APRIL 28, 2015

MAY 15-27

ROBOTIC AGRICULTURAL DATA ACQUISITION

PROJECT REPORT

ADVISERS

PHILIP JONES

NICOLA ELIA

PAUL UHING

MATT RICH

TEAM MEMBERS

DYLAN GRANSEE

ROBERT LARSEN

ALBERTO DI MARTINO

IAN MCINERNEY

AARON PEDERSON

ROHIT ZAMBRE

FENGXING ZHU

OUTLINE

TERMINOLOGY

DEFINITIONS

ACTUATOR

A motor or some means of providing force into a system.

GIMBAL

A pivoted support that allows the rotation of an object about a single axis. In the context of this project, a gimbal is a mechanical structure that allows the sensing and actuation of the movement of the instrumentation arm.

INERTIAL MEASUREMENT SYSTEM

A sensor system consisting of a gyroscope, accelerometer and magnetometer to provide data about the current position of a physical system.

KALMAN FILTER/SENSOR FUSION

An algorithm to combine current sensor measurements (with sensor noise) with past system states to create an estimate of the current system state.

MIXING MATRIX

An algorithm to combine a set of controller outputs (pitch, roll, yaw, altitude, etc.) into commands to send to individual actuators.

PID CONTROLLER

Proportional Integral Derivative Controller; a mechanism used to calculate the system input based upon the error between the target position and the measured position.

- *Proportional (P)*: Setting that controls how quickly the system will move from the current position to the target position using the current error.
- *Integral (I)*: Setting that uses the past system error to force the system steady-state error to zero.
- *Derivative (D)*: Setting that uses the predicted system error to limit the rate-of-change of the system error.

MICROCART

MicoCART is an ongoing senior design project at Iowa State University that primarily focuses its work on building an aerial, controllable vehicle. For the last two years, the MicroCART team has built quad-copters.

ACRONYMNS

ESC

Electronic Speed Controller

FPGA

Field-programmable gate array

GUI

Graphical User interface

MICROCART

Microcontroller Controlled Aerial Robotic Team

PCB

Printed Circuit Board

VHDL

VHSIC Hardware Description Language

1. INTRODUCTION

While conducting agricultural experiments, researchers must take a variety of measurements of every research specimen. Currently this process is done manually by research technicians walking the fields, which requires extra personnel, time and money for the project. The clients of the RADA project, Dr. Nicola Elia and Dr. Philip Jones, wish to develop a system that could automate these agricultural measurements using a robotic platform. An automated system would reduce the personnel required and the time spent by researchers gathering information about their specimens.

The final implementation of the RADA project will be able to reliably traverse through the rough terrain of experimental fields. Considering the current resources and the complexity of the final implementation, the objective of the May 15-27 senior design group is to implement a prototype of a robotic system that could feasibly traverse in a controlled environment. The prototype system consists of an instrumentation arm connected to a ground robot at a single pivot point. Four propellers atop the instrumentation arm are responsible to balance the arm about the pivot. The environment consists of a camera system that tracks the robot and also serves as a feedback for the robot.

The following sections outline the roles and functionalities of the different components responsible for controlling the robotic system. Figure 1 portrays a conceptual sketch of the system.

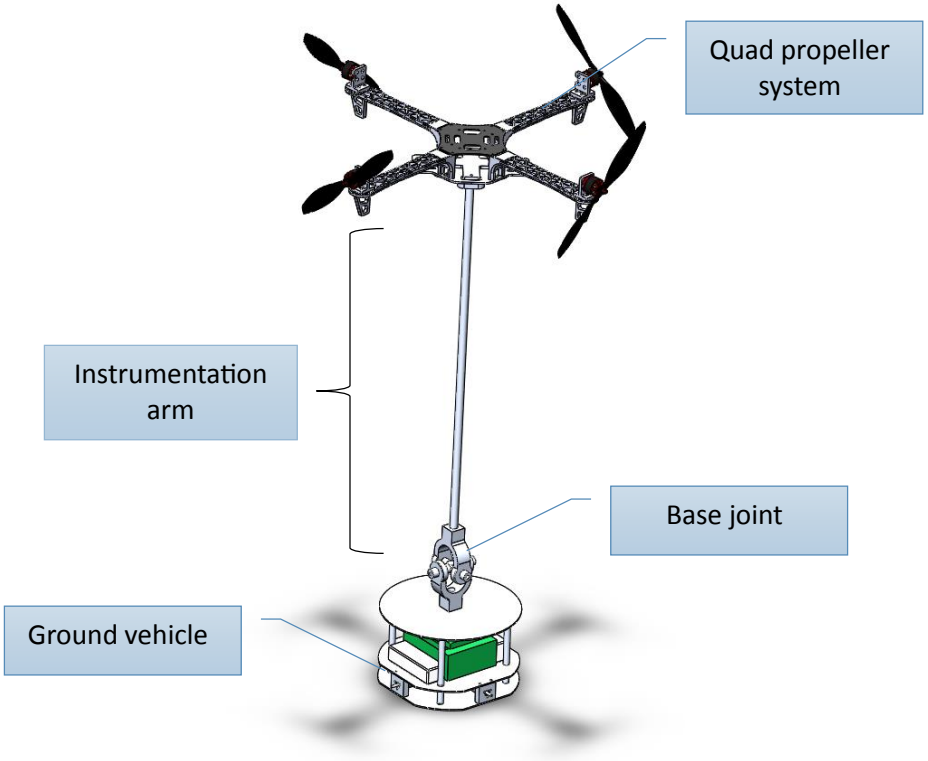

FIGURE 1: CONCEPTUAL SYSTEM

2. SYSTEM REQUIREMENTS

Our project has two distinct sets of system requirements. The first set consists of the requirements provided by the clients for their ideal final system design. These requirements provided their ideals for what a final measurement system would look like. The second set of requirements were derived from the initial requirements, and then approved by the client. These are the requirements that our team used when creating the prototype system.

2.1. FINAL SYSTEM REQUIREMENTS

1. The system must be robust and able to traverse the terrain present in a corn field
2. The instrumentation arm must remain vertical while traversing terrain
3. The instrumentation arm must be balanced by actuators at the top of the arm and at the base of the art
4. Will get position feedback through sensors in the base of the arm and an IMU at the top of the arm
5. The instrumentation arm must be taller than the corn height (approximately 12 feet)
6. System must log instrumentation arm stability data for later analysis
7. Instrumentation arm stabilizing controller must be easily upgrade /changeable

3.2. PROTOTYPE SYSTEM REQUIREMENTS

1. System must log instrumentation arm stability data for later analysis.
2. Instrumentation arm stabilizing controller must be easily upgradable/changeable.
3. The instrumentation arm must remain vertical while traversing obstacles.
4. The instrumentation arm will be approximately 6 feet tall.
    - This is a modification of client requirement 5. It was modified due to the space constraints present in the lab that our team used for development.
5. The instrumentation arm will be balanced from the top actuators.
    - This is a modification of client requirement 3. The bottom actuator requires a custom designed base-joint.
6. The system will receive position input from a high speed camera system in the development lab.
    - This is a modification of client requirement 4. This allows for quick prototyping of the instrumentation arm system while there is no suitable base-joint.

Overall, our project was focused on designing a prototype system in the clients' laboratory to demonstrate the feasibility of the concept of a robotic plant measurement system.

3. SYSTEM OVERVIEW

Figure 2 depicts a high-level overview of the robotic system. The ground robot hosts all of the hardware and software that provide the computational and networking needs of the system. The camera system relays data regarding the system's state to the ground robot. This data serves as feedback for the control system responsible for the stability of the instrumentation arm. The four propellers atop the arm provide the actuation required for the control system. The base joint allows the instrumentation arm to rotate about the roll and pitch axes. During experiments, data is logged continuously; logged data can be analyzed by the Data Analysis Tool post-experiment.

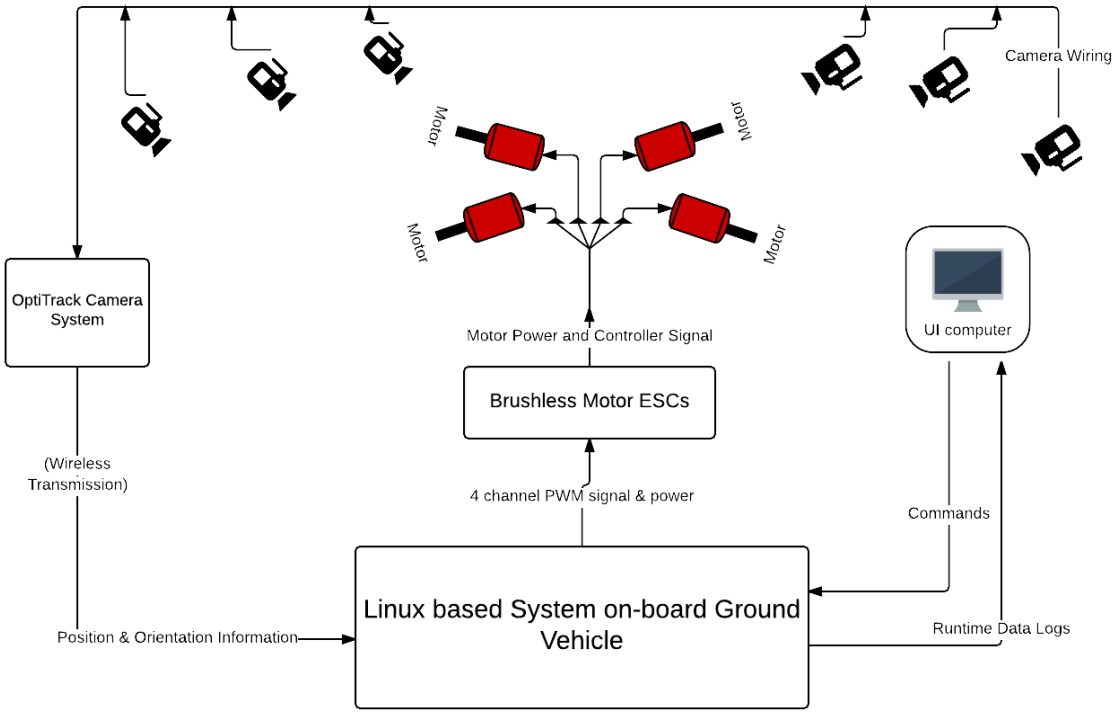

FIGURE 2: SYSTEM BLOCK DIAGRAM

3.1. GROUND ROBOT

Our clients provided us with a ground robot chassis constructed by a previous senior design team in 2010.

3.1.1. HARDWARE

This robot chassis contains the following hardware components:

- ETX-270 Pluto Computer running a 1.6GHz Intel Atom processor
- Mesa 4i68 motor controller containing a Xilinx Spartan III FPGA
- Faulhaber 2232 U DC-Motors and Faulhaber Series IE2-16 Encoders
- Adapter board
    - Pololu/Freescale MC33926 Motor Drivers
    - Power regulators

3.1.2. ROBOT CHASSIS

The ground Robot chassis contains the following components:

- Circular Aluminum plate CNC machined to have appropriate holes and shapes to accommodate wheels, PCB mount and the base of the joint
- Mounts to accommodate Eris' wheels
- A raised platform used as a base for the joint and to protect the PCB systems underneath
- Threaded rods secured by washers sandwiched by nuts around each aluminum plate, securing all components tightly

3.1.3. BASE JOINT

A special joint which only allows for rotation in two directions. Also called a universal joint. It allows bending but not twisting.

3.1.4. OPERATING SYSTEM

This robot system runs a Linux operating system compiled by the 2010 senior design team. This Linux install provides PC/104 drivers for the Mesa 4i68 board, and contains a wireless network interface to communicate to the main lab network.

4.1.5 SOFTWARE

The software to control the robot movement and balance the instrumentation arm was adapted from the previous senior design team's application. It is an application written in C++ that runs on Linux OS.

3.2. PROPELLER SYSTEM & INSTRUMENTATION ARM

This system includes the rigid metal pole which is supporting the propeller system. It will serve as the device holding sensing equipment in the end goal product.

3.2.1 INSTRUMENTATION ARM

The instrumentation arm is the pole that is being balanced in the system. It would contain any measurement devices in the final system, such as cameras. The arm consists of the following components:

- Aluminum 6’ long pole
- Propeller System

3.2.2. PROPELLER SYSTEM

The propeller system is a quad frame with some custom part additions. It consists of the following components:

- DJI F450 frame
- DJI 30A ESC with BLHeli firmware
- DJI 920kv motors
- DJI 1038 10” propellers

3.3. DATA ANALYSIS TOOL

The team designed and developed a Data Analysis Tool to parse and analyze logged experimental data. The tool was developed so that it can be used by various future senior design teams. The tool allows logging of data and generation of plots based on user-configurable plotting options. Additionally, the helper functions of the tool allow the user to generate custom plots. The tool has two types of interfaces:

- MATLAB command-line interface
- MATLAB GUI

The command-line interface requires the users to have prior experience with MATLAB. This interface allows the users to play with data contained in MATLAB structs. Furthermore, the user can use MATLAB helper functions to generate additional required or custom plots.

The GUI is meant for users that are not familiar with MATLAB. The GUI offers extensive, simple and easy-to-use plotting configuration options.

3.4. SYSTEM BLOCK DIAGRAM

The following diagram represents the interaction between the different components enlisted above.

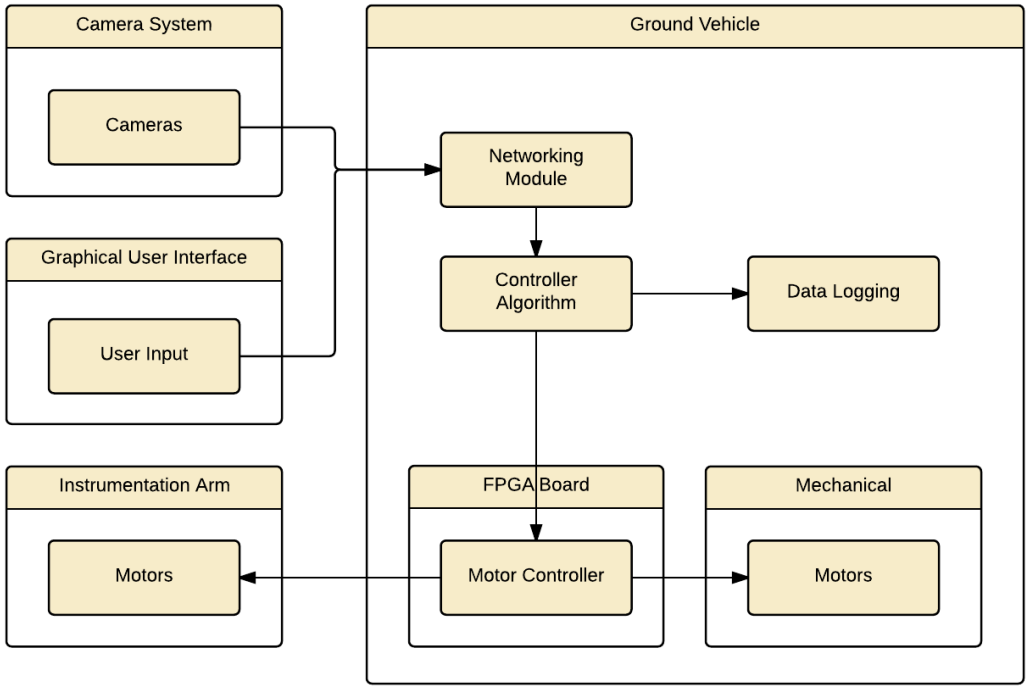

FIGURE 3: SYSTEM DATA FLOW

4. IMPLEMENTATION DETAILS

4.1. GROUND ROBOT

Our clients provided us with a ground robot chassis constructed by a previous senior design team in 2010.

4.1.1. HARDWARE

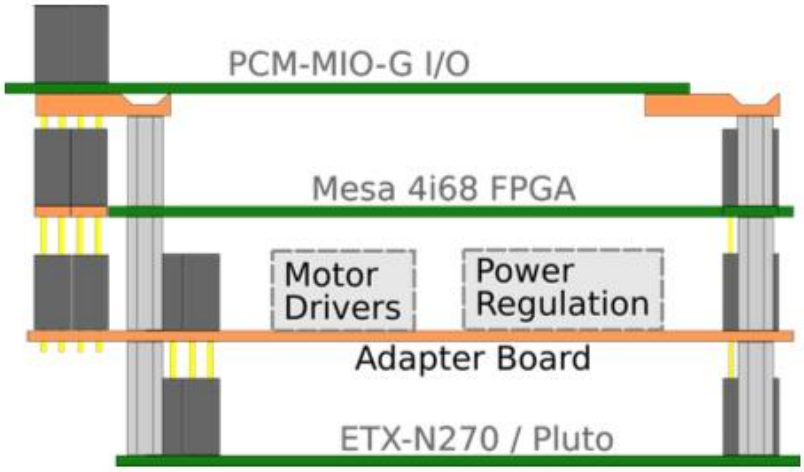

FIGURE 4: BOARD STACK ON THE GROUND ROBOT

4.1.1.1. PLUTO PC BOARD

The Eris platform uses a 1.6GHz Intel Atom processor and 1GB of DDR2 memory on an ETX 3.0 motherboard. This board runs the Linux operating system and hosts the control software for the system.

4.1.1.2. MESA 4I68 FPGA BOARD

The 4i68 I/O board is connected to the PCI portion of the PC/104+ bus. The board has a Xilinx Spartan 3 FPGA programmed as a motor controller. The robot currently uses 20 of 72 pins on the board leaving room for future expansion.

VHDL Modifications

A previous senior design team made VHDL modifications for use with driving the robot our team was using. These modifications were lost, and all that remained was the programming file located on the ground robot and some description of their modifications from a design document. From this description the modifications had to be reverse engineered so that the existing code base for driving the robot could be used while also controlling the instrumentation arm. This consisted of added a generate block to the softDMC module running on the FPGA that changes the output from one PWM and a direction signal to two one PWM output over two different pins, one at a time, based on what the direction signal is.

Once this system was reverse engineered modifications could be made to the VHDL in order to add the control for the instrumentation arm. This was done by adding a new PWM generator that outputs over an unused PWM signal. The arm motors require a 50-450 Hz PWM signal, while the wheels for the robot use a 28 kHz signal. The softDMC module only allows for one global PWM signal to be used by writing to a pre-scale and post-scale register. For this reason a modification was made within the VHDL that uses a hardcoded value to generate the PWMs for the arm motors.

4.1.1.3. PCM-MIO-G I/O BOARD

This board was integrated into the stack by previous senior design teams. It currently does not have any function, but there is code that previous teams implemented. Our team also does not use this board, but it should be available for future expansion. This I/O board was capable of handling analog inputs and do digital pin toggling I/O. It was not able to practically do communication protocols such as SPI, I^2C, or UART.

4.1.1.4. FAULHABER 2232 U DC-MOTORS AND FAULHABER SERIES IE2-16 ENCODERS

Faulhaber motor and encoder

The ground robot features four DC motors and attached encoders that are used to drive the robot. Our team did not make any changes to this system.

Wheels

Our team purchases new wheels to replace the old ones on the robot, which were custom and tended to be brittle. The new wheels were purchased from Vicenza Thunders[1]. Four 6 cm and four 8.1 cm wheels were purchased, and the system currently uses the 8.1 cm wheels.

4.1.1.5. ADAPTER BOARD

The adapter board was already assembled and working with the ground robot when we began working with it. All design of the board can and its components can be found within the past Omnibot senior design SVN repositories (Dec11-13 and Dec 10-01).

Polou/Freescale MC33926 Motor Drivers

There are two Pololu dual “motor carrier” boards on the robot. The carriers are directly connected to one of the battery packs. Each board contains two Freescale MC33926 dual H-bridge drivers. A PWM signal generated by the motor controller core on the Mesa I/O board is used as input for the speed of each motor.

Pololu 6CHR-6d power regulators

A 5V switching regulator from digital electronics.

4.1.1.6. MISCELLANEOUS DEVICES

WIFI

A USB Wi-Fi module is used for all communication with the ground robot. The Wi-Fi is used to SSH into the robot when executing code and to communicate with the camera system.

USB Flash Disk

A USB flash disk is used to store the operating system on it.

4.1.2. ROBOT CHASSIS

The 3-D CAD models obtained from the previous Omnibot Senior Design Team did not fulfill our needs. We needed a broader, sturdier base with room for larger wheels. So we modified the CAD model for the robot frame to have a larger radius and flat cuts along the sides for wheels. This allowed us to

[1] http://www.vicenzathunders.com/

accommodate for large wheels. We also created four holes on the corners of the frame to use as a place to insert a structure which would hold the base joint stable above the robot.

4.1.3. BASE JOINT

A 2-dimensional joint is attached to Eris' main frame to limit the movement of the instrumentation arm to only two dimensions. The main reason to choose this approach despite many possibilities is the significant simplification of the control system when yaw rotation is removed. We are also unable to actuate a yaw rotation since the propellers were mounted at right angles. This joint was designed using the CAD tool SolidWorks, and tested with the simulation environment Simulink. Time and price limitations led the team to purchase a 3rd party universal joint instead of manufacturing with the student resources on campus.

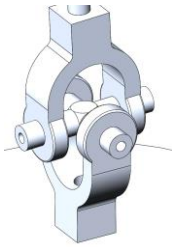

FIGURE 5: BASE JOINT

4.1.4. OPERATING SYSTEM

The Linux operating system being used on board of the ground robot (Eris) was built by Ed Cramer, a member of the Dec11-13 senior design team, and the following documentation is their latest documentation regarding the development OS being used for Eris.

The Linux kernel eris-00.03 was a more traditional desktop OS. It was merged into a Debian installation image and used to install the Development OS on the robot. The eris-00.04 kernel is very similar to its predecessor, but additional file systems were included to make it compatible with Debian Live systems. All of the kernels were compiled on a Debian 6 workstation in Coover 3050 (most likely the Omnibot-Linux-3 computer). Each kernel and its menuconfig settings file were committed to the SVN repository for the Dec11-13 team along with a document describing the changes and rationale behind each version. This is the most complete system available to run on the robot. It has the newest drivers and kernels plus a wide variety of tools to test the robot's performance. It is the primary target for new robot features. The development OS can also be used to build Debian Live systems.

Good

- Boots the newest kernel and initial ram disk
- Connects to the Wi-Fi network automatically
- Support for the PCM-MIO-G board (not currently used for our system)

Bad

- System can be corrupted if system power is lost

For more information on the operating system running on Eris, see the SVN repo and the final report for the Dec13-11 senior design team.

4.1.5. SOFTWARE

Our project's codebase is written in C/C++ for Linux systems. We had many versions of successful code that was used throughout the year our team worked on the project. Each of these versions of code can be found in a tar.gz file within the software section of the team git repository.

4.1.5.1 GROUND STATION CODE

On the first development phase our project we decided to use the pre-designed MicroCART framework used by previous senior design teams. The communication from the C/C++ embedded software to the system was done through a Digilent Nexys 2 FPGA in combination with a Spektrum D6Xi transmitter, while the position data was retrieved from the camera system to a Linux PC where the control software

was running. For a better understanding of this set-up the reader can refer to the MicroCART 2014 senior design team and their documentation.

The original software was targeted to control and drive a custom quadcopter, therefore our custom control software had to have a considerable amount of changes to target this application. The main objective at the early stage of this project was to be able to balance/control the instrumentation arm with 1 degree of freedom.

To accomplish our main objective we implemented the following features:

- A PID controller for the pitch direction
- Trimming variables to account for offset factors on the propellers and motors
- Refactor the code using structures for better organization and global access
- Created log file format to record the behavior of the system

4.1.5.2 MAIN PROGRAM VERSION 1.0

The 1.0 version of our code is essentially the unmodified Omnibot Eris codebase that was inherited from the Dec11-13 senior design team. The codebase is fully integrated with the Korebot code that was used in the 2008 and 2009 omnidirectional robot senior design teams. These teams featured two Korebot series use XScale processors and the Familiar Linux distribution. Eris was produced by the 2010 senior design team, and it was centered around the Intel Atom processor and the PC/104+ hardware platform.

Since the Eris code was integrated with the Korebot code, there were many external libraries that were not required for the compilation of the Eris ground robot code. This created a bloated system with many dependencies.

4.1.5.3 MAIN PROGRAM VERSION 2.0

Version 2.0 is the first huge revision to the software codebase. Within this revision, the Korebot dependencies were completely removed from the codebase. These changes were made in order to port the codebase to a new system, because we were only able to develop using one computer (omnibot-linux-2). Once the arm compiling dependencies required for Korebot were removed, the code was able to be compiled on the new Red Hat computers set up by ETG.

The changes required to remove the dependencies were quite simple. By deleting the makefiles completely and modifying the makefile.am files to remove the extra arm compilation, we were able to regenerate the makefiles that only compiled for the new system.

Another feature of version 2.0 is the ability to control the 1D pendulum (possibly the 2D pendulum, but we are unsure of this). The pinout is different from the recent versions because this version was outputting the PWM values over the pwmgena pins in the FPGA.

4.1.5.4 MAIN PROGRAM VERSION 2.1

This version of code uses the softdmc_1.2.bit file to control the pendulum. The output pins when using this .bit file are pwmgenc. 2D balancing and logging is available when using this version.

4.1.5.5 MAIN PROGRAM VERSION 2.2

This is the most recent version of code as of May 2015. The big change about this one is the removal of many unnecessary files that were not needed for our project (they may be added again later, and that is why we keep tar.gz file backups of the previous versions). This version also had a huge code restructuring happen with it. Many poor programming conventions were fixed.

Some of the issues fixed include:

- Circular dependencies
- Addition of pre-declared classes
- Removal of #include calls from header files
- Addition of include guards
- Formatting of poorly structured code
- Removal of dead code

You will also find that we are using the softdmc_1.4 bit file, and we can control the wheels on the Omnibot. By using this .bit file, we are able to control the wheels at the same time as the pendulum.

This version of code has access to three different running modes for the demo. We can start up in PID tuning mode that has no ability to move the ground robot, but we can change the PID values during runtime. The standard mode allows us to move the ground robot by the pendulum balances. The final mode is the square stepper demo that keeps the ground robot stationary while changing the set-point of the pendulum to make it move in a square formation.

4.1.5.6 MAIN PROGRAM PAULLQR

This codebase is a modification of the MainProgram_2.1 that Paul Uhing wrote. He wrote this version to test more advanced controllers for the instrumentation arm.

4.2. PROPELLER SYSTEM & INSTRUMENTATION ARM

4.2.1. INSTRUMENTATION ARM

4.2.1.1. ARM DESIGN

The instrumentation arm is constructed out of a single aluminum pole, and is 6 feet 1 inch long. The pole contains a mounting bracket for the IR camera constellation approximately half-way from the base.

The original instrumentation arm was constructed out of two aluminum rods bolted together in the middle. During testing though, the pole experienced large forces at the bolt joint and snapped the bolt.

Other options for instrumentation arm materials were then explored, including steel and carbon fiber. Both of these options provide more rigidity than the aluminum pole, however each has its downsides. The steel pole is heavier than the aluminum pole, which therefore limits the possible recovery angle of the instrumentation arm. The carbon fiber pole would be more expensive than either the steel or aluminum poles, and would most-likely involve a custom layup of the carbon fiber. These two alternative options were not explored fully due to time and material limitations.

4.2.1.1. BALANCING FROM THE TOP

The instrumentation arm is balanced using a propeller system located at its top. This method of balancing was chosen over balancing using the base of the arm due to limitations on the ground robot.

The standard method of balancing the instrumentation arm from the base would involve moving the ground robot in two dimensions (from side-to-side) to impart a torque onto the instrumentation arm. The corn rows this robot must traverse are only 30” wide, this makes movement in one of the dimensions too constrained to allow balancing from the base.

Additionally, if the instrumentation arm was balanced from the base, any reaction torque generated by the movement of the arm could flip over the robot base (due to the small profile of the robot in comparison to the length of the arm). The actuation at the top will cancel out the reaction torque felt by the ground robot, allowing it not to flip over when the instrumentation arm moves.

4.2.2. PROPELLER SYSTEM

The propeller system (figure 6) is the actuator we designed for the instrumentation arm. This system is a modification of a quad frame. The modifications that were made are explained in the following sections.

FIGURE 6: PROPELLER SYSTEM

4.2.2.1 FRAME

The DJI FlameWheel F450 Quadcopter stock frame was used as the main structure for the actuator system. Modifications were needed in order to fulfill our requirements of being able to actuate.

FIGURE 7: FRAME

4.2.2.2 MOTOR MOUNTS

Motor mounts were needed to redirect propeller thrust to a 90 degree angle from the instrumentation arm axis. This was designed using SolidWorks CAD software and mounted as seen in figure 8.

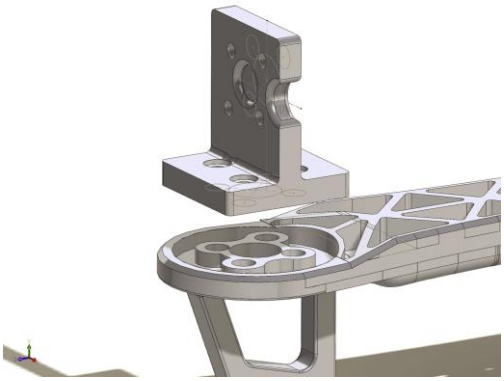

FIGURE 8: MOTOR MOUNTS

4.2.2.3 ARM MOUNTING BRACKET

This mount (figure 9) was needed to be able to mount the quad propeller system in a fixed position on top of the instrumentation arm pole.

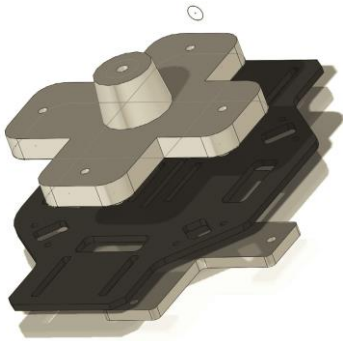

FIGURE 9: ARM MOUNTING BRACKET

4.2.2.2 ELECTRONIC SPEED CONTROLLERS

The Electronic Speed Controllers (ESCs) used to control the brushless motors on the propeller system were bought from DJI with the chassis kit. They are DJI Opto 30A ESCs, and were the standard ESCs used by DJI. The ESCs come with factory firmware preinstalled, and there is no way of calibrating the minimum and maximum PWM pulse width for the input signal. This meant the factory firmware always would have a deadband of ~20% duty cycle on the low range of the PWM, and would reach maximum thrust at ~90% of the PWM input signal's duty cycle. This can be seen in figure 25.

After performing research on the web, 3rd party firmware was located that allowed calibration of the minimum and maximum PWM pulse width for the input signal, along with other motor control parameters. The two major firmware projects are: SimonK[2] and BLHeli[3]. The SimonK firmware was developed for Atmel based ESCs while the BLHeli firmware was developed for both Atmel and Silicon Labs based ESCs. The DJI Opto 30A ESCs are Silicon Labs based ESCs, so the only firmware available was the BLHeli firmware.

Modifications were made to all four ESCs to allow for programming of the ESCs. The modifications consisted of removing the heat shrink cover, soldering four wires to the programming pads, then installing a new heat shrink cover. The bare ESC circuit board with the four programming wires attached can be seen in figure 10.

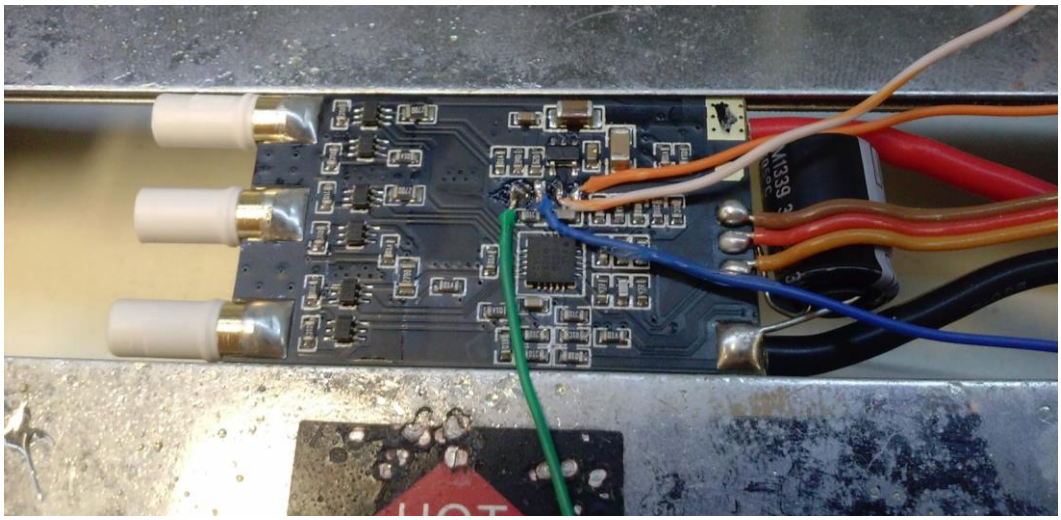

FIGURE 10: DJI 30A OPTO ESC WITH PROGRAMMING WIRES ATTACHED

This programming header allowed for the BLHeli firmware to be downloaded using an Arduino and the BLHeli Suite firmware flashing tool[4]. The BLHeli Suite software also allowed for various motor controller parameters to be configured.

After the BLHeli firmware was flashed onto the ESCs, further testing of the ESCs was conducted. This testing showed that the new firmware provided control from a 2.5% duty cycle to a 97.5% duty cycle, a

[2] SImonK firmware found at: https://github.com/sim-/tgy
[3] BLHeli firmware found at: https://github.com/bitdump/BLHeli
[4] BLHeli Suite found at: http://www.rcgroups.com/forums/showthread.php?t=2136895

larger control range than the DJI firmware provided. Additionally, the new firmware provided for a larger linear region in the motor speed, as seen in figure 25.

4.2.2.3 MOTORS AND PROPELLERS

The propeller system uses the DJI 920kv motors that are provided with the DJI Flamewheel 450 kit. These motors provide 920 RPM per volt on the input, allowing for a maximum speed of 11,040 RPM for the 12 volt power supply.

The propellers are the DJI 1038 propellers provided with the DJI Flamewheel F450 kit. These propellers are 10-inch with a pitch of 3.8. This kit also came with the DJI 0845 propellers, which are 8-inch propellers with a pitch of 4.5. These propellers produce different amounts of thrust when operating at the same speed. The speed versus thrust curves for four propellers all pushing the frame down can be seen in figure 27. This graph shows that four 10” propellers produce a combined maximum of ~1260g, while the 8” propellers produce a combined maximum of ~1060g.

Our final design used the 10” propellers due to the increased thrust it provided. This system prioritized the thrust of the propellers in its design because the thrust directly affected the stabilization region of the instrumentation arm (the more thrust, the larger the stabilization region).

The propellers on the propeller system were oriented to be at right-angles to the vertical axis, and were positioned on the frame so that one propeller was providing thrust in each cardinal direction. The angling of the propellers allowed for the thrust vector countering gravity to increase as the instrumentation arm moved from vertical. Additionally, the angling allowed for the thrust vectors to oppose each other when the instrumentation arm was vertical, allowing for the instrumentation arm to be easily balanced.

Since the propellers are oriented at right-angles to vertical, the airflow for every propeller is through the same space in the middle of the frame. With the propellers pulling air towards the middle of the frame (while still producing thrust outwards from the frame), a large pocket of turbulence was created in the middle of the frame. This turbulence creates inefficiency in the propellers, lowering the thrust production.

By reversing the airflow through the propellers (moving the air towards the outside of the frame) but still maintaining the thrust direction, the amount of turbulence created in the middle of the frame is reduced. This reduction in turbulence allows for an increase in the thrust production for each propeller.

The DJI 1038 propellers that are used in this project have unfortunately been recently discontinued by DJI. The DJI replacement propellers do not allow for the reversing of the airflow direction, so they are not compatible with this project. At this time, replacement propellers have yet to be determined.

4.3. DATA ANALYSIS TOOL

The command-line interface and the GUI of the Data Analysis Tool were developed using MATLAB. The tool parses log files that contain data in a certain format and stores the parsed data into a MATLAB struct, on which analysis is performed.

4.3.1. LOG FILE

The log files of an experiment must be in a .txt file. Since the tool is designed to be usable by any team, the format of the logged data is not fixed; however, every format of a log file will need a corresponding MATLAB parsing function that extracts the data in the log file into the format of the MATLAB struct. For the RADA project, the data is logged in a matrix format such that the data for a certain entity is in a column. The entity names, or headers, corresponding to each column of data are mentioned once in a line before the data is continuously logged. The units of the entities are also mentioned in a separate line before the data is continuously logged; the line with units can be located either before or after the line that mentions the headers. The log file should contain different symbols at the beginning of each line to signify the type of information being specified in that particular line. For the RADA project, the following symbols were used (figure 11):

- # - Information regarding the experiment's configuration. In the command line interface, this kind of information will be displayed once on the command line when the tool is executed. In the GUI interface, this information is disregarded.
- % - Names of the entities being logged. We refer to entity names as "headers."
- & - Units of the headers

```
#Constants      pitch_P     pitch_I     pitch_D
#PIDValues      1650        3       550
#Constants      roll_P      roll_I      roll_D
#PIDValues      1650        3       550
%Time       Marker      Motor_1 Motor_2 Motor_3 Motor_4 P
&sec        Marker      %thrust %thrust %thrust %thrust d
7432.245736 0   26000   56000   56000   26000   1.116891
7432.225854 0   30645   34606   51355   47394   1.433347
```

FIGURE 11: LOG FILE FORMAT FOR THE RADA PROJECT

If n entities are being logged, n units must be mentioned and n data values need to be logged in each row. Although any entity can be logged in the log file, the following entities must be logged:

- Time - timestamps of the logged data. This column contains time values for every row of data that is logged. In this column, each value must be greater than the previous value. For the RADA project, the first value of this column need not be 0 or 1. RADA's log-parsing function computes a new time vector which contains values relative to the first one. In other words, the parsing function replaces the values in the Time column with the results of the following function: Time(i) = Time(i) - Time(1) [where Time(1) refers to the first value in this column and Time(i) refers to the i^{th} value in this column].

- Marker – locations at which the marker has been applied. Whenever a marker is applied during the experiment, a non-zero value that corresponds to the marker number should be logged under this column. For example, when the third marker is applied, the value 3 should be stored under this column. When no markers are being applied, a 0 is logged under this column. For the RADA project, the number m is logged under this column whenever the m^{th} marker is applied. For example, when the third marker is applied the value 3 is logged continuously under this column until the fourth marker is applied. The parsing function converts the logged values into the format described above; the Marker data will contain non-zero elements only at the times at which the markers were applied and zero elsewhere.

4.3.2 STRUCT

The data logged in the text file, when parsed, is stored in a MATLAB struct called "main." This is true for both interfaces of the tool. The "main" struct is primarily divided into two structures:

- *params* - a struct that contains the parameters of the analysis. These parameters include plotting options and log file details.
- *expData* - a struct that contains all of the data logged in the text log file in the form of MATLAB vectors.

The following list hierarchy precisely describes the organization of the contents within the "main" struct:

- main
  - params
    - file
      - *name*: name of the file
      - *path*: location of the directory in the system where the file is located
      - *pathName*: location of the file in the system (path+name)
    - plotting
      - *plot*: main plotting switch
      - *separatePlot*: switch to generate separate-plots
      - *multiPlot*: switch to generate multi-plots
      - *subPlot*: switch to generate sub-plots
      - *clearFigs*: switch to close all open figures (used only but the GUI)
      - *separateData*: MATLAB cell-array that contains names of the headers to plot using separate-plots
      - *multiData*: MATLAB cell-array that contains names of the headers to plot using multi-plots
      - *subData*: MATLAB cell-array that contains names of the headers to plot using subi-plots
      - *color*: specifier of color of plotting line
      - *marker*: specifier of marker of plotting line
      - *style*: specifier of style of plotting line
      - *backgnd*: specifier of background color of plot
  - expData
    - <header-name>
      - *data*: a vector that contains all of the logged data contained in the log file for a particular header
      - *unit*: units of the header as mentioned in the log file

- params: struct that contains information about the plotting parameters to be used for this header in specific. These parameters override the values of main.params.plotting, if any different
  - *plot*: switch to generate plots for this header
  - *style*: specifier of style of plotting line
  - *color*: specifier of color of plotting line
  - *marker*: specifier of marker of plotting line
  - *backgnd*: specifier of background color of plot

4.3.3. PLOTTING

There are three types of plotting options (that have corresponding switches):

- plot_separate: separate-plots (figure 12) are those that contain only one entity. If n entities were to be plotted using separate-plots, n plots will be generated with one plot for one entity. If the separate-plotting switch is set to 0, no separate-plots will be generated regardless of the other plotting options. If set to 1, separate-plots will be generated for the required entities mentioned in the *separateData* cell array.

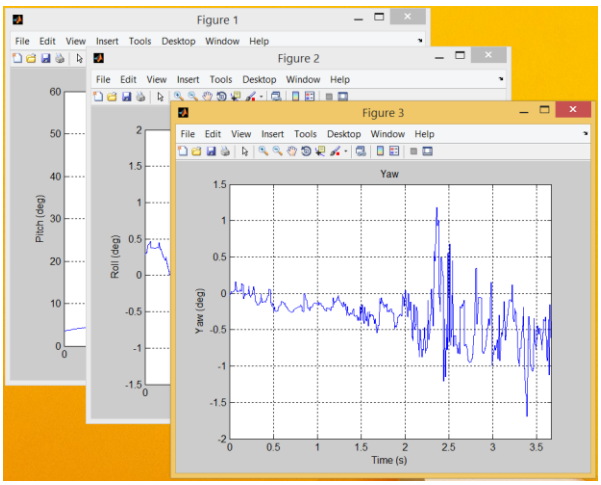

FIGURE 12: SEPARATE-PLOTS

- plot_multi: multi-plots (figure 13) are those that contain multiple entities. If n entities were to be plotted using multi-plots, n plots will be generated with one plot for all entities. If the multi-plotting switch is set to set to 0, no multi-plots will be generated regardless of the other plotting options. If set to 1, multi-plots will be generated for the required entities mentioned in the *multiData* cell array.

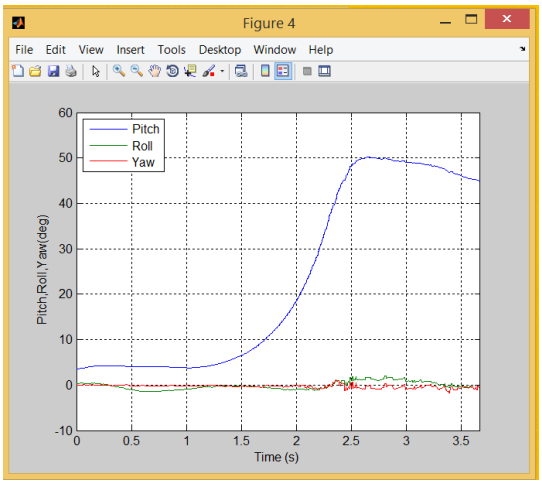

FIGURE 13: MULTI-PLOTS

- plot_sub: sub-plots are those that contain multiple independent graphs in a single window. This tool generates 2x1 subplots i.e. if n entities were to be plotted using sub-plots and if n is even, n/2 plot windows will be generated with one window containing two independent plots. If n is odd, n/2 + 1 plot windows will be generated; the last entity will be generated using a separate-plot. If the sub-plotting switch is set to set to 0, no sub-plots will be generated regardless of the other plotting options. If set to 1, sub-plots will be generated for the required entities mentioned in the *subData* cell array.

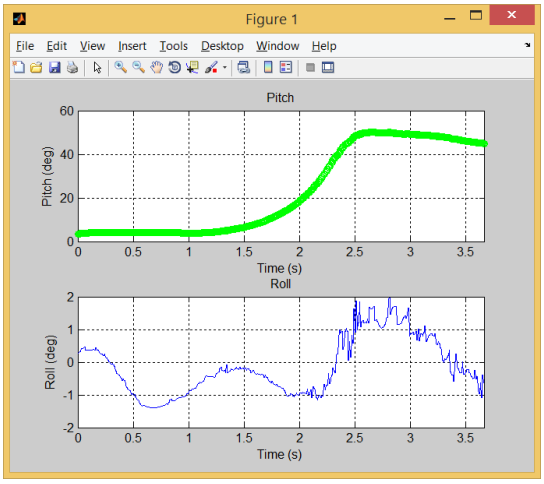

FIGURE 14: SUB-PLOTS

4.3.4. MATLAB GUI

MATLAB GUI (figure 15) utilizes the Data Analysis Tools to provide an easy to use interface to users who aren't as familiar with MATLAB.

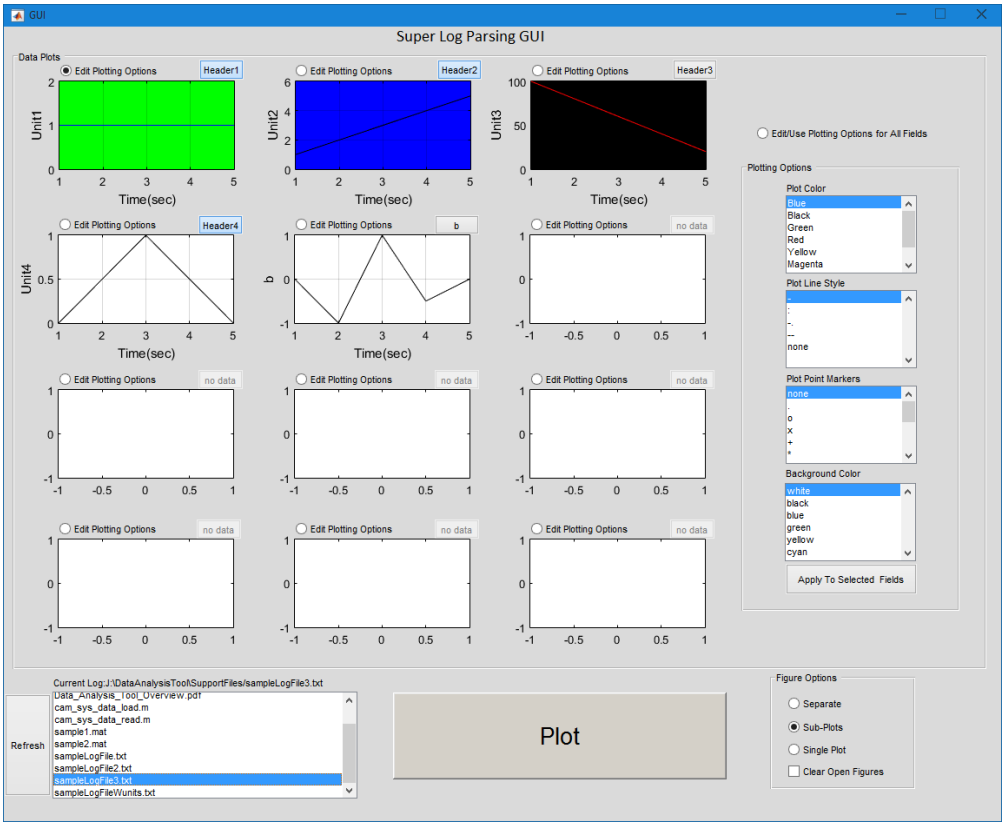

FIGURE 15: MATLAB GUI

The GUI makes it easy to customize the plot outputs down to the background color of the plot. Although this can be done with the DAT, the list style selection options make it very easy to see available options.

5. TESTING

5.1. BASE JOINT SENSOR TESTING

During the development phase of our project we tested different kinds of sensors to implement in future versions of this system. A sensor would be used to determine the orientation of the instrumentation arm relative to the ground robot.

5.1.1. SOFTPOT MEMBRANE POTENTIOMETER

To gather the most accurate measurements of data we decided to perform two different tests with this peripheral.

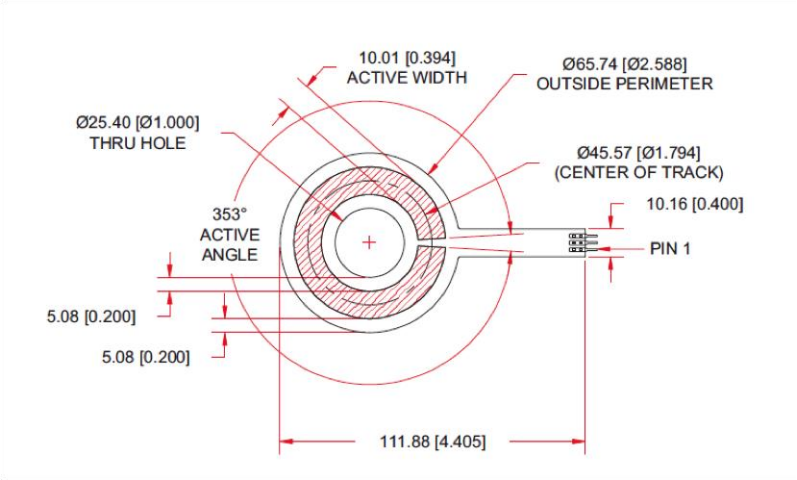

FIGURE 16: SOFTPOT MEMBRANE POTENTIOMETER

5.1.1.1. TEST 1

We used a protractor and a multimeter to measure the resistance for different angles of pressure application (figure 17).

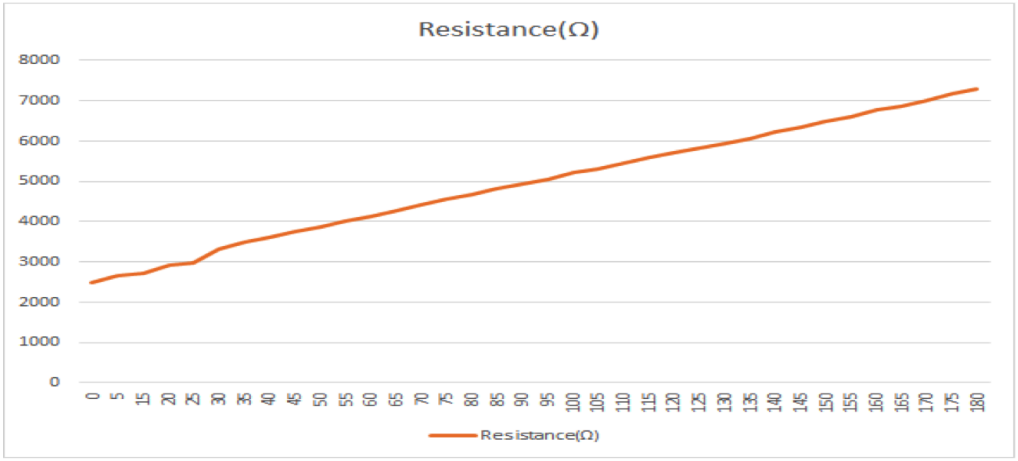

FIGURE 17: RESISTANCE VS ANGLE

5.1.1.2. TEST 2

The second time, we used the camera system to record the angle precisely while simultaneously recording the voltage drop across the potentiometer using MATLAB and an Arduino. In figure 18, the graph on the bottom represents the angles measured by the camera system and the one on the top represents those measured by the sensor.

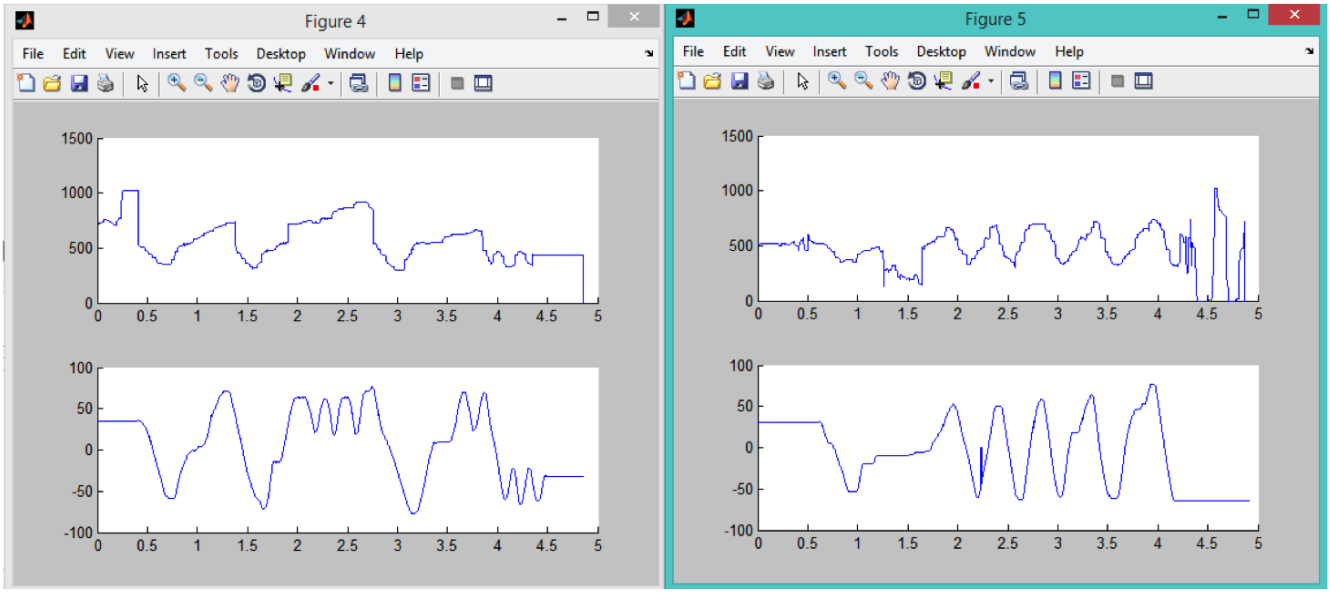

FIGURE 18: DIGITAL OUTPUT VS TIME

5.1.2. HALL EFFECT ENCODER (AMS ROTARY SENSOR - AS50489)

This product is a digital output rotary sensor integrated circuit. It requires an adaptor board in order to test. The results of the tests are as follows

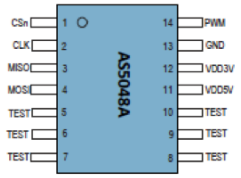

FIGURE 19: HALL EFFECT SENSOR

5.1.2.1. TEST 1 - RANGE:

Similar to the Rotary sensor, we tested this sensor with the help of the Tracking tools camera system but used the chipKIT uC32 board to communicate through SPI to MATLAB for analysis. This is just one of many test runs of tracking the angle reading of the sensor along with the camera reading data.

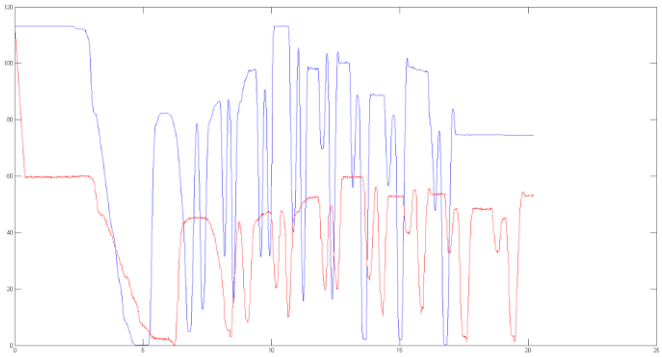

FIGURE 20: RANGE TESTING (BLUE - CAMERA; RED - SENSOR)

5.1.2.2 TEST 2 - PRECISION

Figure 21 shows the result of this test. This test observes the output of the encoder compared to the camera system data. There is a definite step between increases in angle but it is obviously noisy. However, this is much better than the membrane potentiometer (figure 21: sensor is red; camera is blue).

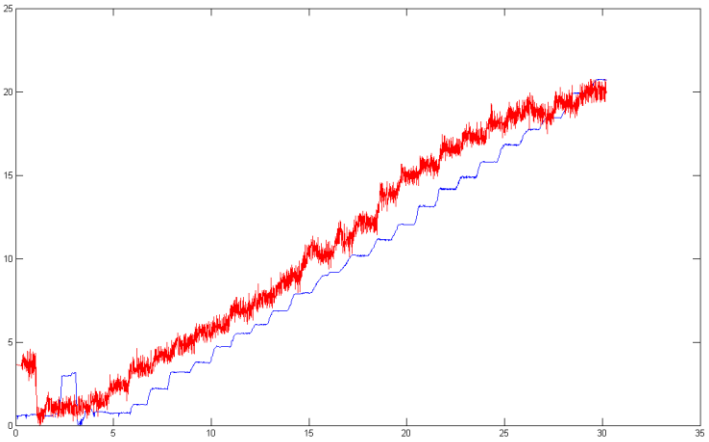

FIGURE 21: PRECISION TESTING

5.1.2.3. TEST 3 - NOISE

As seen in figure 22, the noise is comparable to the camera system noise (figure 22: sensor is red, camera is blue).

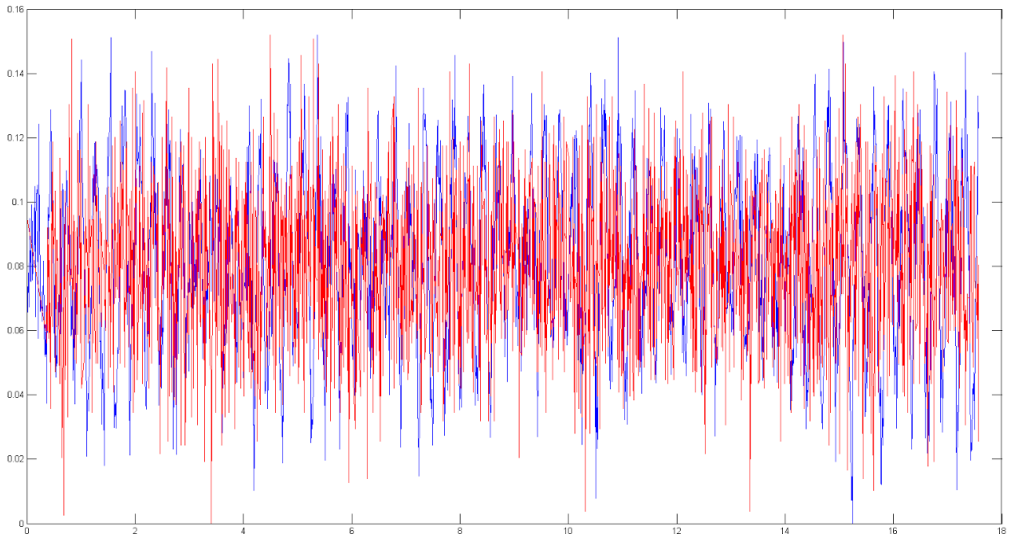

FIGURE 22: NOISE TESTING

5.2. MOTOR TESTING

5.2.1. TEST SETUP

The motor speed was tested by measuring the motor speed at various PWM duty cycles. The motor speed was measured using an Extech 461895 Tachometer with IR reflective tape[5] and an Arduino configured to output a specified PWM duty cycle to the ESC. The physical setup can be seen in figure 23.

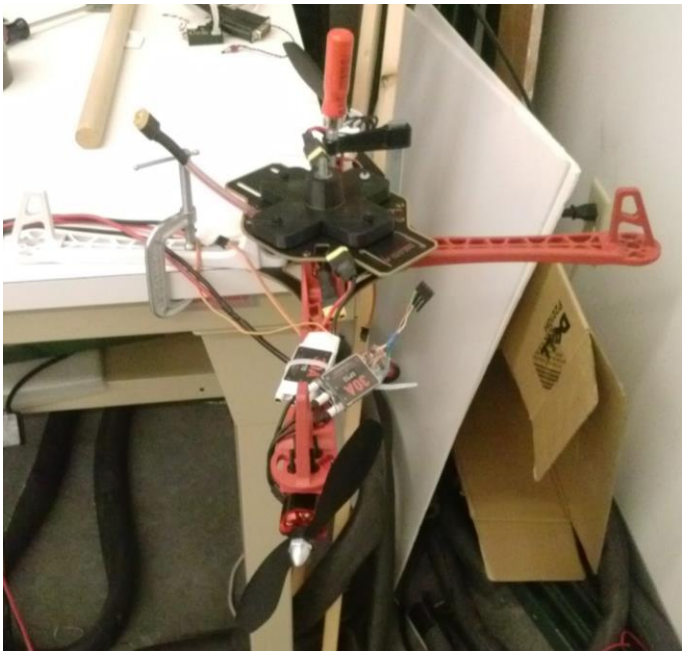

FIGURE 23: TEST SETUP FOR THE MOTOR SPEED

The testing for the motor speed with various firmware versions was conducted using the following test procedure:

1. Install the desired firmware/ESC into the propeller system
2. Start the Arduino program and send 0% duty cycle
3. Increase the duty cycle by increments of 2.5%, measuring the speed at each step
4. Repeat the above steps for each firmware/ESC to be tested

5.2.2. TEST RESULTS

One test conducted using this setup was a motor correlation test. This examined how closely the four motors behaved in terms of the speed produced versus the commanded duty cycle. The results of this test can be seen in figure 24, and show that the four motors and ESCs perform very similarly.

[5] http://www.optitrack.com/products/motion-capture-markers/#msc1040

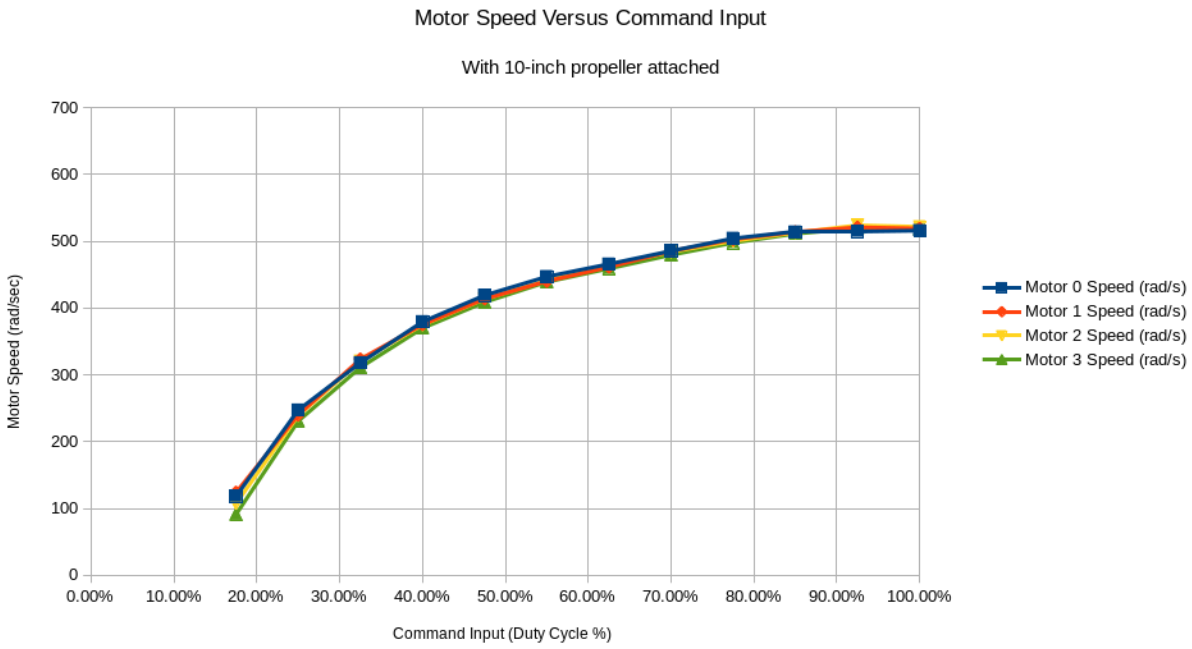

FIGURE 24: MOTOR SPEED FOR FOUR DIFFERENT ESCS AND MOTORS

A second test performed using this setup was to measure the linearity of the motor speed with different firmware versions. The results of this testing can be seen in figure 25.

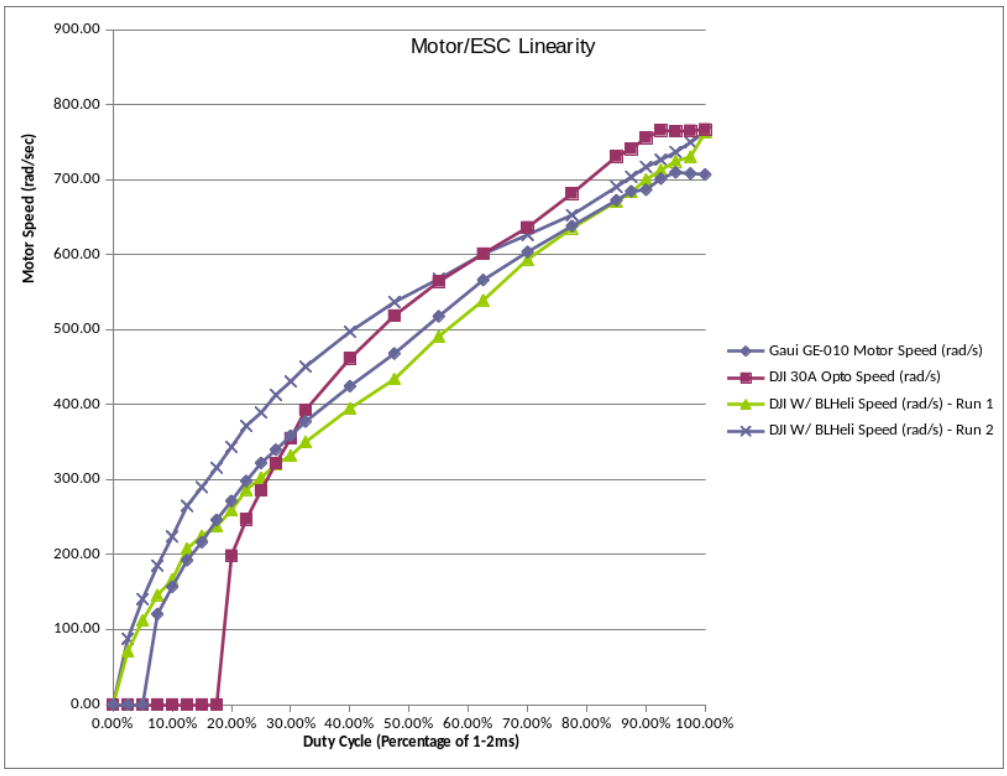

FIGURE 25: MOTOR SPEED VS PWM DUTY CYCLE FOR VARIOUS ESCS AND FIRMWARE

The stock DJI firmware contained a large deadband at the beginning (~20% duty cycle), and a large deadband at the end (~10% duty cycle). Additionally, the linear region of the speed was fairly small (~50%). The BLHeli firmware on the DJI ESCs contained a deadband of ~2.5% at the bottom, and virtually no deadband at the top. The modified firmware also contained a larger linear region (~90%).

5.3 PROPELLER TESTING

5.3.1 TEST SETUP

To measure the thrust produced by the 8” and 10” DJI propellers, we mounted the MicroCART frame onto a Dymo M10 scale set to measure grams. The test setup can be seen in figure 26.

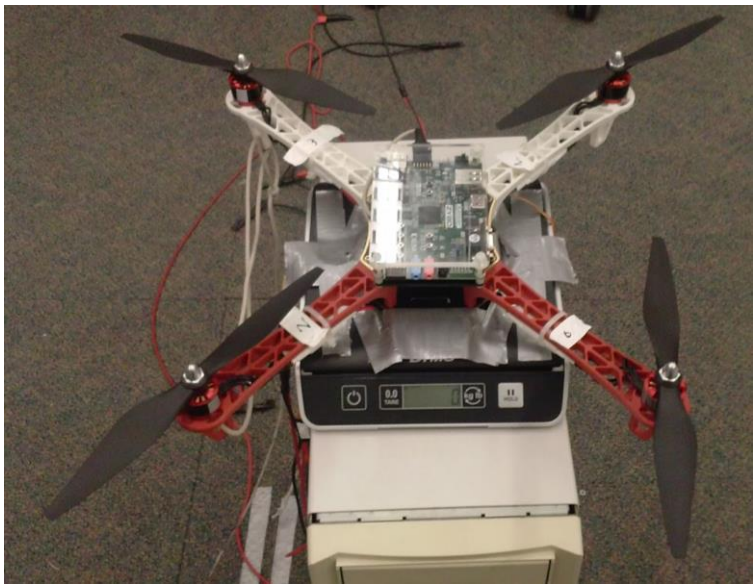

FIGURE 26: TEST SETUP FOR MEASURING THE THRUST PRODUCED BY FOUR PROPELLERS

The procedure followed when conducting the test is:

1. Initialize the motors to 0% duty cycle
2. Increase the duty cycle
3. Record the speed of each motor and the overall thrust produced
4. Repeat for duty cycle

5.3.2 TEST RESULTS

The testing using this setup was designed to measure the thrust produced by the two different DJI propellers provided in the DJI Flamewheel 450 kit. The overall results are shown in figure 27. These results show that more thrust is produced by the 10” propellers than the 8” propellers.

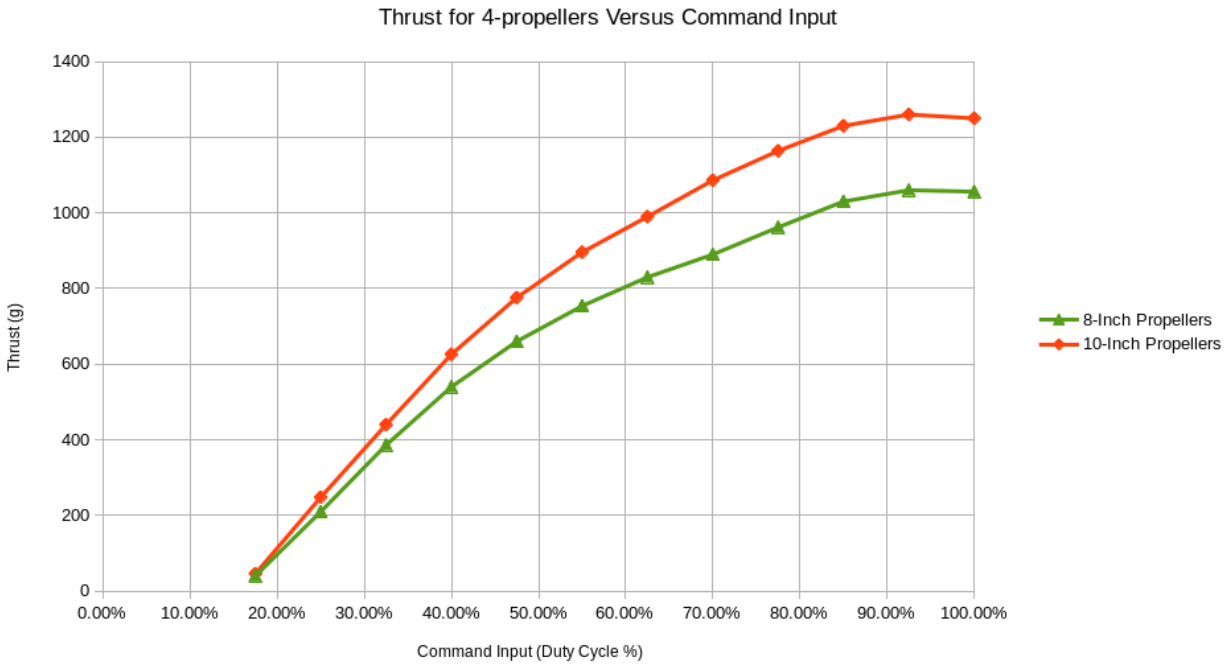

FIGURE 27: PROPELLER THRUST VS INPUT COMMAND FOR 8-INCH AND 10-INCH PROPELLERS

APPENDIX I

This appendix details how to use the system and the GUI for plotting.

SYSTEM USAGE

1. Power on the ground robot
    a. Wait for the blue LED on the Wi-Fi card to light up solid
2. Run SSH root@192.168.0.53 to remote into the ground vehicle[6]
3. Run the 1-co3050-11.sh script (./1-co3050-11.sh)[7]
4. Once the script has run, the display should indicate how to control the system

For more information, see the RADA wiki[8].

GUI USAGE

REQUIREMENTS

- MATLAB(Linux or Windows) Version R2014b or higher (will probably work on previous versions, but not tested)
- GUI.m – Matlab code file which determines functionality of the GUI
- GUI.fig – figure file which generates GUI graphics
- parse_log.m – log parser function used by the GUI

STEP-BY-STEP

1. Opening GUI
    a. Navigate to the directory containing GUI.m and GUI.fig
    b. Right click on GUI.m
    c. Click Run
    d. Wait a few seconds and the GUI should open
2. Navigating In-GUI browser to Log-File
    a. Double-Click on listbox entries to change directory until you find the log-file directory
    b. The current log or directory will be displayed above the listbox
    c. There is a refresh button which will load any files that have changed since the listbox was last loaded.
    d. Once you reach the Log-file location, Double-click the Log-file and the data will be parsed
    e. Once parsed, The GUI will automatically plot the data on the axes and export the data to an M-file
3. Changing Plotting Options
    a. Select a field for which you would like to change the settings
    b. Alternatively, one can have all plots use the same options by selecting the radio button above the plotting options UI-box
    c. Then, select the desired settings

---

[6] Note this will only work on a computer connected to the local KNET network
[7] Note this script can be customized to mount to any computer, but there will need to be a /home/shared directory and NFS needs to be set up on the computer being mounted to (recommended to use Linux)
[8] http://wikis.ece.iastate.edu/robotic-agriculture-data-acquisition/index.php/Main_Page

d. Plot Color
e. Changes the color of the plot line or marker
f. Plot Line Style
g. Changes line which connects data samples to the desired style. Can be dashed, dotted and dash-dot or no line at all.
h. Plot Point Markers
i. An option to mark data samples with a character.
j. Background Color
k. Changes the background color of the

Note: The background color for the single-Plot Plotting method will always be white because there are multiple fields in one plot. plot

l. Press the large Plot button at the bottom

POTENTIAL PROBLEMS WITH SOLUTIONS

- Log file Format
- There are unmatched Units, Names or Data
  - One or Two of the numbers of Units, Names or Data does not match the others.
  - This is most likely a logging problem and can be fixed by adding or removing the needed lines.
  - It can also be caused by having too many tabs between data values. The parser will see an empty data value in-between two tabs.
- Could not find &, # or % headers
  - You need to include which ever header is missing to the log file.
  - Can be fixed by adjusting the logging to include the missing header.
  - Also might be because the order of the headers is wrong.
    - The # headers must be first. The order of the & and % headers does not matter but it is suggested to have the % before the & headers.
- Mat-Data Format
  - Most problems with the M-data are due to inconsistent data.
  - The go-to solution for these problems is to delete the main structure and re-import the data from the log file.

APPENDIX II

SENIOR DESIGN WEBSITE

http://may1527.ece.iastate.edu/

Our senior design website contains the documentation (weekly reports, design documents, presentations, etc.) that was created throughout our senior design project.

TEAM WIKI PAGE

http://wikis.ece.iastate.edu/robotic-agriculture-data-acquisition/index.php/Main_Page

Along with our website documentation, we used an ISU wiki to write more detailed documentation with regards to steps taken during development, issues that were encountered, and other important issues.

TEAM YOUTUBE PAGE

https://www.youtube.com/playlist?list=PLJU0aKMTM5ea51fXcESAZ5q5wWAF9ZJZG

This link leads to a playlist for different tests that were completed throughout the year.

APPENDIX III

SOFTWARE TOOLS

MATLAB

Purpose

MATLAB was used to parse log files which contained data from experiments and test-runs. The log files could either be text files generated by the “control-GUI” or they could be CSV files generated by the OptiTrack Tools software of the camera system. MATLAB was also used to generate plots for the parsed data according to user’s configuration options.

SIMULINK

Simulink will be used during the design of the controller to model the overall system and simulate the response of the system to different control algorithms. A differential equation model for the non-linear system dynamics of the instrumentation arm will be created and then placed into a Simulink block-diagram. Then, a model of the desired control algorithm will be created in Simulink. This model will then be paired with the system model, and the overall system response to the controller will be measured.

This analysis will assist in the design and identification of possible controllers for the instrumentation arm. In addition, the detailed simulations will provide early insight into possible issues that may be faced in the physical system, and allow for possible solutions to be integrated into the design faster.

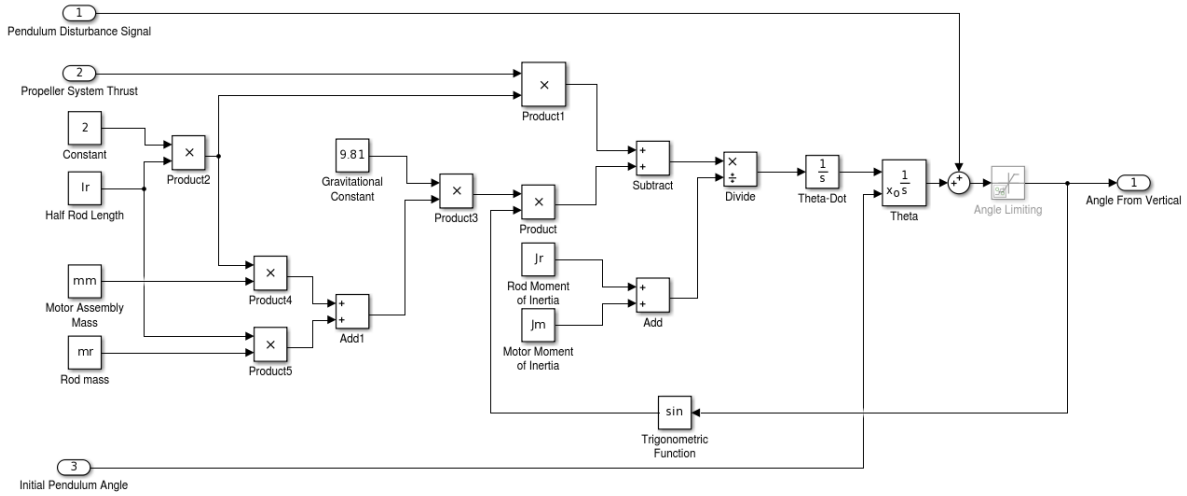

FIGURE 28: SAMPLE SIMULINK BLOCK DIAGRAM

SOLIDWORKS

Used to create concept for hardware design in order to envision and streamline prototyping process.

HARDWARE TOOLS

XILINX ISE

The Xilinx ISE is used to write VHDL files and generate BIT files that can be programmed onto the FPGA on the ground robot.