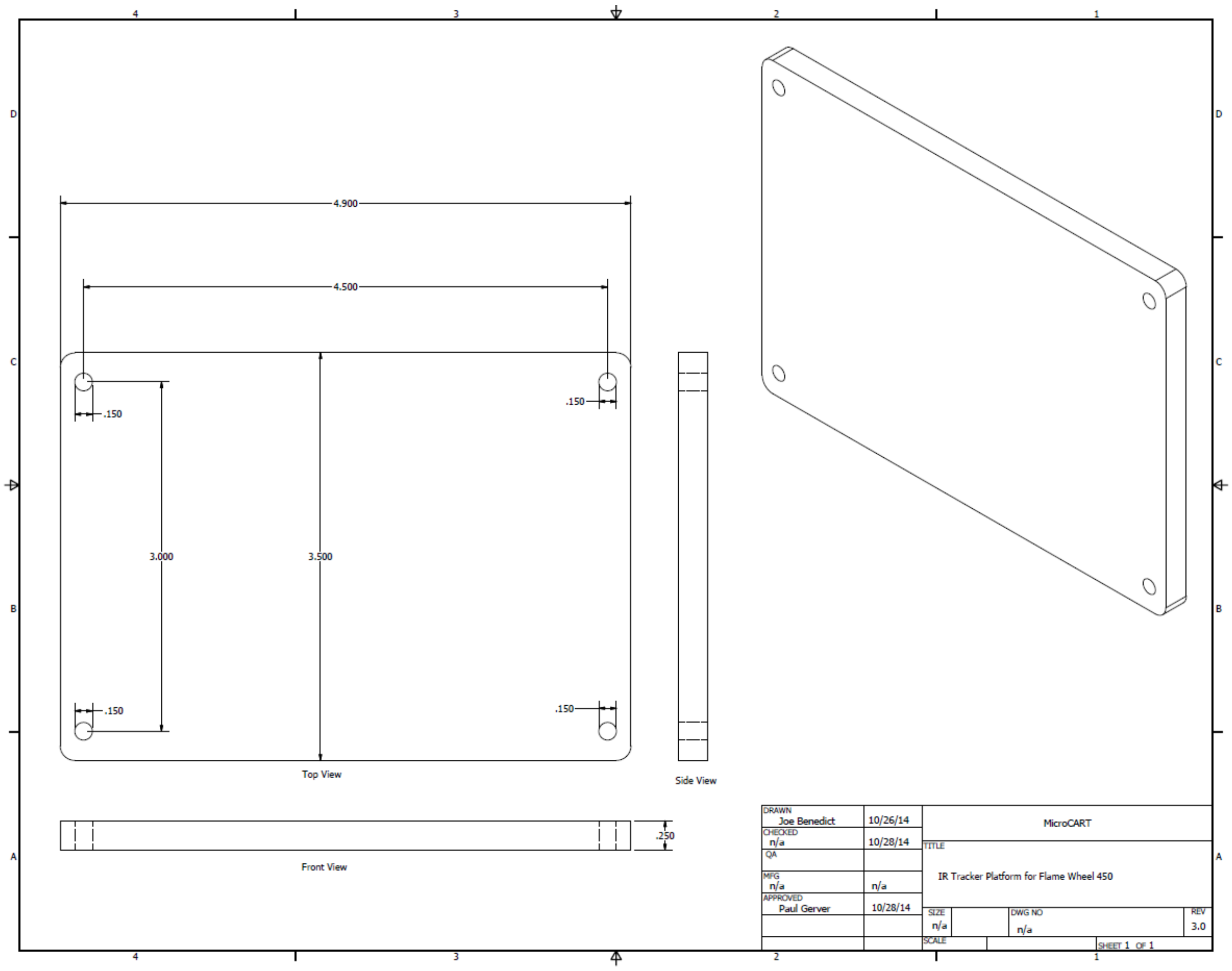

Figure 17: CAD diagram for chassis' IR mount

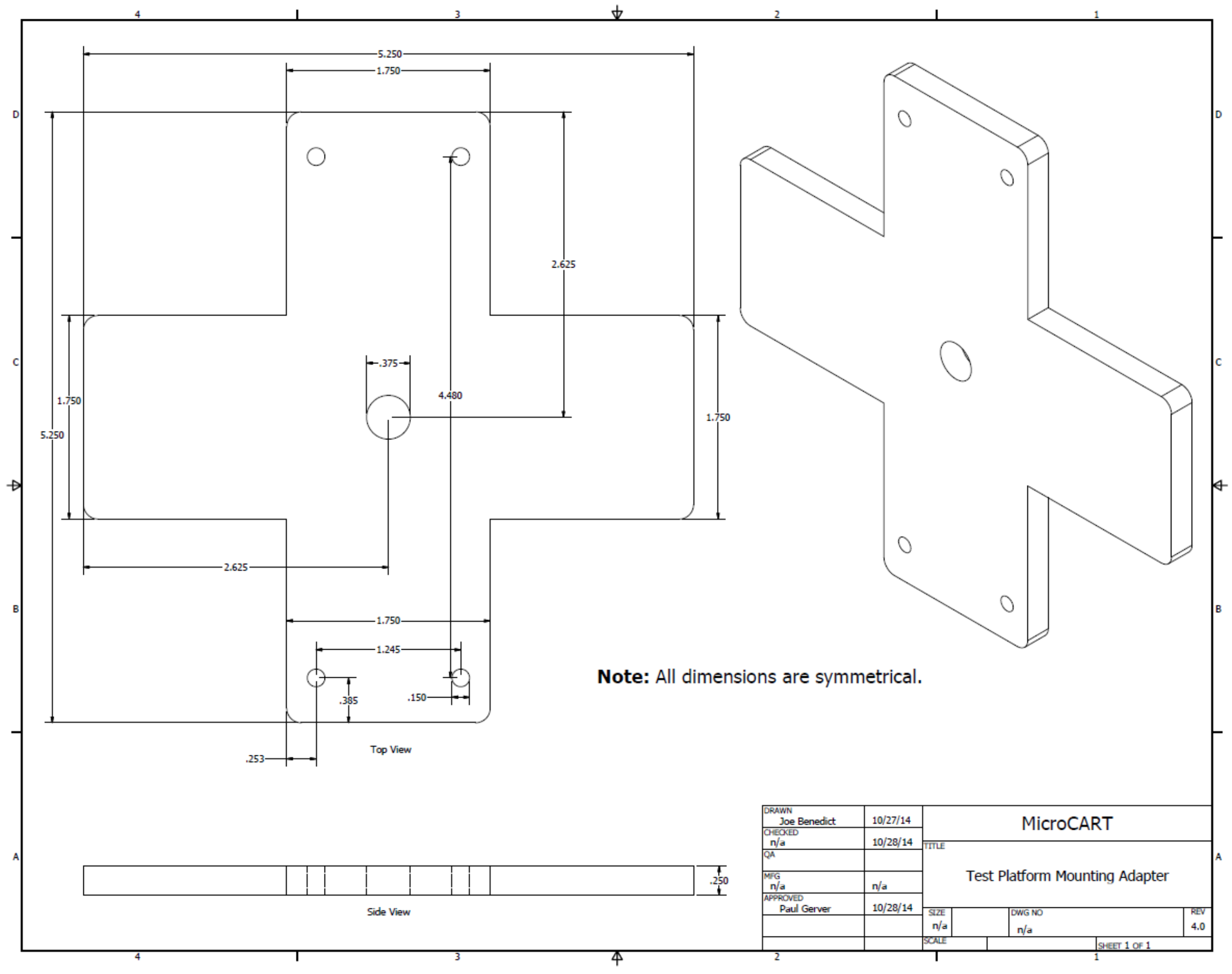

Figure 18: CAD diagram for chassis' test platform

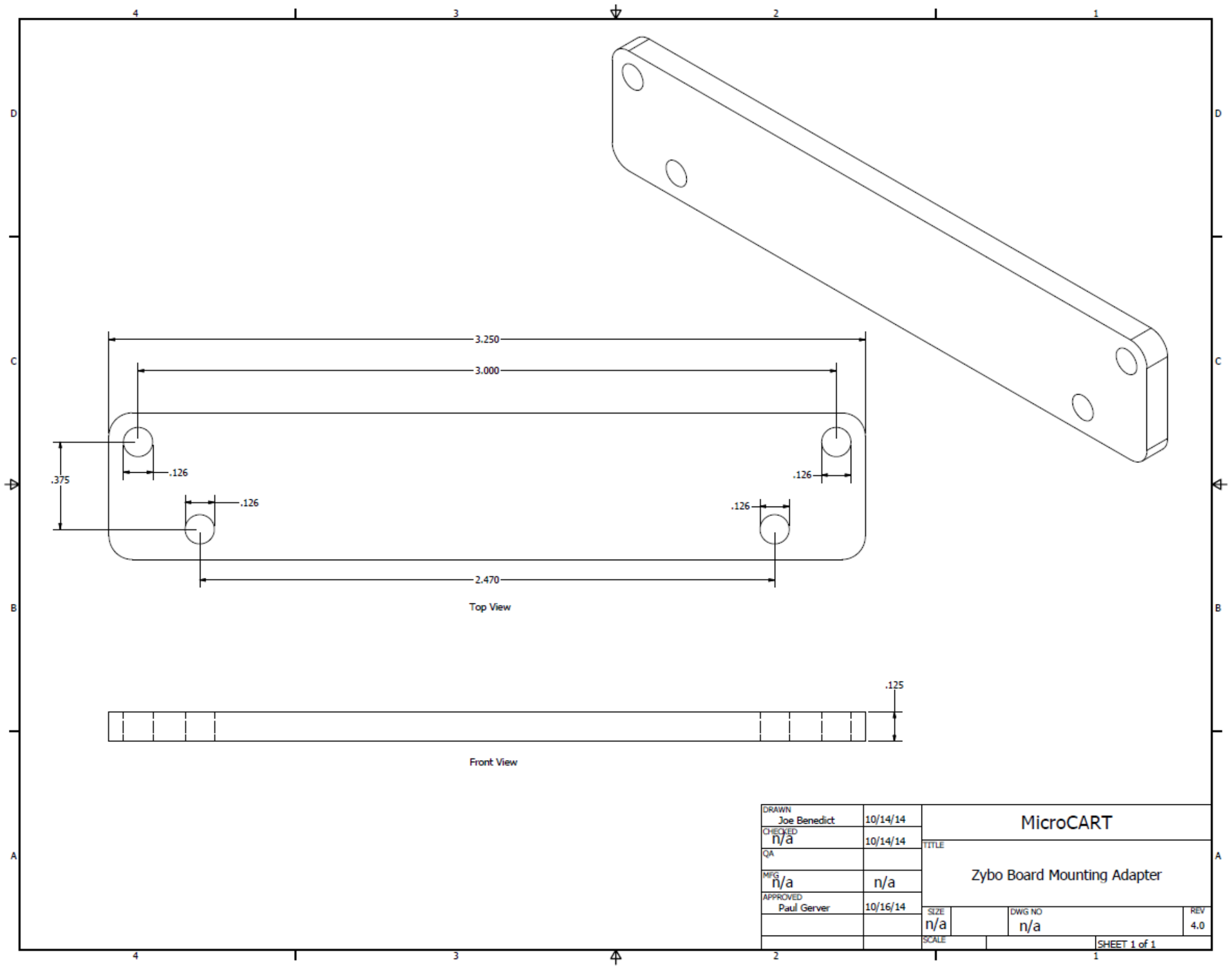

Figure 19: CAD diagram for Zybo board mounting adapter

5.3 Zybo Board Software and Logic Specification

While the quadcopter hardware is a conglomerate of pieces that are used to fly the quad, software is needed to orchestrate the different pieces and peripherals to work together. Additionally, there are several software modules that are used to communicate, control the quad, and provide data analytic tools for all the data in the system.

5.3.1 Sensor Board

For each peripheral added to the Zybo board, appropriate software needs to accommodate the new module. By developing through the Xilinx Software Development Kit (XSDK), users can initialize, configure, and operate devices through C code and pre-defined function calls thanks to Xilinx's provided libraries.

One example use case for programming the Zybo board is the software required to speak with the SparkFun MPU9150 sensor board. The sensor peripheral uses I2C to communicate, so the board's software needs to accommodate for the board's internal I2C controller. A process flow for this specific example can be seen in Figure 20; however, the initial setups seen in the diagram are similar across different peripherals with the only difference being the type of controller or I/O.

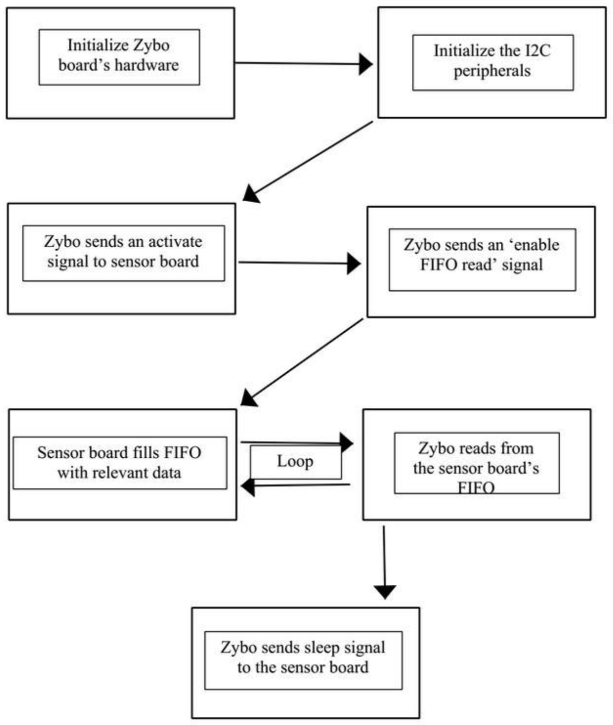
Initialize Zybo board's hardware
Initialize the I2C peripherals
Zybo sends an activate signal to sensor board
Zybo sends an 'enable FIFO read' signal
Sensor board fills FIFO with relevant data
Loop
Zybo reads from the sensor board's FIFO
Zybo sends sleep signal to the sensor board

Figure 20: Program flow for reading data from the SparkFun MPU9150

To initialize the sensor board, the main program on the Zybo board sends a wake-up signal to the board and then configures the sensitivity of the instruments and also configures filtering features that are available on the MPU9150. Once initializations have occurred, the main program reads several test gyro and accelerometer readings to ensure no errors occur. If something is found wrong either by corrupt data or no data received, the program will quit. Currently, it takes ~465 microseconds to obtain a gyro, accelerometer, and magnetometer reading in its entirety.

5.3.2 Sensor Filtering

When working with sensors like gyroscopes, accelerometers, and magnetometers, there are several frames of reference for a rigid body to be under when rotation. To convert the quad in its inertial frame of reference to Euler angles which mostly account for rotation fixes, a linear system transformation is applied to the gyroscope readings based off the the current X and Y angles, ϕ and θ respectively:

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

One issue that arises with performing these calculations is the occurrence of gimbal lock which can occur when the quad is tilted 90 degrees off an axis and can result in undefined answers.

Although not a functional requirement, sensor filtering is crucial for the quadcopter to appropriately use its measured gyro and accelerometer readings in flight. To combine the accelerometer and gyroscope readings, a complimentary filter is used to identify the angles the quad is oriented. Since gyros are quick to react, a high weight is given to the gyroscope and previous angle of the quad while a small weight is given to the current reliable accelerometer angles if not moving radically. The final complimentary filter values are used heavily in the PID controllers and are set to be zero degrees to keep the quad parallel to the ground.

$$\begin{aligned}\text{Complimentary Angle} = 98\% \text{ of Previous Angle} + \text{Current Gyro Velocity} * \text{Loop Period}\\ + 2\% \text{ of Accelerometer Angle}\end{aligned}$$

5.3.3 PID Controller

A Proportional-Integral-Derivative (PID) controller is a basis for steady quad flight. A PID controller works well because it factors in the present, past, and future error of the system and uses it to correct itself rather than overshoot its ideal destination. Based off a desired location, the controller will take a weighted sum of the different errors and provide feedback to the current system. The main quad programs code for the PID controller can be seen below:

```
PID_values pid_computation(PID_t *pid) {

  float P=0, I=0, D=0; // Each term's contribution
  //float T;        // Time between runs
  float error = pid->sensor - pid->setpoint; // Error in the system

  // Accumulate the error
  if (fabs(pid->Ki) <= 1e-6) {
    pid->acc_error = 0;
  } else {
    pid->acc_error += error;
  }

  // Compute each term's contribution
  P = pid->Kp * error;
  I = pid->Ki * pid->acc_error * dt;
  D = pid->Kd * (error - pid->prev_error) / dt;

  PID_values ret = {P, I, D, error};

  pid->prev_error = error;  // Store the current error into the structure

  pid->pid_correction = P + I + D;  // Store the computed correction
  return ret;
}
```

For this project, nested PID controllers are used for each of the three axes: one for the angular velocity and one for the angle of the quad. Here, the angular velocity PID is called the inner PID and the angle is the outer PID which feeds into the inner one. A PID control diagram can be seen in Figure 21 and shows the outer PID correction value being set as the inner PID's setpoint, specifying a certain angular velocity the quad should be at. For example if the outer PID specifies an angular velocity of 20 degrees per second and the quad is not be moving, then the inner PID provides a correction of the RC commands that are adjusted and eventually outputted to the motors.

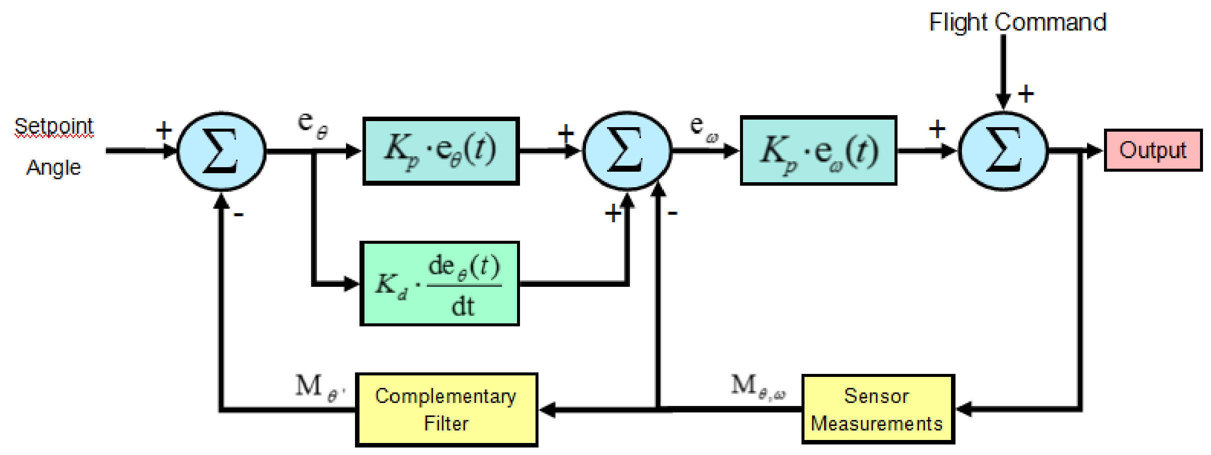

Figure 21: Diagram of PID Controller Flow

The key components of the PID controllers are the coefficients used to weigh each error (i.e. P, I, and D coefficients). Through several tuning and testing sessions, the new quad's PID coefficients were obtained to provide the right amount of correction for each unit to stabilize the quad when it is moved or disoriented. For the final product, the PID coefficients can be seen in Table 11.

PID	**(P,I,D) Coefficient**
Pitch Angle	(-7, 0, -0.45)
Roll Angle	(-6, 0, -0.4)
Yaw Heading	(60, 0, 0)
Pitch Angular Velocity	(-96, 0, 0)
Roll Angular Velocity	(-96, 0, 0)
Yaw Angular Velocity	(470, 0, 0)

Table 11: Quadcopter's PID Coefficients

5.3.4 Bluetooth

To operate the Bluetooth, software was created to utilize one of the Zybo board's UART controller, UART0, to enable sending and receiving data easy through Xilinx's pre-defined functions for checking FIFOs and sending bytes through either polled or interrupt-based approaches. As mentioned earlier, the Bluetooth is configured to run at 912,600 baud rate which took some investigation to work as Xilinx sets the default baud rate and maximum value to 115,200.

There are a few types of commands that the Bluetooth can receive through a user-defined data packet: an update or a command. An update data packet is received on the quad frequently to update position or direction from the camera system data while a command can be used to control PID coefficients or setpoints to allow for in-flight tuning which greatly speeds up the process. A diagram of the entire process can be seen in Figure 22 and shows the different paths it takes depending on the type of packet.

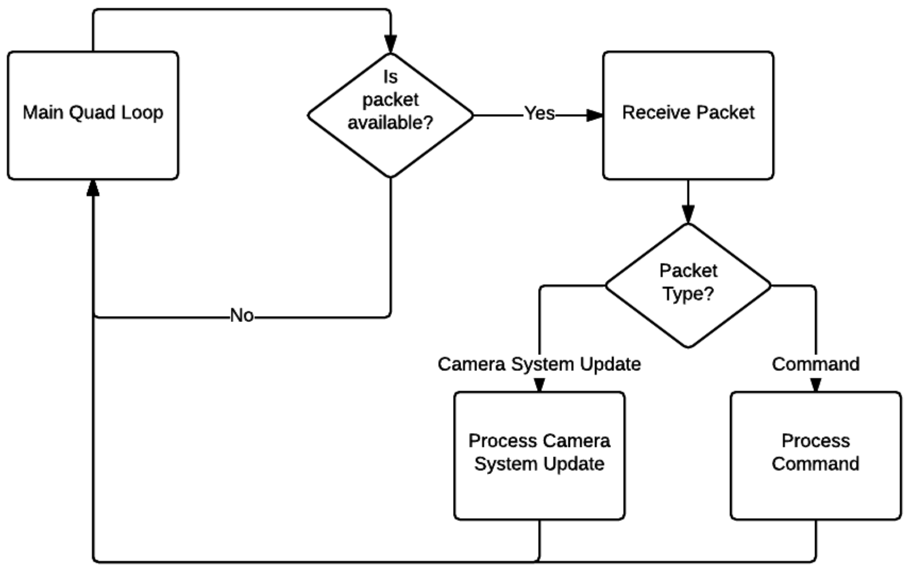

Figure 22: Diagram of quad's Bluetooth data packet procedure

5.3.5 RC Receiver Recorder

To collect PWM signals from the RC receiver, several logic cores were created to record the signals coming into the board and count the amount clock cycles the signal was high. By placing the signal count into a register, a user can read the custom register in software thanks to the AXI-Lite bus and, for example, tweak the values for stability reasons. The clocking used on the RC recorder core runs at 100MHz, so an expected value to be in the register ranges from 100,000 to 200,000 because there are that many clock cycles in a 1ms to 2ms time period. Once through the main quadcopter program, the registers are read and placed as raw values before any software mixing or processing is done.

5.3.6 Control Mixer

The control mixer for the system mixes the incoming throttle, roll, pitch, and yaw PWMs from the RC receiver or Bluetooth values and translates the individual signals for a flight component to a signal that all four motors on the quad react to. For example, if a users wants to move the quad horizontally to the right, the RC receiver will change the roll to rotate slightly clockwise which means the two motors on the left side of the quad need to increase power. The code snippet for the mixing can be seen below:

```c
/**
 * Converts Aero 4 channel signals to PWM signals
 * Aero channels are defined above
 */
void Aero_to_PWMS(int* PWMs, int* aero) {
  // Throttle, pitch, roll, yaw as a percentage of their max - Range 0.0 -
      100.0
  float throttle_100 = (aero[THROTTLE] - THROTTLE_MIN) / (THROTTLE_RANGE*1.0);
  float pitch_100    = (aero[PITCH]    - PITCH_MIN)    / (PITCH_RANGE*1.0);
  float roll_100     = (aero[ROLL]     - ROLL_MIN)     / (ROLL_RANGE*1.0);
  float yaw_100      = (aero[YAW]      - YAW_MIN)      / (YAW_RANGE*1.0);
  // This adds a +/- 300 ms range bias for the throttle
  int throttle_base = BASE + (int) 60000 * (throttle_100 - .5);
  // This adds a +/- 200 ms range bias for the pitch
  int pitch_base    =        (int) 60000 * (pitch_100    - .5);
  // This adds a +/- 200 ms range bias for the roll
  int roll_base     =        (int) 60000 * (roll_100     - .5);
  // This adds a +/- 75 ms range bias for the yaw
  int yaw_base      =        (int) 15000 * (yaw_100      - .5);

  int pwm0, pwm1, pwm2, pwm3;
#ifdef X_CONFIG
  pwm1 = throttle_base + pitch_base/2 - roll_base/2 + yaw_base + motor1_bias;
  pwm3 = throttle_base + pitch_base/2 + roll_base/2 - yaw_base + motor3_bias;
  pwm0 = throttle_base - pitch_base/2 - roll_base/2 - yaw_base + motor0_bias;
  pwm2 = throttle_base - pitch_base/2 + roll_base/2 + yaw_base + motor2_bias;
#else
  pwm0 = throttle_base + pitch_base + yaw_base;
  pwm1 = throttle_base + roll_base - yaw_base;
  pwm2 = throttle_base - roll_base - yaw_base;
  pwm3 = throttle_base - pitch_base + yaw_base;
#endif
  /**
   * Boundary checks
   */
  if(pwm0 < min)
    pwm0 = min;
  else if(pwm0 > max)
    pwm0 = max;

  if(pwm1 < min)
    pwm1 = min;
  else if(pwm1 > max)
    pwm1 = max;

  if(pwm2 < min)
    pwm2 = min;
  else if(pwm2 > max)
    pwm2 = max;

  if(pwm3 < min)
    pwm3 = min;
  else if(pwm3 > max)
    pwm3 = max;

  PWMs[0] = pwm0;
  PWMs[1] = pwm1;
  PWMs[2] = pwm2;
  PWMs[3] = pwm3;
}
```

5.3.7 Motor Output

After the control signals have been mixed, the tweaked throttle, roll, pitch, and yaw values are ready to be translated to PWMs for the ESCs to consume and control the motors. Similar to how the main quad program was able to read PWM values from the RC recorder logic cores, the program writes out the final mixed values to PWM generator cores to translate clock cycle counts to PWM signals out to each of the four ESCs. Besides the main function of generating PWM values, the PWM generator core has a special safety built-in feature that will write minimum throttle values to the ESCs to stop all motor movement in case the software gets locked up which is another benefit of having an FPGA work in tandem with the microprocessor.

5.4 Ground Station Software Specification

With multiple sources of data to communicate between the system, the ground station software performs several tasks at once to accomodate the needs of the system. To handle all the operations, several programs were created to allow users to input commands to the quad, send camera data, and log data from the quad. A full diagram of the ground station's software flow can be seen in Figure 23 which shows the initialization of the system, the different processes it creates and the end result of plotting logged data when a flight is complete.

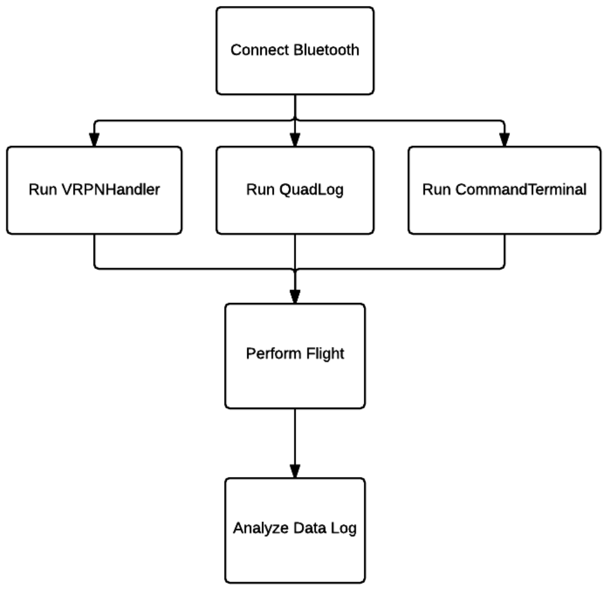

Figure 23: Diagram of entire ground station software flow

The data analytics for the system are done using MATLAB to parse data logs for both the entire system and produces visual plots of the data with respect to time. Figure 24 shows a flow of how the the logs are created from the ground stations program, the parsing through MATLAB, and the visualization of plots using the custom MATLAB GUI.

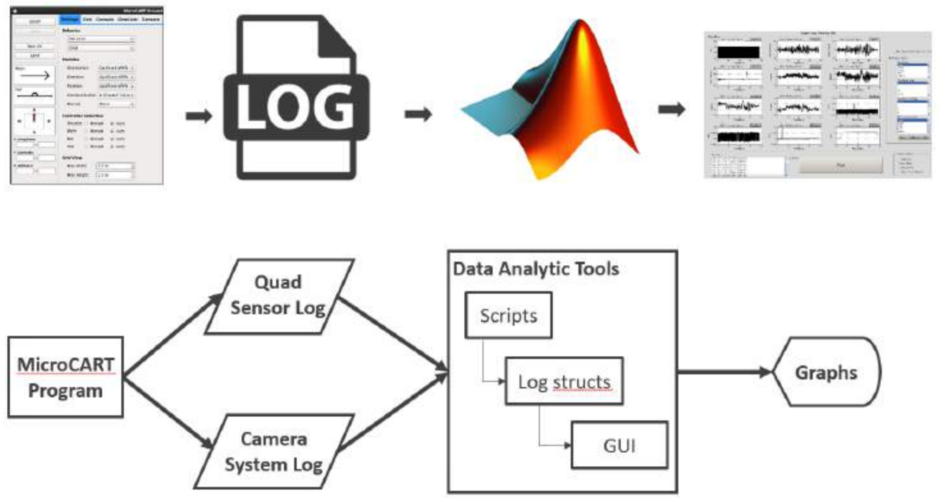

Figure 24: Data flow for ground station data analytics

To keep logging consistent across different platforms, a logging format was created and implemented in the ground station programs to making plotting easier and more robust. The format can be seen below with the details:

#Comments for users looking at the log are marked with the # sign
#MicroCART Log
#Samples Collected: 14875
#To specify column headings, the next line uses the % sign and the row after uses & to identify column units
%Time GyroX GyroY GyroZ AccelX ...
&sec dps dps dps G

5.4.1 Quadcopter Logging

The logging program for the quadcopter, known as quadlog, polls the Bluetooth module of the ground station and prints any received bytes to the screen. A simple diagram of the procedure can be seen in Figure 25. A benefit of using the program allows for users to pipe output to a file which can be used to be plotted later rather than go through other serial terminals. For a majority of the flight, the log is empty until the flight program has been terminated by the pilot in which the quad unloads its on-board data to the ground station.

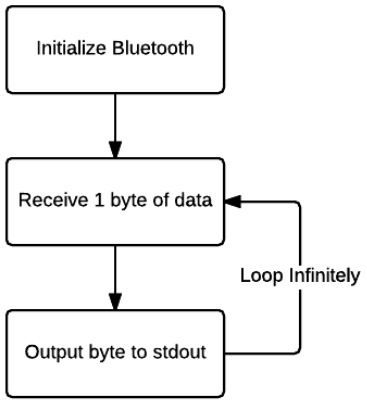

Figure 25: Diagram of quadcopter logging program

5.4.2 Command Terminal

Although not used when demonstrating quad flight, the command terminal allows for users to send commands to the quadcopter be it aerial commands, PID variables, or anything as long as the command is programmed on the quad to handled. As previously mentioned, the command terminal allowed the team to tune the PID coefficients during single or multi axis testing without having to stop the program which greatly sped up the process. A figure of the program flow can be seen in Figure 26.

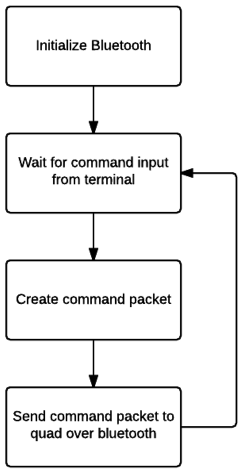
Initialize Bluetooth
Wait for command input from terminal
Create command packet
Send command packet to quad over bluetooth

Figure 26: Diagram of command terminal program

5.4.3 Camera System Data Pass-through

The last program used in the ground station arsenal is the camera system data pass-through program, known as VRPNHandler, that communicates update data packets to the quad through Bluetooth as well. The data it sends include an X, Y, and Z position coordinates, recently changed to North, East, and Altitude to reduce confusion, relative to the camera system and Pitch, Roll, and Yaw angles at rate of 50Hz. Although the camera system records at 100Hz, the quad only uses the Yaw angle to provide a global sense of direction so immediate updates are not required.

A figure of the process flow for the VRPNHandler program can be seen in Figure 27 with the split of the different threads in the program.

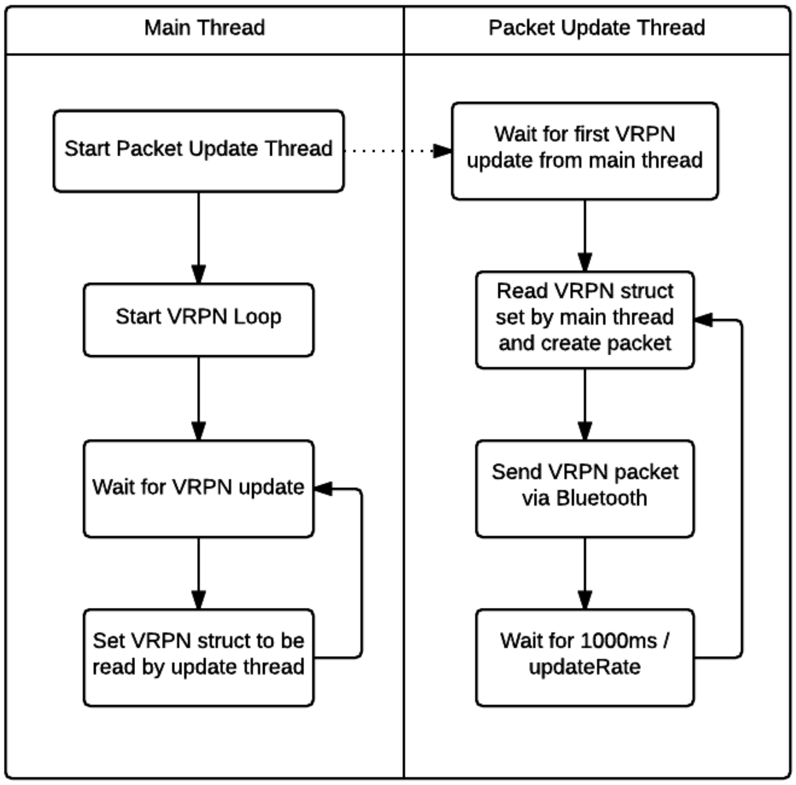
Main Thread
Packet Update Thread
Start Packet Update Thread
Start VRPN Loop
Wait for VRPN update
Set VRPN struct to be read by update thread
Wait for first VRPN update from main thread
Read VRPN struct set by main thread and create packet
Send VRPN packet via Bluetooth
Wait for 1000ms / updateRate

Figure 27: Diagram of camera system pass-through program (VRPNHandler)

Regarding the use of the camera, the quad is receiving all the camera information, but is only using the Yaw angle which may seem like extra overhead, but the team decided to send all data in case future projects would like to use utilize the camera system's X, Y, and Z positions.

5.4.4 MATLAB GUI

The MATLAB GUI is a custom tool created by the Robotic Agricultural Data Acquisition (RADA) team which worked closely with this project's team. The GUI consists of a fully capable log parser, data processor, and plotting tool packed into a single module. The GUI allows users to plot a log file that follows the specified logging format mentioned earlier and allows users to extract or change graphs and their properties for easier analysis. A screenshot of the GUI can be seen in Figure 28 which 12 different data plots drawn. By clicking the name of the subplots and selecting the type of plotting in the lower right hand corner, a user can bring up several larger figures depending on the options selected.

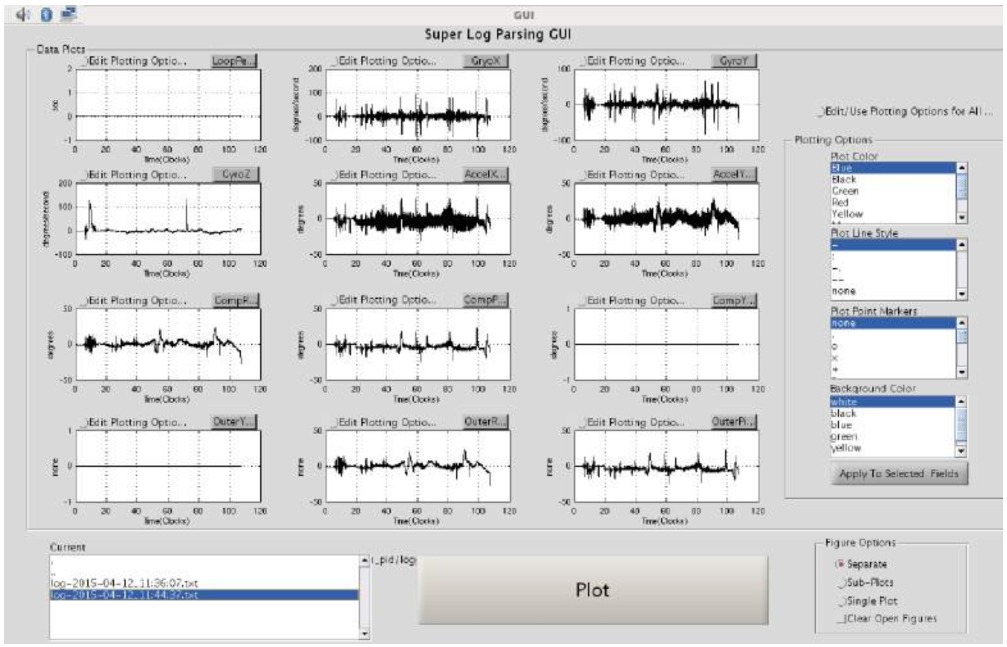

Figure 28: Screenshot of the MATLAB data analysis GUI

5.5 Test Specification

Testing the individual components and pieces of the system were conducted as each task was accomplished. For initial hardware and signal debugging, an oscilloscope was used to identify issues in the system like the custom hardware logic cores on the Zybo board. For example, proper I2C communication can be observed when probing the sensor board while its attached to the Zybo board.

For software, an agile approach was taken by creating a small working prototype and building more functionality upon the prototype. As more functionality is introduced and problems arise, a developer can roll back the source control to a working state.

As the system became elaborate, the following order of logically questions were asked while stepping through the hardware and software flow of the system:

- Is the quad's power supply connected?
- Is the Zybo board being programmed? Is the "Done" LED on?
- Is the main program on the ground station started?
- Are the proper PID constants used?
- Is the RC transmitter plugged in, set to the proper mode, and armed?
- Is the camera system software on and set for the quad?

5.5.1 PID Tuning

In regards to controls, heavy testing and manual tuning were done for the quad's flight dynamics through the PID controllers. To tune the inner PID coefficients for angular velocity, the quad was placed on an Educational Control Product (ECP) machine which had rotating plates and an encoder to read the speed of the rotating plate. The quad was placed on the machine for each axis rotation and the inner P coefficients were tuned for the quad to be responsive, but not unstable which is often a problem with having too much of a P coefficient. A plot of a well-tuned pitch PID can be seen in Figure 29 which shows the quad spinning at a constant rate and is given some disturbance. The quad is quick to respond and gets back to its setpoint without oscillating too much.

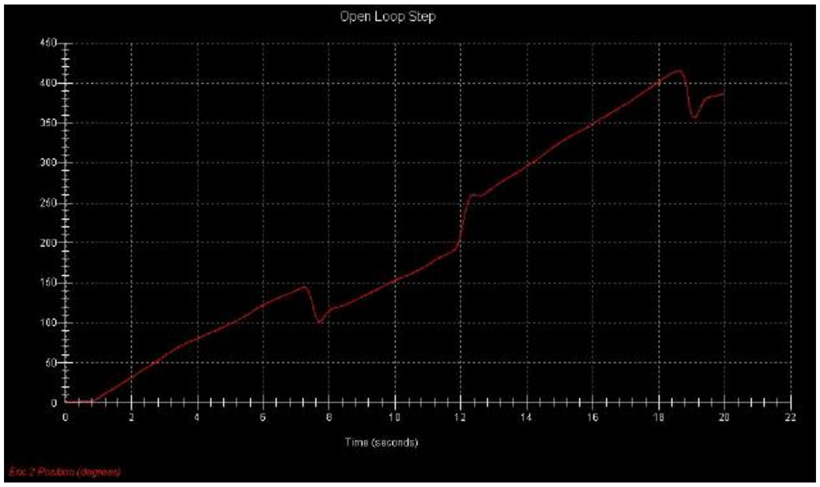

Figure 29: Plot of pitch-axis rotation during inner PID tuning

Once the inner PIDs were tuned, the quad was moved to a single axis wooden dowel in order to tune the outer PIDs based off the quad's angle. The P coefficients were slowly increased to get the quad stabilized when disturbed and observe if it would over shoot or compensate too quickly. After tuning the pitch and roll angle PIDs, since a yaw angle cannot be detected with a gyro and accelerometer alone, the quad was placed on a 3-axis stand that allowed the quad to move about all three axes in a limited range. A photo of the 3-axis stand can be seen in Figure 30 and shows the platform the quad was mounted on. The red lever near the bottom of the photo can be pulled down to drop the platform guard and allow the rotation platform to lean in any direction.

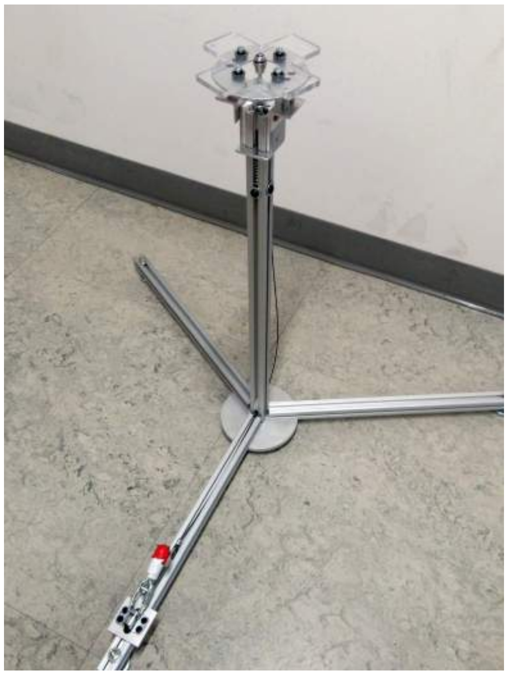

Figure 30: Picture of the 3-axis stand

While a majority of the PID tuning was done before moving to the 3-axis stand, the quad still needed to be tweaked since there were some unaccounted differences when movin to multiple axis freedom. Making the necessary adjustments, the quad was made stable as the team prepared for tethered manual flight. After moving to free flight, the quad was noticed to sway along the roll axis and required introducing a D coefficient to the PID controller. By adding the D coefficient, the controller dampens any oscillation, though a lot of consideration went into the decision of adding a derivative component since it can cause instability if too great. It should be noted that PID tuning occurred several times over the course of development due to changes in sensor filtering or propellers.

5.5.2 Software Testing

Software testing was done with a function and unit testing paradigm first. After ensuring the core functionality worked, the program was expanded to include additional features which were then tested functionally and as a system in total. While progressing toward final system integration of the camera system, ground station, and quad, the software modules were tested regularly in the system when performing flights. Although not ideal, when issues came up during development, defensive coding was done for some edge cases that were overlooked or forgotten during the design process. Luckily, the team did not run into many errors when integrating software components across the system.

5.5.3 Test results

A concern that surfaced early in the project's life was latency of communication devices. This concern was mostly warranted from the camera system which was found to have a 100ms delay. For Bluetooth, a team member tested roundtrip timings to find the Bluetooth had roughly 140Kbps transfer rate and a 25ms latency one-way from the ground station to the quad:

Bluetooth statistics:
Able to get consistent transfer speed around 140kbps
Transfer is reliable, 100% file integrity tested up to 1MB (57 second transfer)

Timings:
~50ms roundtrip latency from base station computer using C
~55ms roundtrip latency from base station computer using Java
~75ms roundtrip latency from Adam's laptop using Java

For the power boards to regulate voltage to the Zybo board and monitor motor batteries, non-linear voltage outputs were found for the boards and did not match their specifications. The only specification that was met was the detection of reverse polarity.

Throughout the tuning process plenty of quad plots were created. Besides the ECP machine plot shown earlier, Figure 31 shows the quad with a zero setpoint and tries to not correct itself to not have any angular velocity.

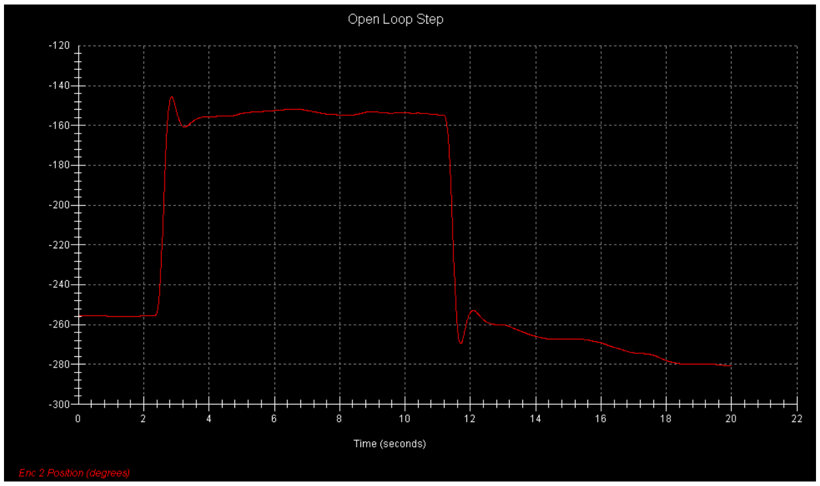

Figure 31: Plot of pitch-axis rotation of the quad with a zero velocity setpoint

5.6 Implementation Details

Throughout the development of the project several tasks were completed to progress the project along toward its goals. While the team was not able to choose many of the peripherals, the team did design the placement, materials, and connectors for the quadcopter and designed the logic to be configured on the FPGA as well as the power boards.

The first implementation tasks included mounting components to the chassis of the quad and implementing hardware and software configurations to get peripherals working. Some tasks went quicker than others. For example, to get the I2C controller working on the Zybo board took nearly a month due to some obscure hardware configuration setting that required a pull-up pin rather than the default pull-down. Afterwards, the sensor board was able to successfully read the data and get it in a usable format.

Progressing forward with other peripherals and hardware logic for the PWM recorder and generators done, the main quad software was created by integrating all the communication and peripherals into a single program. With a semi-functioning quadcopter, PID tuning started, though was quite unsuccessful the first time with a rushed approach and simple PID controller that did not account for several factors. Regrouping afterward, PID tuning was more successful by implemented the current nested PID approach thanks to a lot of discussion between team members and advisers with more control background suggestions.

Currently, the quad waits until the end of its flight program to offload its data to the ground station rather than during flight for two reasons. The first is a speed issue with the Bluetooth not having fast enough communication to send data to the ground station quickly while keeping a fast control loop. The second issue is the cross communication that becomes apparent when trying to send data to the quad while also receiving data from it and communication quickly locks up. So, the decision was made to wait until the very end of a flight to send data. There is still an issue, however. A user needs to terminate the VRPNHandler program on the ground station first before the quad pilot terminates the quad's flight program to get sufficient data. Otherwise, the cross communication issue rises again. A recommended fix to the problem is to add another communication module to the quad, or add a faster method of communication like Wi-Fi which would greatly speed up data communication and allow the quad to offload data quickly.

Carbon fiber blades are the current propeller choice for the quad, though there are several issues with them mostly being vibration while on tuning equipment. A recommended plan of action for the future is to investigate other propellers of the same size and pitch.

5.6.1 Hardware & Software Safety

During development of the new platform, several safety concerns were discovered and discussed that were handled over time. The first obstacle was the mounting of hardware components on the quad chassis. While the main goal was to have the different piece physically on the quad, a lot of thought and consideration went into the placement and material used for mounting the pieces on the quad as to not have pieces fall off or break easily when the quadcopter would fall as well as have easy accessibility in case something were to go wrong. Adjustments to the quad's body were done as needed to fit components like the sensor board and microprocessor on the center of the body and secured using appropriate bolts that wouldn't exceed the height of the motors, another concern. Overall, the placement of the components have worked well and hold up nicely even when the quad takes a large fall.

To ensure the safety of the quad and users around it, a kill switch was programmed into the PWM generator logic on the Zybo board to cut off all power to the motors when it reads the same values in its registers for more than one second. This can happen when the software becomes locked and doesn't update the PWM registers in the core. Two software approaches were created to cleanly cut power to the motors as well by either flipping the Gear switch on the top left of the RC transmitter or by placing the left stick of the transmitter to the bottom right. Of course, if the software is locked these two methods do not work. One last guard to protect users is if the RC transmitter is turned off while powering on the quad, then it will wait for an acceptable range before starting the program; otherwise, the quad could place some random values into the PWM generators and make the motors act sporadic.

5.6.2 Issues

The main issue the team is focused on is the safety of the quad system when the batteries' power levels are too low. As discussed earlier, the batteries at some time will hit a point where it cannot output the necessary voltage needed to operate the motors normally. At this point, the power to the motors will gradually decrease even when throttle is max and make the quad descend toward the ground. While this is better than the quad just turning off completely and falling, the team would like a way to know when this point is about to be reached and safely land before any diminish starts to occur. The power management PCB board for the motors would solve this problem as it can send an AC signal to the Zybo to be interpreted. Thus, the Zybo can monitor the battery and start landing if needed. However, the power management PCB boards were unsuccessful and did not have many of the features required of them except for the reverse polarity detection even through several design iterations.

A major issue that arose several times throughout development was the occurrence when software would lock up due to several factors and cause issues both for the hardware and users. This reoccurring issue was mainly attributed to the connections from the Zybo board and its peripherals. If a peripheral would become disconnected during a flight, the software would lock and make it uncontrollable from a user's point of view. The only way to stop the quad was to unplug the batteries from the board. When the quad motors were still spinning at high rates, the situation clearly became dangerous. Several steps were taken to minimize this happening including the hardware logic kill switch inside the PWM generators to stop the quad after a few seconds of having repeated values. To combat this in software, several measures were taken including attempting to switch to an interrupt based approach for data retrieval guards in the software. Most of the software guards did not prove helpful for the situation either due to incorrect interrupt handling or not handling all lock up locations.

Another issue that came fairly late in development was the issue of the carbon fiber blades. The blades that were ordered had several problems themselves including non-centered axes about center on the propeller, unbalanced wings due to one side being heavier than the other, and the center mounting hole of the propeller being too small for the motors even though they specified they are made for the DJI make and model. When the carbon fiber blades were used on the quad, severe vibration while on the tuning equipment, and little work could be done to further tune the quad. Several plots can be seen in Figures 32 and 33 which show the vibration especially in the gyroscope plots.

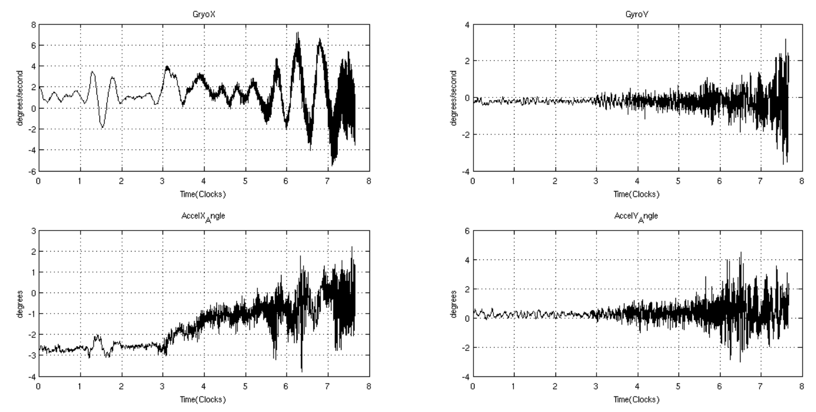

Figure 32: Plots of sensor board readings from quad with original DJI propellers

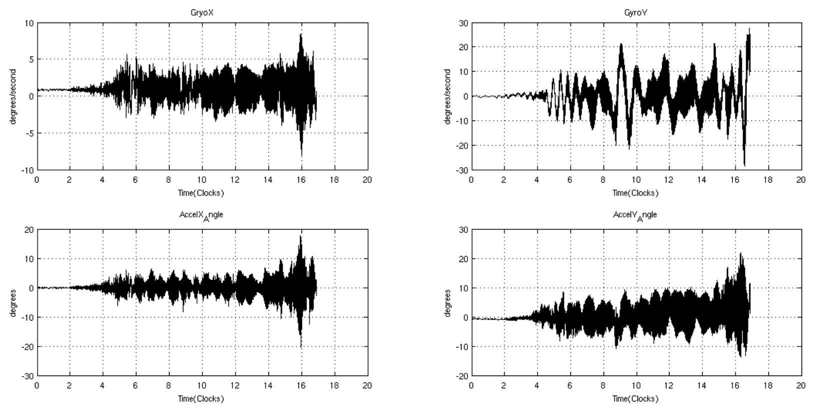

Figure 33: Plots of sensor board readings from quad with carbon fiber propellers

6 Deliverables

The team plans to deliver the new working quadcopter and system to the client at the end of their project term. The final system consists of the quadcopter with its mounted board, sensors, and peripherals as well as all software and configuration settings for the microprocessor. Additionally, the team plans to relinquish all source code for the project which include the quad and ground station software, experimental peripheral code, data logging and analytic programs. To accompany the many aspects of the project, there are several flavors of documentation that are present within the repository. The first set of documents are high-level "dry-run" documents that explain the project, provide how-to tutorials like developing logic and programming the Zybo board, charging batteries, or operating the quad. The second set of documents are task specific and revolve around each of the components that were created to progress the project explaining what the task is, why it's important, and how to run it, if applicable.

All source code and documents are located in the team's GitLab repository hosted by Iowa State. The repository is broken into easy to distinguish directories: docs, tasks, Xilinx_Tools, and misc. The "docs" directory contains the projects class documents like the design doc, presentation, poster, project plan, and this document. The tasks directory contains all source code for the quad and ground station as mentioned earlier. The two nontrivial directories are "Xilinx_Tools" which contain guides and scripts to help developers start making programs for the Zybo board, and "misc" which has miscellaneous code or tools that were researched during the project's development. Besides the main repository, a Google Drive folder is shared with the client which holds the team's weekly reports, pictures, videos, and informal plans for the project. For future teams, these two sources will be great to review and study in order to start development quickly and efficiently.

7 Conclusion

The MicroCART team is focused on improving the old quadcopter system in the Distributed Sensing and Decision Making Lab. The new project includes replacing the old quadcopter with a brand new on with a stronger frame and a development board that is reconfigurable. The new system shifts more of the system processing to on-board the quad. Additionally, new data analytic tools and processes were created to plot data through MATLAB and the ground station to allow for more robust visualization.

8 Closing Material

8.1 Client

Distributed Sensing and Decision Making Laboratory, Iowa State University
Contact: Dr. Phillip Jones III

8.2 Team Info

Members

Joe Boldrey	(Electrical Engineering)	Quad Structure Lead
Ravi Nagaraju	(Electrical Engineering)	Power Management Lead
Tyler Kurtz	(Electrical Engineering)	Peripheral Logic Lead
Jacob Rigdon	(Computer Engineering)	Communication Lead
Matt Vitale	(Computer Engineering)	Documentation Lead
Adam Campbell	(Software Engineering)	Ground Station Lead
Matt Post	(Electrical Engineering)	Controls Lead
Paul Gerver	(Computer Engineering)	Zybo Board Lead

Advisors

Dr. Phillip Jones III (Assistant Professor, Department of Electrical and Computer Engineering)
Dr. Nicola Elia (Professor, Department of Electrical and Computer Engineering)

Acknowledgments

Paul Uhing (M.S. Candidate)
Matt Rich (Ph.D. Candidate)
RADA Project

A Operations Manual

1. **Secure quad components**
   (a) Make sure propellers are on the motors tightly
   (b) Make sure connections are correct for sensor board, RC receiver, and Bluetooth is plugged into correct slot
   (c) Make sure IR constellation is on quad and in proper direction
2. **Tether quad to floor**
   (a) Place quad in center of camera system
3. **Start camera system**
   (a) Log in to co3050-07 as microcart user
   (b) Open NaturalPoint Tracking Tools program
   (c) Load (MicroCART) Configurations
   (d) Select UAV constellation in program
4. **Prepare Zybo board on quad**
   (a) Log in to co3050-12 or computer with repository
   (b) Place microSD card into adapter and plug into computer
   (c) Navigate to repo/tasks/Quad_PID/sdcard
   (d) Copy April27_Demo.bin onto top level of microSD card
   (e) Rename April27_Demo.bin to BOOT.bin
   (f) Eject microSD card and place into Zybo board
   (g) Ensure blue jumper is set to SD on board
5. **Prepare RC transmitter**
   (a) Place left stick all the way down
   (b) Turn on transmitter by sliding middle button to on
   (c) Ensure current setting reads GAUI
6. **Power and start quad**
   (a) Secure 3-cell battery to velcro strip
   (b) Secure 2-cell battery or 4AA battery pack to quad's other base platform
   (c) Plug in 3-cell battery to main quad power cord and battery pack to Zybo board
   (d) Turn on Zybo board
7. **Start ground station**
   (a) Log in to co3050-09 computer
   (b) Open terminal and navigate to repo/tasks/ground_station_pid/terminal
   (c) Run connect_zybo_bt with su permissions

   (d) Start quadlog program and pipe program to desired output file
   (e) Start vrpnhandler program

8. **Fly quad**
   (a) Use RC transmitter to give quad throttle, roll, pitch, and yaw commands
   (b) When done, terminate quad program by flipping top Gear switch from 1 to 0 or put left stick down and to the right

9. **End flight, start log transfer**
   (a) Terminate vrpnhandler program
   (b) Flip top left Gear switch from 1 to 0 on RC transmitter, or put left stick down and the right
   (c) Wait for quad to finish offloading data to quadlog program
   (d) When done, terminate quadlog program

10. **Visualize data**
    (a) Start MATLAB on ground station
    (b) Navigate to repo/tasks/ground_station_pid/logging/Camera_System
    (c) Run "GUI" command in MATLAB
    (d) In GUI, navigate to terminal directory and open created log

B Alternative Designs

The original plans for the project were to have a new quadcopter and remove several dependencies including the camera system environment which is best represented by Figure 34. This mindset would allow the quad to be moved outside the laboratory environment into the outdoors.

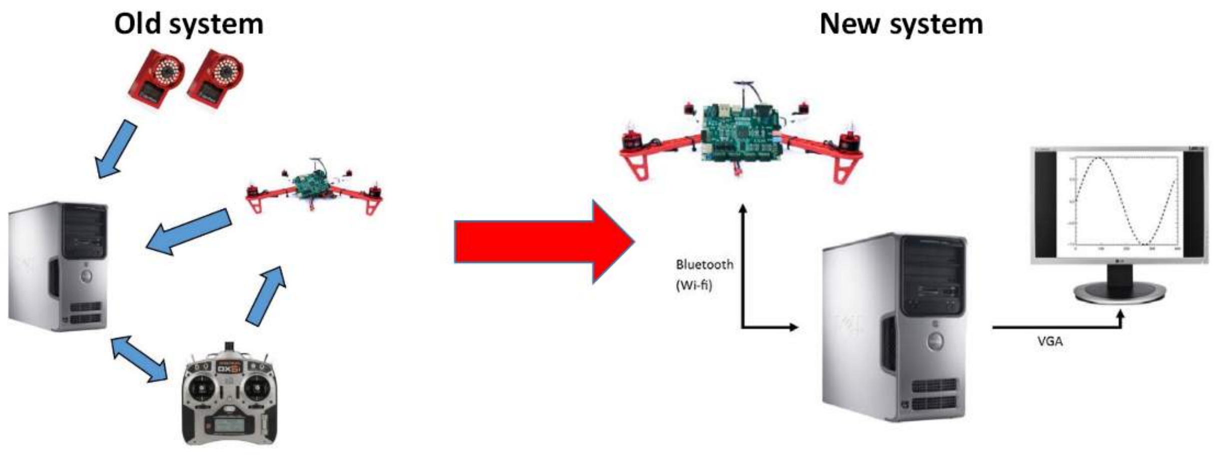

Figure 34: Old project concept diagram

While the new quadcopter was successful and did remove some dependency on the ground station, more work needs to be done to further achieve these goals. Considering that the camera system is used to keep the quad pointed in a cardinal direction, this was almost a reality. The idea was to utilize the magnetometer to allow the quad to identify North and use other peripherals to fly outside. However, like many things, a major obstacle appeared in the form of magnetic disturbance from being inside Coover Hall.

Another design that changed over the course of development was the system's data logging. Starting with the sporadic old systems logs and little documentation, the team worked to comprehend the different log files and set to work on creating MATLAB scripts to visualize the data. The original work flow for the data logging and analytics flow can be seen in Figure 35.

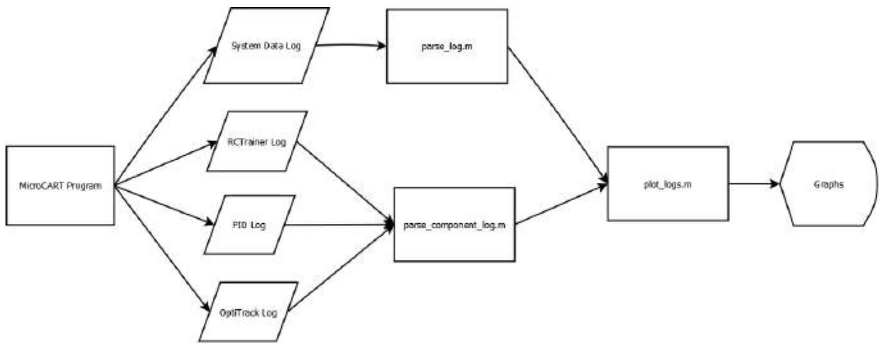

Figure 35: Old data analytics diagram

Considering that all the logs except for the camera system logs did not have headings to identify what columns of numbers represented, the team set to work on revamping the system setting up proper headings and documentation to inform users what the data was. While this was done in the project, the finished MATLAB tool was created by a member of the sister project: Robotic Agriculture Data Acquisition (RADA) with the teams initial scripts as a starting basis.

For many of these goals, the team is leaving the next team with plenty of documentation to learn the system, understand how it works, and potential paths to improve or take the system next. The team hopes next years team will continue to add peripherals to the quad to allow it to fly outdoors.

C Considerations, Lessons Learned

The first remark for this project is the amount of team members working on the project. While the project is certainly large enough to warrant a large team, communications challenges certainly arose throughout the process not only for team members, but also clients and lab personnel. There were many email chains and people may have missed some details along the way pertaining to the lab equipment or process.

Regarding the design and implementation process for the project, the group worked hard to achieve tasks in great leaps and bounds over a short period of time which would cause errors later on in development. While components were being integrated quickly and tasks were completed in bare minimum fashion, there were many times that documentation would have greatly beneficial. To counter this, the team took several breaks in development to update documentation. However, documentation alone could not fix the issues of quick implementation and little robustness, and required the team to return to previous tasks and refine code, equations, or designs to make the system better rather than just runnable.

D Code

The following code is the main software program run on the quadcopter for flight demonstrations. Note that the code contains only the core of the program does not include the entire source and functions used in the code.

```
#include <stdio.h>
#include "platform.h"
#include <xparameters.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "util.h"
#include "xtime_l.h"
#include "uart.h"
#include "controllers.h"
#include "iic_mpu9150_utils.h"
#include "PID.h"
#include "stringBuilder.h"
#include "terminal.h"
#include "commandproc.h"
#include "gam.h"
#include "quadposition.h"
#include "sleep.h"

/**
 * Define Constants
 */
#define PI_OVER_180 0.017453278


/**
 * Global Variables
 */
logging logs = { };

/**
 * Function Prototypes
 */
int initializeAllComponents();  // Initializes all required components
void setPIDCoeff(PID_t* p, float pValue, float iValue, float dValue); // Sets
    the given PID's coefficients


int main() {

        int i, rc;
        int mixer[5];
        int pwms[4];
        double compX = 0, compY = 0;    // Complementary Roll and Pitch angles
        float time_stamp = 0;
        gam_t gam;                 // GAM struct for gyro, accelerometer, and
            magnetometer data

        XTime before = 0.0, after = 0.0;
        int firstLoop = 1;

        stringBuilder_t* sb = stringBuilder_create();
        int started = 0;
        int loop_counter = 0;


        // Structures to hold the current and desired quad position &
            orientation
        quadPosition_t currentQuadPosition = { };
        quadPosition_t desiredQuadPosition = { };

        currentQuadPosition.yaw = -1.58;

        char buf[1000] = { };

        PID_t rollCFPID = { }, pitchCFPID = { }, yawCFPID = { }; // External
            Comp filter PIDs
        PID_t rollGPID = { }, pitchGPID = { }, yawGPID = { };  // Internal Gyro
             PIDs

        // Initialize all required components:
        // Uart, PWM receiver/generator, I2C, Sensor Board
        rc = initializeAllComponents();
        if (rc == -1) {
                printf(buf, "ERROR (main): Problem initializing...Goodbye\r\n")
                    ;
                return rc;
        }

        // DO NOT CHANGE THESE NUMBERS: PID Coefficieints
        // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        // Outer PID coefficients (Angle)
        setPIDCoeff(&pitchCFPID, -7,0, -0.45);
        setPIDCoeff(&rollCFPID, -6.0, 0, -0.4);
        setPIDCoeff(&yawCFPID, 60, 0, 0);

        // Inner PID coefficients (Angular Velocity)
        setPIDCoeff(&pitchGPID, -96,0,0);
        setPIDCoeff(&rollGPID, -96,0,0); // inner = -96 4/23
        setPIDCoeff(&yawGPID, 470, 0, 0);



        // Protection Loops:
        // 1) Wait until RC Transmitter is turned on
        // 2) Wait until Quad has received data packet from ground station
        while(mixer[THROTTLE] < 50000){
                read_rec_all(mixer);
        }
        while(!tryReceivePacket(sb, 0));




        // Main Flying Loop:
        // 1.Read in RC Receiver values
        // 2.Collect data from sensor board
        // 3.Calculate PID corrections (Outer and Inner)
        // 4.Output PWMs to motors
        // 5.Check if command from base station
        //    a.Increase parameter based off command
        // 6.Update log
        while (1) {

                // Calculate loop period
                float LOOP_TIME = (after - before) / (float) COUNTS_PER_SECOND;
                XTime_GetTime(&before);

                // Read in values from RC Receiver
                read_rec_all(mixer);

                // Cheap way for user to disarm the qaud. Disarm by: have
                    throttle/yaw stick down and to the right
                if (started && mixer[THROTTLE] < 125000 && mixer[YAW] > 160000)
                     {
                        break;
                }

                // Read gyro and accelerometer data
                // TODO: Replace camera system heading for magnetometer
                get_gam_reading(&gam);
                gam.heading = 0;

                // If it's the first loop, initialize our angles and run again
                if (firstLoop) {
                        XTime_GetTime(&after);
                        firstLoop = 0;
                        // Sets the first iteration to be at the accelerometer
                            value since gyro initializes to {0,0,0} regardless
                            of orientation
                        compX = gam.accel_xAngle;
                        compY = gam.accel_yAngle;
                        continue;
                }


                // Perform gimbal equations to adjust rotational velocities for
                     multidimensional rotation
                // Please see page 74 of the green book in Coover 3050
                double sin_phi = sin(compX * PI_OVER_180);
                double cos_phi = cos(compX * PI_OVER_180);
                double tan_theta = tan(compY * PI_OVER_180);
                double sec_theta = 1/cos(compY * PI_OVER_180);

                double n_gx = gam.gyro_xVel + (gam.gyro_yVel*sin_phi*tan_theta)
                     + (gam.gyro_zVel*cos_phi*tan_theta);
                double n_gy = (gam.gyro_yVel*cos_phi) - (gam.gyro_zVel*sin_phi)
                    ;
                double n_gz = (gam.gyro_yVel*sin_phi*sec_theta)+ (gam.gyro_zVel
                    *cos_phi*sec_theta);


                // Compute quad's X and Y angles using complimentary filter
                // Uses 98\% of quad's current angle + integrated gyro velocity
                     over time
                // Uses 2\% of quad's angle relative to accelerometer reading
                compX = .98 * (compX + n_gx * LOOP_TIME)
                               + .02 * gam.accel_xAngle;

                compY = .98 * (compY + n_gy* LOOP_TIME)
                               + .02 * gam.accel_yAngle;



                /*
                 *                                      Outer loop
                 * Calculates current orientation, and outputs
                 * a pitch, roll, or yaw velocity to an inner loop PID
                 */

                pitchCFPID.sensor = compX;
                pitchCFPID.setpoint = 0;

                rollCFPID.sensor = compY;
                rollCFPID.setpoint = 0;

                // Camera system expects quad to be facing north at -1.58
                    radians
                yawCFPID.sensor = currentQuadPosition.yaw;
                yawCFPID.setpoint = -1.58;

                // Compute outer PID corrections and save them into log struct
                logs.outer_pitch_PID = pid_computation(&pitchCFPID);
                logs.outer_roll_PID = pid_computation(&rollCFPID);
                logs.outer_yaw_PID = pid_computation(&yawCFPID);


                /*
                 *                              Inner Loop
                 * Takes the desired angular velocity from the outer loop,
                 * and uses a PID with the current angular velocity
                 */

                pitchGPID.sensor = n_gx;
                pitchGPID.setpoint = pitchCFPID.pid_correction;

                rollGPID.sensor = n_gy;
                rollGPID.setpoint = rollCFPID.pid_correction;

                yawGPID.sensor = n_gz;
                yawGPID.setpoint = yawCFPID.pid_correction; // TODO : Implement
                     with magnetometer

                // Compute inner PID corrections and save them into log struct
                logs.inner_pitch_PID = pid_computation(&pitchGPID);
                logs.inner_roll_PID = pid_computation(&rollGPID);
                logs.inner_yaw_PID = pid_computation(&yawGPID);


                /*
                 * Motor corrections
                 * Adjust motor power based off RC values and PID corrections
                 */
                mixer[PITCH] += pitchGPID.pid_correction;
                mixer[ROLL] += rollGPID.pid_correction;
                mixer[YAW] += yawGPID.pid_correction;
```

```
                // Convert from the aero inputs to PWMS
                Aero_to_PWMS(pwms, mixer);

                // Write the PWM values to hardware registers
                for (i = 0; i < 4; i++)
                        pwm_write_channel(pwms[i], i);


                // Listen on bluetooth and if there's a packet,
                // then receive the packet and process it.
                // Use the stringbuilder to keep track of data received so far
                int echo = 0;
                int hasPacket = tryReceivePacket(sb, echo);
                if (hasPacket) {
                        processPacket(sb->buf, &currentQuadPosition, &
                            desiredQuadPosition, &rollCFPID);
                        sb->clear(sb);
                }

                //Finish logging
                logs.time_stamp = time_stamp;
                logs.time_slice = LOOP_TIME;

                logs.gam = gam;

                logs.pitch_angle = compX;
                logs.roll_angle = compY;

                logs.motors[0] = pwms[0];
                logs.motors[1] = pwms[1];
                logs.motors[2] = pwms[2];
                logs.motors[3] = pwms[3];

                // Update quad's log every 10 iterations
                // Loop period is ~450us
                if (loop_counter == 10) {
                        loop_counter = 0;
                        updateLog();
                } else {
                        ++loop_counter;
                }

                XTime_GetTime(&after);
                time_stamp += LOOP_TIME;

        } //end while(1)
        stringBuilder_free(sb);

        // Turn off motors
        pwm_kill();

        // Have quad print logging info to ground station
        printLogging();

        return 0;
} //main


// Initializes all hardware controllers on Zybo board
```

```
int initializeAllComponents() {

        int i2c_rc, mpu_rc;

        init_platform();

        // Initialize UART0 (Bluetooth)
        uart0_init(XPAR_PS7_UART_0_DEVICE_ID, 921600);
        uart0_clearFIFOs();

        // Initialize I2C controller and start the sensor board
        i2c_rc = initI2C0();
        mpu_rc = startMPU9150();

        if (i2c_rc == -1 || mpu_rc == -1) {
                return -1;
        }

        // Initialize PWM Recorders and Motor outputs
        pwm_init();

        printf("\nStarting\r\n");
        return 0;
}

// Utility functio to set the P, I, and D coefficients for a given PID struct
void setPIDCoeff(PID_t* p, float pValue, float iValue, float dValue) {

        p->Kp = pValue;
        p->Ki = iValue;
        p->Kd = dValue;

}
```