SphereBot and DOC

Robert Wortman, *CpE Student* Aaron Diab, *CpE Student* Darrell Cahail, *CpE Student*
Rebecca Wingo, *EEE Student* Emmanuel Dupart, *EEE Student*

Abstract

In order to reduce human exposure to the hazardous work environments involved with power distribution line inspection, the developers present a thrust-vectored flying robot to perform inspections remotely. The particular features of the Mechanical Build System, the User Interface System, the Control System, the Camera System, and the Stabilization System are analyzed for expected and actual resources and subtasks. Funding is briefly discussed, the expected tasks required to construct the prototype are outlined, and the actual time worked is summarized. An operational manual is provided, as well as hardware, software, and mechanical design documentation. Finally, hardware and software testing are described.

Index Terms

Aerospace electronics, Pulse width modulation, Robot control, Robot programming, IP networks, Aircraft navigation, Command and control systems, Robot sensing systems, Servomotors, Unmanned aerial vehicles, Attitude control, Multithreading, Robot motion, Sensor systems, Microcontrollers, Electricity supply industry, Mobile robots, Industrial accidents, PD control, Wireless networks, Parallel programming

Manuscript created Apr 17, 2014; revised May 4, 2014.

SphereBot and DOC

CONTENTS

List of Figures

List of Tables

I. Introduction

IN ORDER to ensure reliable power distribution systems, inspection personnel must work in the hazardous environment of energized high-voltage transmission lines. In order to alleviate this occupational hazard, the SphereBot developers have produced a proof-of-concept prototype for an airborne industrial inspection robot that can, among other possible uses, perform these inspections without exposing personnel to risk. Using a thrust-vectoring model to achieve superior stability, the robot will visually inspect the condition of transmission lines and provide telemetry that can be used to identify problematic areas which require further attention.

The SphereBot has achieved a high degree of development, including a third-generation Mechanical Build, second-generation Stabilization System, complete User Interface System, and third-generation Control System. Difficulties encountered thus far include the unfortunate nonfunctionality of an advertised feature of the chosen microcontroller platform, inherent instability in the Inertial Measurement Unit, and the need for late-in-project overhauls of the physical frame and the control software. The outboard Deployable Observational Crawler completes the features of the prototype with a Vision System that gathers optical telemetry.

II. Societal Problem

People the world over are becoming increasingly dependent on electrical energy in their daily lives. This dependence demands a robust, well maintained electrical energy distribution system, the backbone of which consists of high voltage electrical power lines. Despite the increasing prevalence of advanced automation technologies inspection and maintenance of these power lines are still largely done by personnel in direct contact with the high voltage lines. The direct contact with the high voltage lines, as well as the relatively extreme heights involved, present many dangers to the personnel who inspect these lines. The inherent dangers of line inspection also present power line workers with other challenges in their lives outside of the ones that they face while actually on the lines. There are several characteristics of power lines that are taken into consideration when inspecting power lines that could easily be accomplished using sensor technology which has been recently made available. It is suggested that by combining these sensor technologies with a suitable semi-autonomous delivery platform the dangers of having personnel manually inspect power lines could be successfully mitigated.

A. Current state

One of the uses for the industrial inspection platform is to inspect high-voltage transmission lines. In California and across the United States a veritable army of people are employed to inspect these lines. This is a very hazardous job category with many deaths per year.

Currently these lines are inspected by this army of inspectors using physical inspection techniques. Automation has been difficult due to the complexities involved. One of the complexities is the line voltage. The voltage in the lines, ranges from 250 kV up to 1000 kV, with the most common being 250kV to 500kV range.

> The inspectors have a title. They are called Line Installers. They inspect and repair high-voltage electrical power lines. These inspectors often work high off the ground. These wires carry dangerous amounts of electricity. These people require intensive training to perform their job. This training is continuous throughout their careers. Due to the dangers associated with high-voltage electricity, electrical power-line installers tend to make more money than others in this industry [1].

The techniques of inspection vary depending upon the kind, layout, and voltage of the line being inspected. These techniques include using an insulated boom, "bare-hand," and helicopter crew [2] inspection. It must be noted that all of these repair techniques are accomplished without de-energizing the line, even for replacement of any cable and/or insulator. In the eyes of the national energy transmission, it is cheaper to lose a person than it is to de-energize the line.

In an insulated boom the inspector is given a lift up to the high-voltage line to effect repairs and inspect the particular line.

> Utilities have practiced these so-called bare-hand techniques for maintaining high-voltage transmission lines in the United States and Canada for almost 30 years; more recently, other countries have

also adopted the practice. The term bare-hand is actually misleading. When working above 150 kV, the lineman's entire body, including the hands, is covered with a conductive stainless steel suit, hood, and gloves to equalize voltage across the surface of the body. Only the worker's face is uncovered.

Bare-hand methods may be on the rise, according to some industry officials, as utilities build lines at higher voltages and sell more power to other utilities, making it highly inconvenient to take the lines out of service. New technology is also helping; for example, helicopters and crews, again energized at hundreds of thousands of volts, can complete large jobs quickly and service isolated power lines [1].

Besides using a helicopter to inspect the line, some people inspect the line using a trolley that is dropped upon the line. The inspector gets into the trolley and slowly moves down the line inspecting the line as the trolley moves forward under the power of the inspector. This method requires that the inspector get on and off the trolley at each tower. This may be accomplished using either a helicopter or an insulated boom.

Currently, in the State of California there are employed, approximately, 7,260 [3] people inspecting the high-voltage lines. These people make an average hourly wage of $42.25. Of the people employed 26 [4] have been killed so far this year. The number of people working in this field is expected to grow as much as 13% [5] over the next 7 years.

B. Hazards

Manual power line inspection is unquestionably a dangerous occupation. Danger is found in several forms during the course of line inspection by personnel in direct contact with the power lines. The dangers faced by power line workers involved in "bare handed" line inspection take three basic forms, injuries from the unintended flow of electrical current, injuries resulting from a fall and miscellaneous injuries. These three basic categories can be further broken down demonstrating the gamut of occupational pitfalls that the average power line worker must navigate during the execution of their duties.

Given the very high voltages and currents present while working on live electrical lines the possibility of injury due to unintended electrical current flow is very high. Injuries from electrocution can be divided into two categories, burns and injuries due to the physiological effects of electrocution. Burns can occur either due to electrical current flowing through the body or from exposure to the intensely bright light generated when electrical current is allowed to arc. This phenomenon is known as "arc flash" [6].

The physiological effects of unintended electrical current flow are varied and the severity of injury depends on the bodily system effected as well as the characteristics of the current flow [7].

Power lines are most often located high up on electrical poles or towers. Inspection workers must either climb these poles or be transported to them by boom attached bucket or helicopter. The heights at which these workers conduct their business creates an inherent danger of injury from falling. There were 437 fatal falls in 2012 in which the height was reported and of these falls one-fourth of them were from a height of less than 10 feet [4]. With electrical tower heights up to 150 feet the potential for fatal injury from an electrical tower is evident [8].

The miscellaneous category of injuries are harder to quantify. This category includes injuries incurred during the course of work such as in the transportation to the lines. Examples are helicopter crashes, workers being crushed by the bucket boom or pushed into wires. Included are also injuries caused by working in extreme weather.

C. Impact

In addition to the constant threat of injury, power line workers face other challenges attributable to their chosen profession. In some localities, where public policy has not been enacted to combat the problem, many line workers find that purchasing life and health insurance is either extremely arduous or in some cases not possible. The negative impacts of this dangerous occupation are not isolated to the power line worker themselves but extend as well to their families. According to the National Institute of Health the death of a parent as significant impacts for children extending into adulthood. In the NIH study "Death of Parents and Adult Psychological and Physical Well-Being: A Prospective U.S. National Study" the following was noted:

> Overall, we found considerable evidence supporting the idea that because of long-term linked lives across time and because of the typically strong affectional bonds and attachment experienced with mothers and fathers, the death of a mother or father or both in adulthood is associated with a number of negative effects on mental and physical well-being. [9]

D. Measurands

During the inspection, power lines are checked for defects, signs of impending failures, and the environmental hazards that compromise their security and normal operation.

Defects of power lines include physical damages and corrosion of wires, damages in insulators and in structural elements of power line towers. In order to detect them, a range of types of cameras can be used: ones that work either in visible light, infrared (IR), or ultraviolet (UV) range. Damaged wires are usually detected by visual inspection, but UV cameras help more - they make corona effects around broken aluminum strands possible to be seen [10]. Corrosion of wires manifests itself by build-up of heat, which can be detected by IR camera. The defects in insulators usually are detected by visual inspection, but sometimes IR and UV are used [10].

Environmental dangers to power lines include bird-related and vegetation hazards. Bird-related hazards that need to be detected by power line inspection include biological waste build-up on wires and other energized equipment, nests and large flocks of birds [11]. Nesting material and animal waste contribute to corrosion of ceramic insulation and wires. Also, nests attract other animals, which can increase risk of power line failure. Visual inspection is the primary tool for detecting the nests and places of rest for large flocks of birds, although infrared could also be used.

Vegetation, such as trees, also endangers the security of power lines. They can damage the wires or cause the shorts to ground and line-to line shorts. For example,

> The 2006 blackout in the Western U.S. and Canada was initiated by a combination of inadequate tree trimming and high conductor sag, caused by a high conductor temperature, caused by high load current and low wind conditions. [12]

Since trees grow about half a meter per year (or even faster in some cases), the regular inspection of the distance between vegetation and power lines is needed [10]. This task requires only the ordinary visual inspection performed by a simple camera.

E. Our Solution

There are several available approaches to distribution line inspection. Historically, inspection was performed by personnel using binoculars while on foot patrol, though this method has several drawbacks. It is slow at best, and impossible in some terrain. It also lacks for accuracy, as personnel must maintain a safe distance from the wires. Helicopter-assisted remote inspection is faster and not limited by terrain, but it is less accurate due to the improved speed and due to vibration from the platform. Helicopter-assisted, bare-handed inspection does not lack for accuracy, but exposes inspectors to the hazards discussed above. For this reason, a need exists for autonomous systems that can work in close proximity to power lines without exposing human operators to hazardous conditions [10].

Some work has already been performed to develop such a system. There are two major categories of power line inspection robots: climbers and fliers. Climbers are constructed to crawl along the power line, traversing obstacles and recording data. While locomotion along the line itself is reasonably straightforward, such a system faces a daunting variety of obstacles, including turns, junctions, tower components, insulators, and aircraft avoidance markers. Simple approaches may be used to surmount particular obstacles [13], [14] but only very complicated mechanisms can operate reliably under field conditions. Crawlers also operate much more closely to the power conductors, and therefore face dangerously high electromagnetic fields. Fliers, on the other hand, need only follow the line from a reasonable distance, respond to wind conditions, and occasionally avoid nearby vegetation.

Common approaches to the design of flying platforms suffer from a similar problem to that of helicopter-assisted remote inspection: the vibration and speed of the platform blurs the image of the distribution line and degrades the quality of the telemetry. Furthermore, as most such platforms use computer vision to perform navigation and line-following, this image degradation also impairs the robot's ability find and follow the conductor while

minimizing proximity [10]. The vectored-thrust design of the SphereBot has been shown to provide a stable platform that responds well to environmental influences. This will improve the ability of the system both provide telemetry for inspection and obtain for itself environmental feedback.

The spherical chassis of the design mitigates the potential for damage in the event of error. In the event that the robot contacts the distribution line, the enclosed exoskeleton prevents the propellers from gouging the power line or insulators, which would necessitate costly repairs. As nearby vegetation is also a recognized hazard, the contours of the chassis will also help to avoid fouling which would interrupt the inspection and may require a costly and potentially dangerous recovery operation.

F. Proposal

By utilizing an automated, mobile platform for accurate inspection of power distribution lines, we hope to eliminate much of the occupational danger associated with power grid preventative maintenance. And, by minimizing the risk to workers, we hope to help eliminate the harm, not only to inspection workers, but to their families, as well. The rapid growth and advance of automation technologies will soon remove the need for humans to expose themselves to the dangers posed by the high voltages and working altitudes required for this task. These relatively cheap automated systems can ensure a healthy, reliable power grid, and can reduce the final cost of the electrical energy on which we depend.

III. Design Idea

This section reviews the Design Idea at the outset of the Laboratory Prototype stage, then addresses specific redirection along the way to the Deployable Prototype.

A. Feature: Mechanical Build

The mechanical build of the robot is comprised of the airframe of the robot, the control surfaces, the control surfaces actuators, main motors and propellers. The mechanical build of the robot provides the robot with the physical resources needed to stay airborne.

1) Hardware: The airframe of the robot will be spherical in shape. The shape of the airframe prevents the propellers from being damaged or damaging power lines. It also prevents the robot from becoming entangled in power lines or insulator strings. The control and motor drive hardware will be located in the center of the robot in a convenient housing. The propellers will be housed within the spherical airframe thereby being protected by the airframe skeleton. There will be at least four control surfaces that allow for translation and rotation control. They will be built in to the airframe body. The control surface actuators will most likely be fast micro servos that are both light and quick to respond to control signals. The drive motor system will consist of a dual motor system that allows for two counter rotating propellers to eliminate undesired rotation. Each motor will be controlled by a separate motor controller allowing for more fine grained motion control. Energy should be provide by either a power tether or an on board battery.

2) Software: The computer aided design program SolidWorks may be used to create design models for fabrication by the CSUS mechanical shop.

3) Personnel: The mock-up build will be largely assembled by Aaron Diab with help and input from the other team members. Jim Ster and undetermined ME students will be consulted for expertise during the final build.

4) Outcome: The mechanical build feature shall be deemed fully implemented when the robot is able to hover, rotate and move horizontally and vertically in low wind conditions. The robot should be capable of remaining airborne for at least five minutes at a time.

B. Feature: User Interface

The user interface will be comprised of a mobile device application that allows for control of the robot. The mobile device application front end will connect via a to-be-determined wireless technology to a control application back end that runs as a server on the robot control hardware. At a minimum the user interface will provide flight controls and sensor controls.

1) Hardware: The user interface mobile application will be implemented on Apple's iOS mobile platform and the control device used will be an Apple iPad. The wireless technology used will be determined at build time but may include an 802.11

Wi-Fi adapter or other appropriate wireless device. Besides the iPad, a computer running Apple's Mac OS X will be needed to code the UI application.

2) Software: The software needed to create the UI application includes Apple's iOS integrated development environment Xcode. Furthermore a computer running a compatible operating system (Apple's Mac OS X) must be used to write the application. The programming language used will primarily be the iOS native programming language, Objective C. The control backend may be written either in C or in Python or other languages as appropriate.

3) Personnel: The UI application front end will coded by Aaron Diab. The control backend will be coded by Rob Wortman and Aaron Diab.

4) Outcome: The user interface feature will be deemed fully implemented when there exists a mobile application and control backend that jointly provide the user a means of controlling the flight of the robot and actuating its sensors.

C. Feature: Control System

The control system will comprise a multithreaded control routine running in a POSIX environment on an ARM-based microcontroller board. Either Unix-domain sockets will be used for communication with a back-end for the user interface module, or network-domain sockets will be used to communicate directly with the front-end. The control system will communicate with the avionics stabilization module by the latter's Future Technology Devices International (FTDI) connection, and with the flight control modules by an Enhanced High-Resolution Pulse-Width Modulator (eHRPWM).

1) Hardware: The control system will run on a BeagleBone Black development board. This board is built around a Texas Instruments AM3359 microcontroller, which is a implementation of the ARM Cortex-A8 architecture specification. This board includes the aforementioned FTDI and eHRPWM interfaces. A USB/Wi-Fi adapter will provide a connection with the user interface front-end, and a USB hub will interconnect between the development board and multiple USB devices. A 5 V_{DC} power supply will provide a source for the microcontroller, the USB hub, and possibly the Wi-Fi adapter.

2) Software: The POSIX environment will be provided by Ubuntu Linux; any version 13.04 or above with the appropriate ARM and BeagleBone patches will be acceptable. The C development tools and libraries are included with the operating system distribution. The Device Tree Compiler, also included with the distribution, will be needed to configure the ARM processor's interfacing pins.

3) Personnel: The control system will be implemented primarily by Robert Wortman, coordinating with Aaron Diab, and Darrell Cahail for interfacing with their respective systems. It will take approximately 100 personnel-hours to complete this system.

4) Outcome: The control system will be considered complete when the following requirements are met:

- The platform maintains stable flight when affected by wind or physical impact.
- The flight control subsystem responds to commands relayed from user interface.

D. Feature: Cameras

A camera will be used to observe power lines, allowing the operator to find defects, failures, and environmental hazards. It will also be used to help the operator navigate, since the robot is not fully autonomous. The camera system will perform either transmission of data over wireless connection to the base station, or storing the data from camera on a data carrier on board, or combination of both.

1) Hardware: Hardware will include visual spectrum camera primarily. However, if it is necessary, and financially and technically possible, thermal-imaging (long-wavelength infrared, 8–15 μm) and ultraviolet cameras could also be introduced. In addition, a system, which will record the data from camera on the inboard carrier, would be useful. If camera has a need to rotate independently of the rotation of body of the robot, then a pan-tilt system can be introduced into design. Also, separate power supply for camera system will be used if the voltage from control electronics power supply is incompatible with cameras and their supporting circuit.

2) Software: Software for the camera system could include configuration software if the camera needs it.

3) Personnel: The camera system will be implemented in a coordinated effort by team members, including interfacing with control system, supplying the necessary power to camera, and mounting the camera to the frame.

4) Outcome: The camera system is considered complete when following requirements are met:

- Camera observes the power lines and their environments and transmit data to control system;
- Data from camera is successfully sent via transmitter and/or stored on a carrier inboard;
- Camera and supporting circuit do not interfere with other electronic systems of the robot;
- Camera and supporting hardware fit well mechanically into design and do not compromise the stability of the robot and its ability to fly and steer.

E. Feature: Attitude Sensing and Stabilization System

The purpose of this part of the project is the design of a stabilization system that is capable of being programmed and preforming navigation. This means that the design of this section of the flight control focuses mainly on the control feedback systems that allow the BeagleBone Black to control both the directional servos and the motors controlling lift through software executed on the sensor board and on the BeagleBone Black.

1) Hardware: A simplified diagram showing how the stabilization of the sphere bot is obtained is shown in Figure 1. Essentially we are proposing controlling the speed of the various motors and servos to achieve stabilization. This is achieved by taking sensor data from the Inertial Measurement Unit (IMU) and an altimeter, then having the Micro-Controller Unit (MCU - BeagleBone Black) output processed data to via a FTDI serial interface to the BeagleBone Black. The BeagleBone Black will integrate this data according to the movement detected by the accelerometer and gyroscope coupled with the movement desired.

The movement detected by the individual sensors is shown in Figure 2. Using this mechanism, it is possible to create a feedback loop to stabilize the sphere bot.

More specifically, there are three major aspects of the craft that need to be controlled using a feedback system. These aspects are altitude, rotation, and orientation.

1) The rotation of the craft shall be implemented via a control feedback system that uses a gyroscope to determine rotation and adjusts the motors and servos accordingly to keep the sphere bot from spinning out of control.

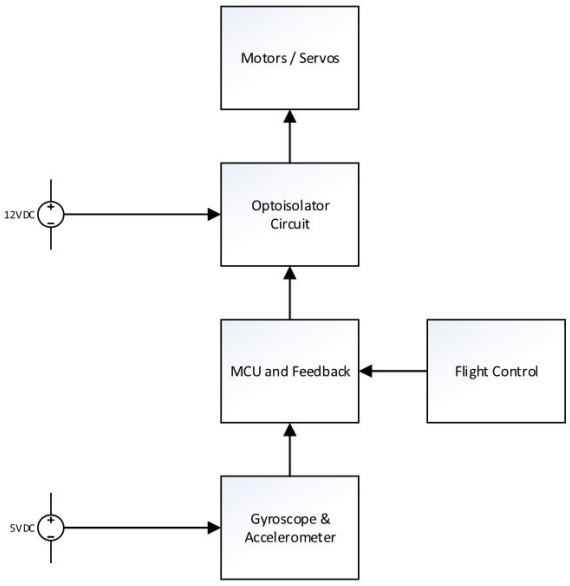


Figure 1: Block Schematic of Stabilization System

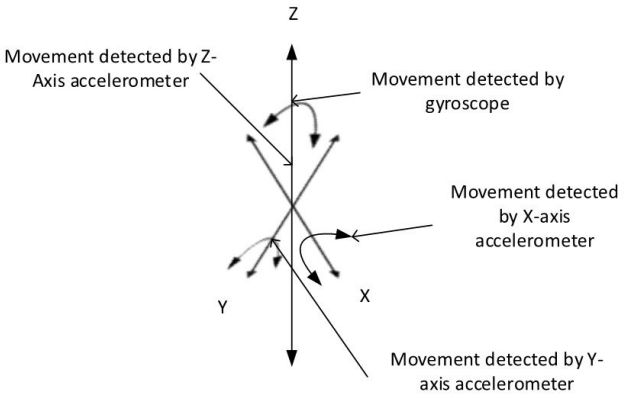


Figure 2: Dimensions of Moment Detected by IMU

2) Orientation will be controlled using a basic algorithm to determine the appropriate PWM needed on the servos and the motors to hover or achieve a specific degree of direction and rate of movement.
3) To control altitude via the z-axis acceleration.

Figure 3 briefly outlines the design of the above systems.

a) Inertial Measurement Unit: The plan is to use an off the shelf IMU with its integrated sensor and processor package to offload the main MCU from the calculation intensive part of the sensor handling. This unit will be interfaced with the MCU via a serial interface. The IMU needs to contain one triple-axis gyro, one triple-axis accelerometer, and one triple-axis magnetometer. These will feed a

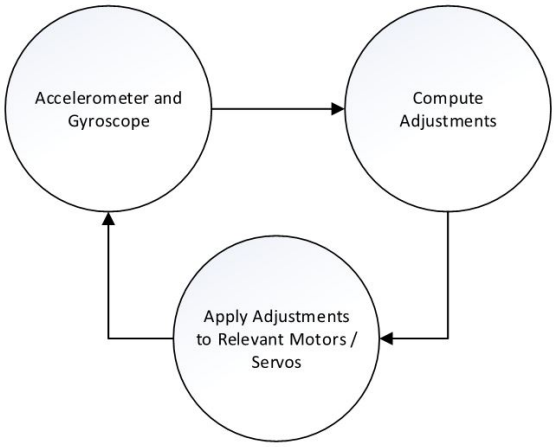


Figure 3: Feedback Path for Stabilization System

dedicated, on-board microprocessor with the output over a serial interface.

The currently identified 9 Degrees of Freedom - Razor IMU fits these specifications. Those being:

- ATmega328 microprocessor - Programmed with the MPIDE package used for the Digilent Chipkit.
- ITG-3200 MEMS triple-axis gyro.
- ADXL345 triple-axis accelerometer.
- HMC5883L triple-axis magnetometer.

b) Altimeter: The Altimeter module is BOSCH BMP085. It is high-precision, low-power barometric pressure sensor, with a measuring range from 300 to 1100 hPa with a resolution of down to 0.03 hPa (0.25 meter in height). It's based on piezo-resistive technology for EMC robustness, high accuracy and linearity as well as long term stability. This will connect and be directly monitored by the MCU. This unit has the following features and specifications.

- Wide barometric pressure range: 300–1100 hPa (9000 meters above sea level to -500 m)
- Temperature measurement included
- Ultra-low power consumption (5 μA in standard mode, and 3 μA in ultra-low power mode)
- Low noise measurement
- Fully calibrated
- Digital two wire (I2C) interface
- Flexible supply voltage range: 3.3–5 V_{DC}
- PCB Dimensions: 19$\times$19 mm
- 7 pin connection head is not soldered
- Gross Weight: 10 g

2) Software: The software will be developed in two main sections. The BeagleBone interface and the on-board IMU processor. Using the processor on the IMU, the raw sensor data will be processed into usable data that will be sent to the BeagleBone Black via a FTDI serial interface. It will then be used in conjunction with the data from the control interface to stabilize the Sphere Bot and/or control its movements.

The software IDE for programming the IMU can be downloaded from http://chipkit.s3.amazonaws.com/builds/mpide-0023-windows-20130715.zip.

Programming on the BeagleBone Black can be done on the device itself or through a cross compiler using the Eclipse IDE under Windows, Mac OS X, or Linux.

3) Personnel: The stabilization system will be implemented by Darrell Cahail.

4) Outcome: The IMU will be considered implemented in a three stage design. Each stage of the design will be considered fully implemented when its goal is achieved.

1) Design is capable of stable flight at a stationary altitude.
2) Design is capable of stable flight at variable altitude.
3) Design is capable of stable flight at variable altitude and is capable of non-vertical movement.

F. Additional resources

1) Mechanical Build Feature: Assembly of the robot will take place at team members homes, on campus in the senior design room and in the CSUS mechanical shop. Jim Ster has offered to assist with creating parts on the mechanical shop router or 3D printer devices. For the initial mock up phase of prototype design ABS plastic will be used to create the airframe and control surfaces. In the final build phase lighter materials such as fiber glass or carbon fiber will be used. A list of materials with budgeted prices for the mechanical build is included in Table I.

2) User Interface System: Coding of the user interface mobile application will be implemented by Aaron Diab on his personal Macbook Pro laptop computer. The mobile application will be installed to his personal Apple iPad mobile device for testing. The control back end will be implemented on a BeagleBone Black single board computer with

Table I: Material costs for mechanical build

Item	Cost
ABS plastic sheet	$50
ABS glue	$20
Contra rotating motor/prop assembly	$70
2 brushless motor speed controllers	$40
solderable connectors	$15
4 light weight fast servos	$70
R/C receiver and radio (for testing)	$250
hardware	$50
LiPo battery and charger	$100
Goodwill water bottle (housing)	$1
Tools (drill bits/razor blades)	$30

communication conducted over a USB 802.11 Wi-Fi dongle. The BeagleBone Black and Wi-Fi will be accounted for under the parts sections for the control system feature.

3) Stabilization System:

a) Hardware: This section of the project will initially require a place to program the boards. A bench power supply and an oscilloscope will be required to insure that the hardware is functioning correctly. Developers have purchased another BeagleBone Black and an IMU along with the FTDI serial interface so that developers can use a laptop to generate the software required.

b) Software: All required development libraries and IDEs are freely available for download on the Web.

4) Control System: Development of the control system will be performed by Robert Wortman using any handy workstation that has a SSH client, but primarily either of two Linux workstations. Material and software costs for the development of this system are included in Tables II and III. Information resources will be obtained from Texas Instruments regarding the ARM processor, from the BeagleBoard.org Foundation regarding the development board, from the Linux Kernel Archives regarding Device Tree manipulation, and from other developers as available and necessary.

Table II: Material costs for control system

Item	Cost
BeagleBone Black	$45
USB Wi-Fi adapter	$30
USB hub	$20
power supply	$10
power wiring	$5

Table III: Software costs for control system

Item	Cost
Ubuntu Linux	$0

G. Spring Revision: Camera System

A camera will be used to observe power lines, allowing the operator to find defects, failures, and environmental hazards. It will also be used to help the operator navigate, since the robot is not fully autonomous. The camera system will perform either transmission of data over wireless connection to the base station, or storing the data from camera on a data carrier onboard, or combination of both.

Due to the weight limitations in the maneuverability of the sphere-bot and the complexity of mounting a camera to the sphere-bot that can capture images of the line to positively identify faults, a line crawler will be implemented which can be dropped onto the line from an airborne system. A line crawling robot will be designed to provide additional information about the line. The combination of the frame design of the robot and the electronic clamping system allows the line crawler to be deployed onto the line from the sphere-bot. The additional camera systems on the line crawler will take images of the line that would be equivalent in resolution and clarity to what a man hanging onto the line could see from a standard visual inspection and store the images onto an SD card. In order to relate the images of the line to the physical location in the case of a fault, the line crawler will also store the location of the image being taken. Additional sensors can be mounted to the platform at the base of the robot or on the frame as needed.

1) Hardware: Hardware will include visual spectrum camera primarily. However, if it is necessary, and financially and technically possible, thermal-imaging (long-wavelength infrared, 8-15 m) and ultraviolet cameras could also be introduced. In addition, a system, which will record the data from camera on the onboard carrier, would be useful. If camera has a need to rotate independently of the rotation of body of the robot, then a pan-tilt system can be introduced into design. Also, a separate power supply for camera system will be used if the voltage from control electronics power supply is incompatible with cameras and their supporting circuit.

The hardware on the line crawler consists of the frame of the robot, the wheels and motors,

the cameras with resolution equivalent or greater than the sight of the line inspecting personnel, and the microcontrollers required to interface with the software and store data. Sensors will be implemented provide feedback to determine the location of the robot on the line. Additional hardware will be implemented as necessary to keep the line crawler stabilized on the line and enable it to be deployed onto the line.

2) Software: Software for the camera system on the sphere-bot could include configuration software if the camera needs it.

Software for the line crawler includes the programming required to control the robots movements on the line to keep it stable and prevent it from driving off the line, as well as the software to interface with the camera, or cameras, and save the camera images to an SD card.

3) Personnel: The camera system on the sphere-bot will be implemented primarily by Aaron Diab in coordination with other team members as necessary to integrate the system with the rest of the control systems on the sphere-bot.

The line crawler will be implemented by Rebecca Wingo and Emanuel Dupart in coordination with other team members to implement a system quickly that will be capable of integrating as well as time allows with the sphere-bot.

4) Outcome: The camera system is considered complete when following requirements are met by either the camera system onboard the sphere-bot or the system onboard the deployable line crawler:

1) Data from camera is successfully sent via transmitter and/or stored on a carrier onboard for later analysis;
2) Camera and supporting circuit do not interfere with other electronic systems of the robot;
3) Camera and supporting hardware fit well mechanically into design and do not compromise the stability of the robot and its ability to fly and steer;
4) Camera observes the power lines with high enough resolution to match standard human vision within one foot of the line;
5) Line crawler is capable of latching onto the line;
6) Images of the line correspond to a known distance down the line

IV. Laboratory Prototype Funding

The majority of the development expenses will be defrayed by the developers' own personal funds. Funding may be available from various grants if ambitious rework is required for project success.

V. Laboratory Prototype Schedule

September 7, 2013 Preliminary frame design created and frame materials acquisition begun.

September 14, 2013 Main cross circles and center bottle holder cut and assembled together.

September 21, 2013 Preliminary frame largely complete. Control surfaces, servos and lift motors currently being added.

September 28, 2013 Large sections of frame cut away to lighten frame.

September 29, 2013 Frame complete. Electronics installation proceeding.

October 10, 2013 IMU ready for testing. Data extremely unstable.

October 26, 2013 Most electronics installed in center bottle housing. BBB PWMs still not working.

November 9, 2013 Electronics fully installed. Servo control moved to Pololu servo controller.

November 15, 2013 Rudimentary control system created to test servo and lift motors.

November 23, 2013 Mechanical build ready for testing phase. UI development begun in earnest.

November 30, 2013 IMU instability satisfactorily resolved.

December 7, 2013 Substantially functional control system created. SphereBot too heavy. Start of Winter break.

VI. Laboratory Prototype Work Breakdown Structure

This section describes a hierarchical system of tasks which will allow the successful construction, testing, and demonstration of the robot laboratory prototype.

A. Task — Mechanical Build

The developer will construct and assemble the mechanical portions of the robot: the air- frame of the robot, the control surfaces, the control surface actuators, main motors and propellers.

1) Subtask — Build Airframe: The developer will create a spherical frame for the robot capable of housing the motor and electronic housing bottle and the control surfaces. The frame will consist of four ring shaped shaped pieces and interconnecting supports.

a) Activity — Design and cut main frame pieces: The members of the group will draw up plans for the rings which will comprise the airframe and then cut them using available tools. Other pieces necessary for construction of the airframe will be cut as needed.

b) Activity — Assemble main frame: Team members will fasten the pieces of the airframe together using glues and hardware to assemble the main body of the airframe.

2) Subtask — Create Control Surfaces: The developer will create movable flaps that will attach via hinges to the airframe capable of adjusting airflow in a manner that will allow control of the robot's flight.

a) Activity — Cut and shape control surfaces: Team members will cut control surfaces of a predetermined shape from a suitable material and process them to attain the final desired shape. A minimum of four control surfaces will be needed.

b) Activity — Install hinges and attach control surfaces: Team members will acquire suitable hinges and install them into the control surfaces and robot airframe to create movable control surfaces.

c) Activity — Install control arms: Control arms will be attached to the control surfaces to provide leverage when actuating the control surfaces.

3) Subtask — Install Control Actuators: The developer will acquire and install fast, light micro servos that will provide a means of adjusting the control surfaces position via electronic signals.

a) Activity — Acquire high speed micro servos: Team members will acquire micro servers that are light and capable of quick response to electrical signals.

b) Activity — Cut mounting holes and mount servos: Mounting holes will be cut into suitable locations in the airframe to allowing mounting of the servos.

c) Activity — Attach control arm rods: Rods consisting of ball rod ends and threaded rod will be attached to the control surface arms and servos arms to mechanically link the control surface to the servos.

4) Subtask — Install Main Motors and Propellers :

a) Activity — Acquire motor and propeller assembly: Team members will acquire an appropriate motor propeller and speed controller assembly that will provide adequate lift and run time.

b) Activity — Acquire bottle housing: Team members will acquire an appropriate housing for the motor assembly, control electronics and power supplies.

c) Activity — Modify bottle and mount motors and propellers: The previously acquired housing will be modified in such a manner as to allow convenient mounting of the motor and propeller assembly as well as control electronics.

B. Task — User Interface

1) Subtask — Create Front End:

a) Activity — Design UI graphical view: The developer will create the graphical component of the user interface that displays the necessary controls.

b) Activity — Code UI Model: The developer will create a conceptual model of the UI application that contains the data structures necessary for control of the robot.

c) Activity — Code UI view controller: The developer will program the component of the user interface that accesses the model and controls the view's interface elements.

2) Subtask — Define Back End:

a) Activity — Define command protocol: Team members involved in coding the UI and Control subelements of the robot software will draft a command control protocol that defines the methods sent by the UI and accepted by the control back-end.

b) Activity — Implement command protocol in UI Model: The UI developer will implement the command control protocol in the model sub-element of the UI. The developer will also implement the communication methods needed for conveying the commands to the robot's control backend.

C. Task — Create Control System

The developer will create a control system which comprises a multithreaded control routine running in a POSIX environment on an ARM-based microcontroller board. Either Unix-domain sockets will be used for communication with a back-end for the user interface module, or network-domain sockets

will be used to communicate directly with the front-end. The control system will communicate with the avionics stabilization module by the latter's Future Technology Devices International (FTDI) connection, and with the flight control modules by an Enhanced High-Resolution Pulse-Width Modulator.

1) Subtask — Write Command Input Module: The Command Input Module of the Control System will receive commands from the back-end of the User Interface System.

a) Activity — Develop a language of commands: The members of the group, especially those working on the User Interface System and Control System, will jointly develop a set of allowable navigation and operational instructions, and to agree on a standard syntax for communicating those demands.

b) Activity — Write Command Parser: The developer will write a routine to tokenize the contents of the stream of commands in the agreed-upon language. This may accomplished with commodity libraries or it may require a bespoke parser, depending on the the specific decisions. The commands will then be relayed to the Core Control Algorithm.

2) Subtask — Write IMU Data Tracking Module: The developer will write a routine to accept input from the IMU, in whatever format is offered by the IMU.

a) Activity — Identify IMU Interface: The developers of the Control System and the Stabilization System will coordinate to select an IMU with an interface that has sufficient bandwidth and detail for the purposes of balancing an airborne system.

b) Activity — Write IMU Polling Routine: The developer will write a routine to sample the IMU data as it is available and relay that data to the Core Control Algorithm.

3) Subtask — Write Actuator Signaling Module: The developer will produce a module to provide the necessary servo signals for the control surface actuators and the lift motor controllers. This will require interfacing with the microcontroller's integrated Enhanced High-Resolution Pulse Width Modulator (eHRPWM) units.

a) Activity — Configure Device Tree: The developer will produce runtime configuration files and scripts that will expose the eHRPWM units to a kernel pin and export the kernel pin to a hardware pin.

b) Activity — Write Per-Servo Signal Supervisors: The developer will create an output supervisor thread for each PWM signal that will receive servo parameter directives from the Core Control Algorithm and will adjust the eHRPWM unit's duty cycle accordingly, accounting for servo calibration and for desired ramping characteristics.

4) Subtask — Write Core Control Algorithm: The developer will create a module that will interface between the other control modules. This module will receive and store the desired position, motion, and attitude of the craft, as determined by the User Interface Module. The module will receive and store the actual position, motion, and attitude of the craft, as determined by the IMU Tracking Module. The module will provide input to the Actuator Signaling Module, that will minimize the error between desired and actual robotic configuration.

D. Task — Construct Camera System

The developer will construct a camera system that can observe power lines, allowing the operator to find defects, failures, and environmental hazards. The system will also be used to help the operator to navigate.

1) Subtask — Observe Power Lines:

a) Activity — Obtain camera:

b) Activity — Find power source for camera:

c) Activity — Test the signal transmission to BeagleBone unit:

2) Subtask — Transmit or Store Data:

a) Activity — Write Data Relay Module: Write code for control unit to transmit video data over the same wireless channel that is used for controlling the robot from ground station.

b) Activity — Establish Data Transmitter: If separate wireless channel is used for task, acquire and setup the transmitter.

c) Activity — Establish Data Recorder: If data is to be stored on board, acquire necessary storage device and supporting electronics.

3) Subtask — Ensure Electronic Noninterference:

a) Activity — Test Data Subsystem: Test the transmission and storage subsystem.

b) Activity — Verify Power Source: Ensure that the power source is providing enough power.

c) Activity — Verify Bandwidth: If control unit wireless channel cannot provide enough bandwidth for video, use separate wireless transmitter.

4) Subtask — Ensure Mechanical Fit:

a) Activity — Determine Dimensions: Measure the camera, supporting hardware, and the robot's frame.

b) Activity — Obtain Materials: Acquire mounting hardware and fasteners.

c) Activity — Mount Camera: Mount the camera.

E. Task — Design Stabilization System

The developer will design and construct a stabilization system that will provide inertial, magnetic, and barometric telemetry so that the overall system can maintain its position and orientation.

1) Subtask — Roll, Pitch, and Yaw Control: In the flying sphere yaw control is going to be an extremely important aspect of the stabilization system. It is going to compensate for any unwanted rotation about the yaw axis. This is the axis that passes vertically up and down through the center of the sphere. There is a large tendency for the sphere to want to rotate about this axis because of the rotation of the lift motors that make the flight possible.

Roll and Pitch control for stabilization come next. The Roll control is the rolling of the sphere toward the left or the right about its horizontal plane. Unwanted roll causes the sphere to drift left or right, making hovering impossible. Pitch is the tipping of the sphere forward (down) or backward (up). Unwanted pitch causes the sphere to drift to the front or back.

Combinations of these three modalities can make for some complex movements of the craft. They also make it difficult to stabilize when hovering or in flight.

2) Subtask — Altitude Control (Lift): Altitude control is exactly what it says. This is what control the power to the lift motors for rising, falling, or maintaining a hovering stance.

3) Subtask — Compass Heading Control: A compass is the sensor that allows the craft to detect and maintain a heading using the earth's own magnetic field. This same sensor can be exploited to detect the distance from a wire carrying an electric current. This can help in the guidance of the sphere when inspecting live power lines.

4) Subtask — Intersystem Communications (Motors and Servos): Lastly, these systems are not useful independently. A communication system that combines the data from these systems and feeds it to the motor and servo control system is required.

F. Task — Final Prototype Assembly

The developers will assemble and test the components completed components.

1) Subtask — Coordinate integration: The developers will coordinate to efforts to integrate the completed subsystems. This will be performed concurrently with the development of the subsystems, as most of them are tightly integrated with one or more other subsystems.

2) Subtask – Test and evaluate: The developers will test the completed system and ensure that the system behaves as described in the Design Contract.

G. Task — Administration

In addition to the directly productive work required, the developers will perform various administrative tasks and follow industry-standard project management procedures, as required to complete the project.

1) Subtask — Project Management: Under the direction of the project manager (i.e. the instructor), the developers will produce reports and give presentations describing completed tasks, current status, and projected work. This section describes actions to be taken during the first half of the project, with actions reserved for the second half to be added later.

a) Activity — Problem Statement and Elevator Pitch: The developers will identify an existing societal problem, analyze the potential for an engineering solution to that problem, and present their findings to their peers for critique. [15]

The developers will review available peer-reviewed journals and similarly-qualified literature in order to explore the subject. No fewer than four such sources will be used in this research.

The developers will create a report of no less than four pages describing the identified societal problem and the potential engineering solution to that problem.

Before an assembly of peers, the developers will present a succinct Elevator Pitch describing the project. This presentation will then segue into an appropriately detailed description of the societal problem to be solved, and the state of existing engineering solutions that attempt to address that problem.

b) Activity — Design Idea Contract - Project Proposal: The developers will draft a proposal

for an engineering solution to the societal problem identified in the earlier Problem Statement. [16]

The developers will devise and describe a solution to the identified societal problem. The solution must be an engineering solution, as opposed to a social or political solution.

The developers will identify the primary features of the proposed solution. For each feature, the developers will describe the hardware and software required to implement that feature, identify the person responsible for implementing that feature, and describe the feature as a deliverable.

The developers will describe the members of the development team, with a focus on existing skills which are vital to the project. For each member of the team, a resumé will be included.

The resulting proposal will be presented to the project manager, and will be reworked until approved by said project manager.

c) Activity — Work Breakdown Structure: The developers will create a report detailing the component activities required to complete the project. This hierarchical arrangement of activities will be inspired by the Work Breakdown Structure [17] and the Program Evaluation and Review Technique [18], [19].

The developers will identify the tasks, subtasks, and activities required to complete the project. Each lower division will describe the required actions in increasing detail and specificity. These components will be compiled into a hierarchical report describing the project as an arrangement of related activities.

Before an assembly of peers, the developers will present the Structure report. Based on feedback from this peer review, the developers will rework the Structure report. Based on feedback from the project manager, the developers will continuously rework the Structure report throughout the duration of the project.

d) Activity — Project Timeline: From the tasks described in the Structure report, the developers will create a project timeline that illustrates the expected progression of the project. Using this timeline and other bric-á-brac, the developers will decorate the bulletin board at their workcenter in the shared workspace. [20]

Using Microsoft Project, the developers will create a Gantt chart that includes all activities required for development through the end of the project. For each shown task, a responsible developer will be identified. This chart will be updated continuously as the project progresses.

On the bulletin board at the team's workcenter, the team will post the Gantt chart, and replace the chart as it is updated. The team will also post the names and pictures of the individual developers, a description of the project, and sufficient material to "tell a story" regarding the project. Team members will update and modify this display continuously as the project progresses.

e) Activity — Bread Board Proof: The developers will assemble and demonstrate a proof-of-concept to illustrate the intended design. [21]

The team will assemble a rapid prototype constructed of available materials in order to demonstrate the viability of the design concept. This prototype will illustrate the expected development of all major subsystems.

Before an assembly of peers, the developers will present the Bread Board Proof for consideration. Based on feedback from peers and from the project manager, the developers will modify their design to ensure successful completion of the project.

f) Activity — Laboratory Prototype: The team will assemble a laboratory prototype constructed of materials and components appropriate to the features included in the Design Contract. This prototype will incorporate all major subsystems. The team will conduct a progress presentation during week 11 [22] and a final presentation during week 15 [23][24].

Before an assembly of peers, the developers will demonstrate progress on the laboratory prototype. Based on feedback from peers and from the project manager, the developers will identify strategies for completing this prototype.

Before an assembly of peers, the developers will demonstrate a completed laboratory prototype which implements all features described in the Design Contract. The presentation will show how the complete prototype addresses the stated societal problem, and will be accompanied by a programme and a slideshow.

2) Personnel Management: The development team will also take actions as necessary to coordinate their efforts and maximize their return on invested labor.

a) Activity — Leadership: The team will designate a leader, who shall rule with an iron fist. At the beginning of week 9 of each half and at the beginning of the second half, the team will conduct

a change-of-command such that each team member serves as leader for an eight-week period.

b) Activity — Progress Reports: At the end of each week, each team member will prepare an individual weekly progress report describing the tasks performed, the status of completion for those tasks, and the number of hours worked per task. These reports will be shared on the Hive collaboration platform and collated by the team leader into a group weekly progress report.

c) Activity — Recordkeeping: Each team member will maintain a log documenting actions taken, the reasons for those actions, references used, and ideas considered. Photographic documentation and design sketches will be generated as appropriate, in anticipation of use in reports and presentations. Datasheets will be kept for reference and citation.

d) Activity — Evaluations: In concurrence with the change-of-command events, the retiring team lead will produce an Outgoing Team Leader Report. At twelve-week intervals, each team member will perform peer evaluations for each other team member.

e) Activity — Professional Development: Team members will attend professional development sessions, as scheduled by the project manager.

H. Summary of Work

A total of 803 hours of work has already been applied during the fall semester to the progress of this project to date. Approximately the same amount of work is anticipated during the spring semester before the project reaches its final stage of completion.

Table IV: Work performed and anticipated, by task, Fall 2013

	Hours	
Task	**Performed**	**Anticipated**
Mechanical Build System	60.5	90
Control System	159	180
Stabilization System	154	50
User Interface System	17.5	30
Camera System	13.5	200
Administration	398.5	400
Total	803	950

I. Risk Assessment

1) Risk Identification
   a) Inability to complete major subtask.
      i) Inability to complete Airframe.
      ii) Inability to complete the control system.
      iii) Inability to complete stabilization system.
      iv) Inability to complete UI.
      v) Inability to complete camera subsystem.
   b) Damage to hardware.
      i) Damage to Airframe.
      ii) Damage to propellers/motors.
      iii) Damage to BBB.
      iv) Damage to IMU.
      v) Damage to camera.
   c) Loss of software.
   d) Loss of software for control system.
   e) Loss of software for IMU.
2) Possible Causes of Failures
   a) Potential causes of inability to complete subtasks
      i) Human failure
         A) Loss of Team member due to sickness or death.
         B) Inability of team member to complete assigned task.
      ii) Damage to hardware.
         A) see above
      iii) Loss of software.
         A) see above
   b) Potential causes of hardware damage
      i) Damage to Airframe.
         A) Crashing the robot.
         B) Electrical shorts/fire.
         C) Children.
         D) Dropping/kicking/mangling bot.
      ii) Damage to propellers/motors.
         A) Crashing the robot.
         B) Electrical shorts/fire.
         C) Children.
         D) Dropping/kicking/mangling bot.
      iii) Damage to BBB.
         A) Electrical shorts/fire.
         B) Dropping/kicking/mangling BBB.
      iv) Damage to IMU.
         A) Electrical shorts/fire.
         B) Dropping/kicking/mangling IMU.
      v) Damage to camera.
         A) Electrical shorts/fire.
         B) Dropping/kicking/mangling camera.
   c) Potential causes of software loss

i) Loss of software for control system.
A) Loss due to deletion.
B) Loss due to incorrect changes made.
ii) Loss of software for IMU.
A) Loss due to deletion.
B) Loss due to incorrect changes made.

3) Mitigation plans
a) Mitigation strategy for inability to complete subtasks
i) Cross train team members on tasks.
ii) Seek outside assistance if task problems can not be surmounted.
b) Mitigation strategy for damage to hardware
i) Keep replacement parts on hand for breakable items.
ii) Check all connections before powering up circuits.
iii) Keep fragile parts/assemblies out of the way so they are not dropped/kicked/mangled.
iv) Keep hardware out of reach of children.
c) Mitigation strategy for software loss
i) Make backups of software to sites accessible by all team members.
ii) Implement a version control system accessible by all team members.

VII. Laboratory Prototype Task Assignments

Aaron Diab was primarily responsible for tasks relating to the SphereBot Mechanical Build and SphereBot User Interface Systems. Robert Wortman was responsible for the SphereBot Control System. Darrell Cahail was primarily responsible for the SphereBot Stabilization System. Another developer was responsible for the SphereBot Camera System, but little work was put into that aspect before the Design Idea change. Naturally, there was a considerable amount of cross-task work, especially where the systems interfaced. Additionally, all developers were responsible for reports, presentations, and assorted administrative tasks. The time worked on each of these tasks is broken down by team member in Table V.

VIII. Deployable Prototype Funding

The majority of the development expenses continue to be defrayed by the developers' own personal funds. A grant from Intel Corporation made possible the water jet cutting required for successful completion of the third-generation, carbon fiber frame.

IX. Deployable Prototype Schedule

February 1, 2014 SphereBot Laboratory Prototype complete. Prototype too heavy to fly.

February 8, 2014 Mechanical Engineering student found to help with SphereBot frame redesign.

February 14, 2014 The final decision to create a line crawler to observe the line was made and development planning began.

February 15, 2014 New SphereBot frame initial design CAD drawings done. Lab prototype will be presented at AIAA.

February 26, 2014 Rob began SphereBot control system redesign. Messaging queue created.

February 21, 2014 The prototype frame of the line crawler was completed.

February 22, 2014 The project was presented at the AIAA conference, hosted at California State University, Sacramento.

February 24, 2014 The line crawler programming was completed and the electronics were mounted to the frame so that the line crawler could travel down the simulated transmission line.

March 1, 2014 UI on iPad connects to SphereBot over Wi-Fi. Control system UI module development started.

March 3, 2014 A camera was added to the line crawler and images could be acquired through a computer interface.

March 15, 2014 New frame parts have been cut and are ready for assembly. New IMU installed into SphereBot.

March 17, 2014 The line crawler met each basic feature, including deployment, traversing the line, and image acquisition.

March 29, 2014 New SphereBot frame largely complete. Electronics installation begun.

April 5, 2014 New SphereBot frame complete. Electronics sled still not done.

April 19, 2014 SphereBot electronics sled complete. Next generation control system ready for testing. IMU data unstable.

April 26, 2014 SphereBot UI refined. Mechanical vibration found to be at fault for unstable IMU data.

May 1, 2014 Filter enabled in SphereBot IMU.

Table V: Hours worked during Fall 2013, by team member

Task	Hours		
	Robert Wortman	**Aaron Diab**	**Darrell Cahail**
SphereBot Mechanical Build	12	118	
SphereBot Control System	134.5	24	67
SphereBot Stabilization System		8	164.5
SphereBot User Interface System		11.5	
SphereBot Camera System		5	
Reports and presentations	114.5	56	
Assorted administration	76.75	90	
Total	337.75	312.5	231.5

Stability improved for at rest state. IMU still unstable when motors ran.

X. Deployable Prototype Work Breakdown Structure

This section describes a hierarchical system of tasks which will allow the successful construction, testing, and demonstration of the robot deployable prototype.

A. Task — SphereBot Mechanical Build

1) Subtask — Design Carbon Fiber frame:

a) Activity — Find assistance for carbon fiber build: The developer will seek the assistance of the tech shop in identifying a mechanical engineering student that will be able to assist the team in creating a lighter carbon fiber frame with the laboratory prototype frame as a model.

b) Activity — Take Measurements of preliminary frame: The developer will measure the laboratory prototype frame and record the measurements for use in designing the new frame.

c) Activity — Design new frame: The developer will work with the previously identified mechanical engineering student to draft CAD drawings to be used in the new frame design.

2) Subtask — Build Frame:

a) Activity — Manufacture carbon fiber sheets: The developer will work with mechanical engineering students to manufacture the carbon fiber sheets that will compose the new frame.

b) Activity — Have parts cut : The developer will send the previously manufactured carbon fiber sheets and the CAD drawings of the frame parts to a qualified facility to have the parts precision cut.

c) Activity — Assemble frame from parts: The developer will work with the assisting mechanical engineering student to assemble the main body of the new frame from the precision cut parts.

d) Activity — Attach servos: The developer will work with the assisting mechanical engineering student to attach the servos to the main body of the new frame.

e) Activity — Attach lift motors: The developer will work with the assisting mechanical engineering student to attach the lift motors and propellers to the center tube of the frame.

f) Activity — Attach control surfaces: The developer will work with the assisting mechanical engineering student to assemble and attach the control surface to the new frame.

3) Subtask — Build Electronics Sled:

a) Activity — Attach electronics to sled: The developer will attach all internal electronics, including the battery, to the electronics sled.

b) Activity — Attach bottom disk: The developer will attach the bottom carbon fiber sandwich disk to the electronics sled.

c) Activity — Attach IMU to sled: The developer will attach the inertial measurement unit to a holder at the top of the sled and attach the associated wiring.

d) Activity — Ensure fit: The developer will ensure that the sled and electronics fit well into the new frame's center tube.

B. Task — SphereBot User Interface

1) Subtask — Create network connection to SphereBot:

a) Activity — Write code to create network socket: The developer will write a function to establish a TCP connection to the SphereBot control system given an IP address and port number. The developer will also write a function to disconnect from the SphereBot control system.

b) Activity — Add connection text input boxes and buttons: The developer will add the user interface elements to allow inputting the SphereBot

IP and port number and to initiate connection and disconnection.

2) Subtask — Create data display: The developer will add user interface elements to display pertinent data to the user such as current throttle setting and connection status.

3) Subtask — Create thumb joystick: The developer will write code to create a custom user interface element that works as a thumb joystick that controls the pitch and roll attitudes of the SphereBot.

4) Subtask — Create Yaw and Throttle sliders: The developer will add throttle and yaw sliders to the user interface and write the code for the sliders to allow them to set the throttle and yaw of the SphereBot.

5) Subtask — Create Kill Switch: The developer will add a user interface element and write the code to implement a kill switch that causes the SphereBot control system to exit and powers off all servos and lift motors.

C. Task — SphereBot Control System

The responsible developer will complete the control system. The architecture of the control system will be refactored with a message queue architecture in order to allow single points of communication between modules, allow a single point of waiting within a module, and to avoid the need for busy-waiting or unnecessary polling. The unfinished portions of the system will be completed, and the whole system will undergo integrated testing and refinement.

1) Subtask — Create Message Queue: The developer will create a message queue that will provide a clean, robust interface between modules. This message queue will be internally protected by a mutex and signaled by a condition variable, and will be provided with an abstract programming interface to eliminate code clutter and redundancy.

2) Subtask — Write User Module: The developer will write the code to implement a user interface module for the SphereBot's onboard control system that receives control data from the user interface running on the iPad and sends that data to the core control module.

3) Subtask — Write Stabilization Module: Likewise, the stabilization module is complete and adequately relays information from the Stabilization System. The developer will integrate it into the refactored architecture of the control system, test its operation, and refine as necessary.

4) Subtask — Write Core Control Module: The developer will transition the core module to the message queue architecture in order to allow both input modules to communicate concurrently. The developer will add the required PID controllers necessary to direct the electromechancial portions of the robot.

a) Activity — Refactor Module Architecture: The developer will use the message queue to coordinate reading from the input modules, processing the input data, and providing the resulting control signals to the appropriate actuators.

b) Activity — Write PID Controllers: The developer will write a set of PID controllers that will maintain the stability of the robot while in flight, as well as respond to control inputs from the user.

D. Task — SphereBot Stabilization System

1) Subtask — Establish Yaw Indication: In the flying sphere yaw control is going to be an extremely important aspect of the stabilization system. It is going to compensate for any unwanted rotation about the yaw axis. This is the axis that passes vertically up and down through the center of the sphere. There is a large tendency for the sphere to want to rotate about this axis because of the rotation of the lift motors that make the flight possible.

2) Subtask — Establish Pitch and Roll Indication: Roll and Pitch control for stabilization come next. The Roll control is the rolling of the sphere toward the left or the right about its horizontal plane. Unwanted roll causes the sphere to drift left or right, making hovering impossible. Pitch is the tipping of the sphere forward (down) or backward (up). Unwanted pitch causes the sphere to drift to the front or back.

3) Subtask — Establish Altitude Indication: Altitude control is exactly what it says. This is what control the power to the lift motors for rising, falling, or maintaining a hovering stance. When this value is combined with yaw it is possible to control vertical velocity.

4) Subtask — Establish Magnetic Compass Indication: A compass is the sensor that allows the craft to detect and maintain a heading using the earth's own magnetic field. This same sensor can be exploited to detect the distance we are from a wire carrying an electric current. This can help in the guidance of the sphere when inspecting live power lines. This sensor is also used in the yaw sensor fusion.

5) Subtask — Establish Serial Communications: These systems are not useful independently. A communication system that combines the data from these systems and feeds it to the motor and servo control system is required.

E. Task — DOC Frame Design

The developer will design and build a frame capable of being deployed onto a transmission line from an aerial device.

1) Subtask — Build Frame: The developer will create a platform capable of traveling down a transmission line after wheels are attached.

a) Activity — Design frame: The developer will draw out the frame of the line crawler in such a way that the drawings can be taken to the ECS Tech shop to be manufactured. Designs may be tested on foam board before being implemented. Assistance from the ECS Tech shop is recommended.

b) Activity — Build Frame: The developer will work with the ECS Tech shop in cutting out the initial frame and forming it into a warm spot.

2) Subtask — Install Propulsion Electronics: The developer will install the electronics that allow the line crawler to traverse the transmission cable.

a) Activity — Install wheels: Continuous rotation servos will be mounted into the pre-cut holes in the frame and wheels will be attached.

b) Activity — Install electronics on platform: Batteries, a chipKIT uC32, and a motor shield will be mounted into the frame at the base of the line crawler.

3) Subtask — Install Deployment System: A system will be installed which will provide feedback for the state of the line crawler in relation to its deployment status.

a) Activity — Design Deployment system: The developer will choose sensors which will allow the microcontroller to correctly interpret it's deployment status

b) Activity — Implement Deployment system: In which, the developer will build and implement the deployment design previously decided on.

4) Subtask — Test Frame: The developer will test the frame's ability to be deployed onto the transmission line and traverse it while getting clear images.

a) Activity — Test Individual Components: Test each new part as it arrives for functionality with the new servo.

b) Activity — Test Frame: Combine the propulsion system and the electronics before running the frame down the cable is to verify it is to verify the ability of the robot to smoothly travel down the transmission line.

F. Task — DOC Control System

The developer will implement a control system to propel DOC down the transmission line without user input.

1) Subtask — Propulsion System: The developer will implement a system to propel the line crawler smoothly down the transmission line.

a) Activity — Wheel Calibration: The developer will program previously-installed tires to run a set distance down the simulated line and return without user input.

b) Activity — Ramp Stabilization: The developer will modify the previously written code to ramp the speed up and down to prevent the line crawler from rocking.

c) Activity — Deployment Integration: The developer will integrate the ability to sense when the line crawler has been deployed onto the line to activate the propulsion system without user input.

2) Subtask — Pole Sensing: After the propulsion system is stable, a sensor will be implemented by the developer which will provide the line crawler with the ability to stop once it gets within a set distance of the power pole.

a) Activity — Sensor Mount: The developer will mount a sensor to the robot in such a way that is capable of sensing an oncoming pole.

b) Activity — Sensor Integration: The developer will modify the propulsion system code to allow the system to stop when DOC is a set distance away from the pole.

G. Task — Latch System

The developer will attach a high-torque servo with a latch arm to provide additional security for the line crawler on the line.

1) Subtask — Latch Mechanical Build: The developer will design, build, and attach a latch system to the previously built line crawler frame strong enough to hold the robot onto the cable.

a) Activity — Prototype Latch: The developer will build a prototype latch out of a cheap, easy to manipulate material to find a design that will provide a feasible connection between the transmission cable and the main frame of the line crawler and find a way to mount the latch servo.

b) Activity — Final Latch: The developer will implement the design of the prototype build using a stronger, more durable material.

2) Subtask — Latch Programming: The developer will program the latch to open as DOC is being lifted off the line, closed while the line crawler is on the line, and open as it is being deployed onto the line.

a) Activity — Latch program: The developer will write a code to control the state of the high-torque latch servo based on the state of a switch.

b) Activity — Integration: The developer will implement the latch code in the previously written code.

c) Activity — Testing: The team will test the code to insure the integration did not cause other errors and to test the ability of the line crawler to stay on the transmission cable with the latch system.

H. Task — System Testing

DOC's developers will test the system to insure it functions as required by the specifications.

1) Subtask — PVC Test: The developers will test the ability of the line crawler to go up and down 3" PVC pipe, which acts as a simulated transmission cable.

a) Activity — Straight Cable: For this test, the basic functions of the control system and the frame are tested as DOC is run back and forth on the pipe.

b) Activity — Angles: This test is conducted to see if DOC can handle going up or down angles. The developers will vary the incline and not the changes in the ability of the line crawler to go up the incline.

c) Activity — Bumps: The developers will test the ability of the line crawler to go over lumps on the simulated line.

2) Subtask — Textured Simulated Cable: The developers will test DOC's ability to travel down different types of simulated cable.

a) Activity — Develop Cable: The developer will construct a simulated cable from a different material

b) Activity — Retest Line Crawler: The developer will note DOC's performance on the line.

I. Task — Administration

The team will continue to perform various administrative tasks and follow project management procedures, as required to complete the project.

J. Summary of Work

A total of 881.75 hours of work were applied during the Fall semester to the progress of this project. During the Spring semester, an additional 1662.25 hours were applied, for a total of 2544 hours.

Table VI: Work performed, by task, whole project

	Hours	
Task	**Fall 2013**	**Spring 2014**
SphereBot Mechanical Build	130	77
SphereBot Control System	225.5	234.5
SphereBot Stabilization System	172.5	155
SphereBot User Interface System	11.5	22
SphereBot Camera System	5	
DOC Frame		37
DOC Attachment System		35
DOC Vision System		192.25
DOC Control System		108
Reports and presentations	170.5	569.75
Assorted administration	166.75	231.75
Total	881.75	1662.25

K. Risk Assessment

The identified risks, likely causes, and appropriate mitigation plans remain largely the same as they were during the Laboratory Prototype phase, as outlined in Section VI-I.

XI. Deployable Prototype Task Assignments

Aaron Diab remained primarily responsible for tasks relating to the SphereBot Mechanical Build and SphereBot User Interface Systems. Robert Wortman remained responsible for the SphereBot Control System. Darrell Cahail remained primarily responsible for the SphereBot Stabilization System. For the Deployable Observational Crawler, Rebecca Wingo was responsible for the DOC Frame and for the DOC Attachment System, Emmanuel Dupart was responsible for the DOC Vision System, and responsibility for the DOC Control System was shared between Rebecca Wingo and Emmanuel Dupart. There continued to be a considerable amount of cross-task work, and all developers remained responsible for reports, presentations, and assorted administrative tasks. The time worked on each of these tasks is broken down by team member in Table VII.

Table VII: Hours worked during Spring 2014, by team member

Task	Hours				
	Robert Wortman	**Aaron Diab**	**Darrell Cahail**	**Rebecca Wingo**	**Emmanuel Dupart**
SphereBot Mechanical Build	6.5	70.5			
SphereBot Control System	91.5	30	113		
SphereBot Stabilization System	0.5	10.5	144		
SphereBot User Interface System		22			
DOC Frame				32	5
DOC Attachment System				35	
DOC Vision System		10		41	141.25
DOC Control System				49	59
Reports and presentations	179	83		175	132.75
Assorted administration	54.5	77.5		25	74.75
Total	332	303.5	257	357	412.75

XII. Market Review

The developers attended the 2014 American Institute of Aeronautics and Astronautics Region IV conference, hosted at California State University, Sacramento. At the conference, the developers presented their work before a panel of judges and an assembly of peers, and solicited feedback from the same.

A. Networking Feedback

The feedback we received during the informal networking sessions provided for during the AIAA conference was generally positive with a couple of constructive suggestions provided. Of the other comments that we received that most constructive was from one attendee that suggested we might consider using a material called G10 for the new frame due to the fact that carbon fiber is conductive. It was suggested that the conductivity of carbon fiber might however serve to shield our electronics to some degree. Many of conference attendees showed interested in the design and one went so far as to request that we present the SphereBot to an UAV club that he currently is a member of. Another commenter suggested that we without fail approach utility companies with the SphereBot because the commenter believed that the utility companies would be keenly interested in employing it. The intense interest expressed by the conference attendees that we spoke to demonstrates to some extent that there will be demand in the market for a fully functional SphereBot capable of performing inspections with the addition of the line crawler.

B. Judges Feedback

The most prominent question asked by a judge was what would happen if a component on the SphereBot failed to which the answer was the SphereBot would likely be crash and be damaged. The judge expressed the opinion that redundancy should be built into the SphereBot so that a failure of any one component would not be catastrophic. We responded that the cost of losing a SphereBot was greatly overshadowed by the cost incurred when a line worker is either injured or killed. The judge remained unswayed in his opinion. It was also asked how much the SphereBot cost to which the answer was between 200 and 300 dollars. According to one team member a judge questioned the accuracy of that sum.

The formal feedback from the judges was primarily concerned with the quality of the presentation and the paper and did not address the quality or marketability of the SphereBot design. The formal feedback did in several places request more data showing the operating characteristics of the SphereBot. Unfortunately not having completed our device test plan our paper was lacking in test data that would paint a greater picture of the utility offered by the SphereBot.

XIII. SphereBot User Manual

A. Introduction

High tension power lines are the backbones of electrical energy distribution systems. As such they must be diligently maintained. In many cases this maintenance must be performed by personnel in direct contact with the power lines. This kind of inspection is inherently dangerous. The SphereBot is designed to assist in the inspection of power lines to mitigate the danger posed to workers.

B. Description of SphereBot

As its name suggests the SphereBot is a spherically shaped airborne robot. By attaching a camera (sold separately) to the SphereBot and guiding it near the power lines to be inspected a ground based operator can obtain pictures or video that can be used to diagnose line faults. Additionally the SphereBot can be used to deploy and retrieve appropriately designed line crawler robots (also sold separately) to more closely inspect lines when needed. Interfacing with the SphereBot is quick and easy using industry standard wireless communication protocols. The user interface program itself runs on the included Apple iPad device. Access to the internal electronics, including the onboard battery, is accomplished by removing the electronics sled from the center of the SphereBot.

C. Figures of SphereBot showing major components

The major components of the SphereBot are illustrated in Figure 4. The User Interface is detailed in Figure 4a, and the Mechanical Build is detailed in Figure 4b.

D. Setting up the SphereBot

1) Charging the battery: The SphereBot comes with a 35C 11.1V Lithium Polymer and the associated charger. Before operating the SphereBot please fully charge the battery. The battery can be charged by connecting the power and balance connectors to the charger and selecting the LiPo Balance program. Charging takes approximately one hour.

2) Putting the battery in the sled: Once the battery has been charged insert in into the battery slot on the electronics sled with the connector wires pointing down on the battery connection side of the sled as in the Figure 1a. Connect the connector and secure the wiring with light packing tape as in Figure 2b.

3) Putting the sled in the SphereBot: Turn the SphereBot over and feed the servo wires first into the center cavity as in Figure3. Gently insert the sled into the center cavity and slide it in until the bottom disk is flush with the bottom of the central housing. Do not force the sled in or you will break the electronic components.

4) Connecting the servo wires: Once the sled is fully inserted you may connect the servo wires. The sled side servo wires are labeled 2 through 5. The servos themselves are numbered 2 through 5 starting with the left front servo and proceeding clockwise. Please attach the servo wires accordingly, making sure that the black wires from the servo and the sled are lined up.

5) Connecting the Motor wires: The motor and electronic speed controller wires are color coded. Using two pairs of hemostats or small pliers reach inside the access holes, being careful not to damage the IMU, and connect the motor wires accordingly.

6) Setting up the Wi-Fi router: The Wi-Fi router comes preconfigured and must be plugged in and active before the SphereBot is powered on. A good way to check if the router is ready to go is to connect to it first with iPad. Once the iPad can connect to the router you are ready to power on the SphereBot.

E. Powering on the SphereBot

1) Where to find the on switch: The SphereBot's main power switch is found in the center of the forward access hole. Slide it to the left to power on the SphereBot.

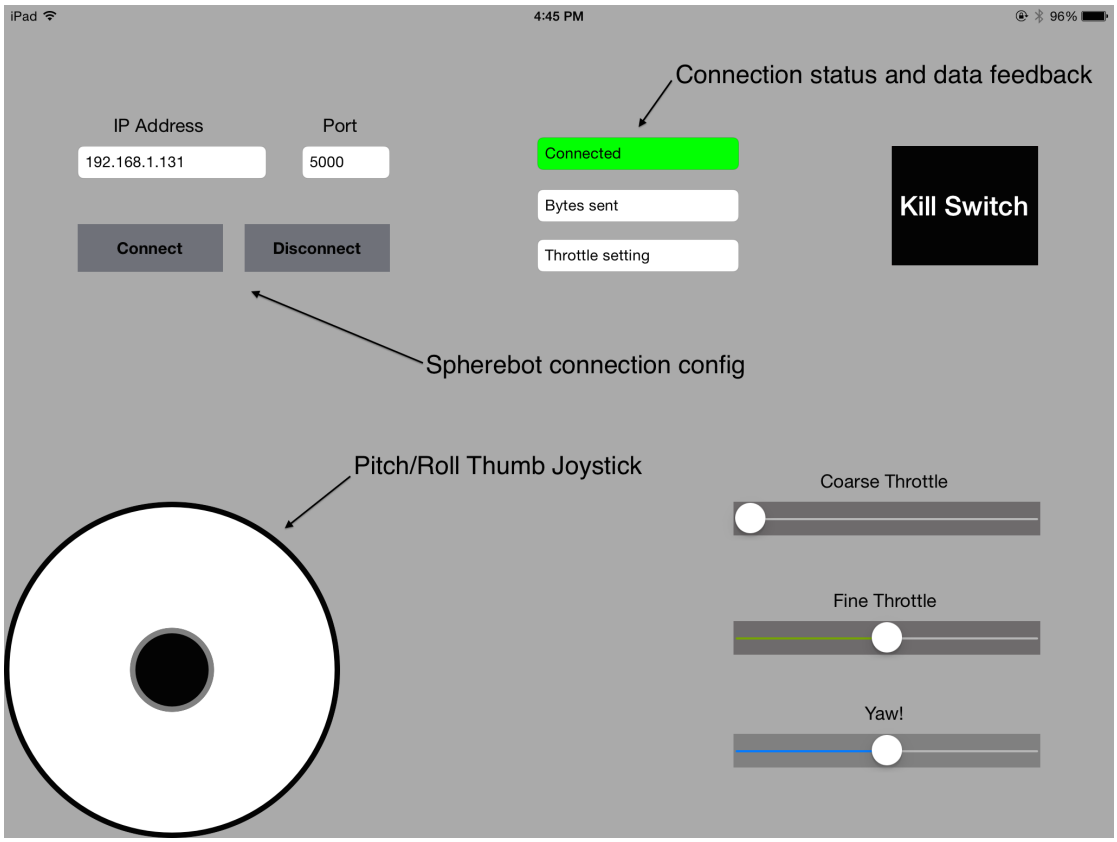
2) Ensuring motor ESCs are cal'd: Immediately after powering on the SphereBot the electronic speed controllers are calibrated. A proper calibration sequence is indicated by a series of three beeps rising in tone followed by three beeps of the same tone, then a short delay, and ending in a single beep. If that sequence of beeps is not heard the SphereBot has failed to calibrate the speed controllers.

3) Wait until SphereBot is fully booted: Approximately thirty seconds to a minute after power on the SphereBot onboard computer should be fully booted and it is time to log into the SphereBot and start its control system.

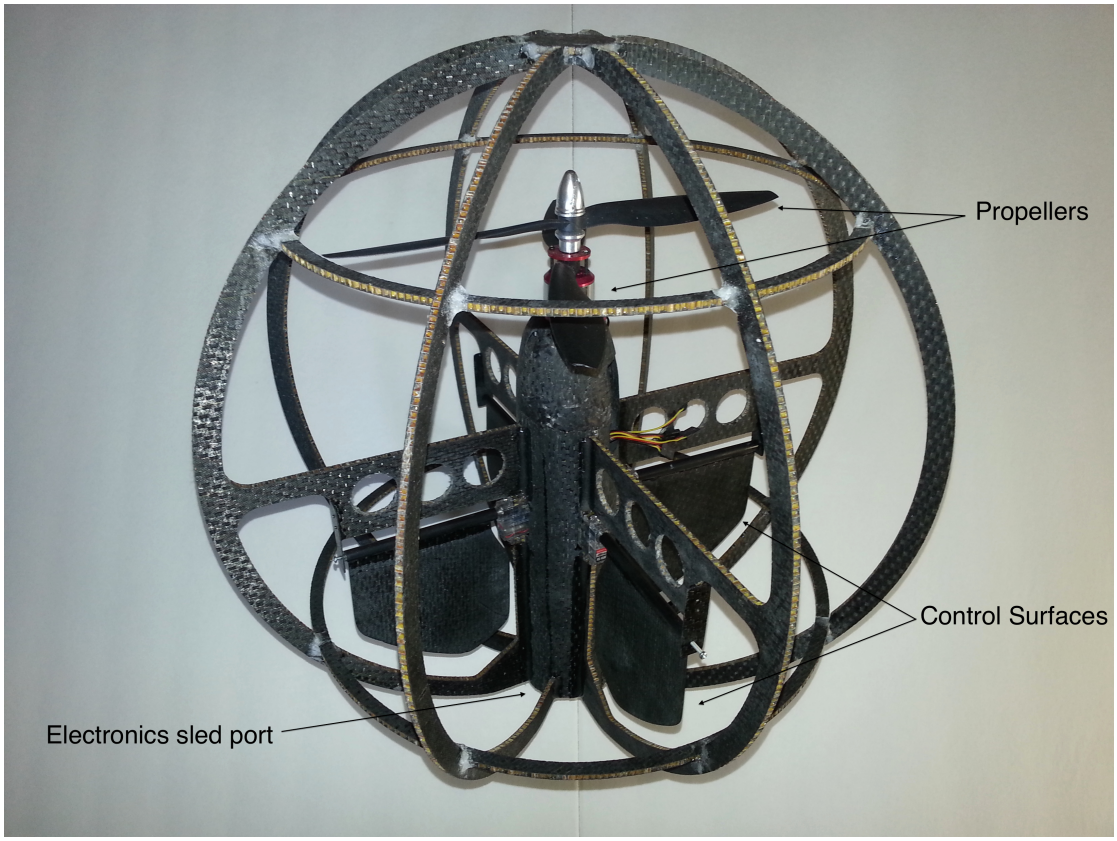
4) Logging in to SphereBot and starting the control system: Logging into the SphereBot is accomplished using the WebSSH application installed on the iPad. Simply tap the WebSSH icon and choose the preconfigured connection for the SphereBot. Once you have logged in change to the SphereBot directory and start the control system by running the ./SphereBot command.

F. The SphereBot User Interface

1) Overview: The User Interface is illustrated in Figure 4a. The connection configuration and controls are at the top left, the status and data feedback



(a) User Interface major components



(b) Mechanical System major components

Figure 4: Major components of the SphereBot

are at top center, and the safety kill switch is at top right. In the bottom left is the thumb joystick used to control the pitch and roll. And, on the bottom right are the sliders used to control lift and yaw.

2) Connecting to the SphereBot: After you have started the control system you can use the user interface app on the iPad to connect to the SphereBot. Simply tap on the SphereBot icon, enter the IP address and port number of the SphereBot you want to control, and tap connect.

3) Controlling the SphereBot:

a) Getting airborne and changing vertical position: Once you have connected to the SphereBot with the iPad user interface it is time to get airborne! With the SphereBot propellers pointing away from the ground increase the throttle by sliding the coarse throttle slider to the left until the SphereBot is airborne and then throttle back until the SphereBot is hovering or is lift neutral. Once the SphereBot is hovering you can use the fine throttle slider to move up and down vertically. If you let go of the fine throttle slider, or slide of the edge, the slider button will go back to center and the SphereBot will return to hovering. Do not run the coarse throttle up to full powering when taking off or your SphereBot will take off like a rocket!

b) Forward, backward and side to side movement: The thumb joystick on the left side of the user interface controls forward, backward, and side to side translation. Slide the thumb stick forward to move forward, backward to move backward and to the right and left to move in those directions respectively.

c) Rotating the SphereBot: Rotating the SphereBot on its vertical center axis is accomplished using the Yaw slider in the lower right of the user interface. Slide the slider to the right to rotate right and left to rotate left.

4) Shutting down the SphereBot: The SphereBot onboard computer's filesystem can be damaged by improperly shutting down the SphereBot. Before powering off the SphereBot, with the SphereBot safely on the ground, tap the Kill Switch on the user interface. Switch to the WebSSH app with the active SphereBot connection and type "sudo poweroff". When prompted for the password enter "temppwd" and hit enter. After approximately thirty seconds it will be safe to power off the SphereBot.

5) When things go wrong:

a) Kill the control system: If the SphereBot goes out of control and you need to stop it in a hurry, hit the Kill Switch. The kill switch will immediately stop the control system and disconnect the user interface from the SphereBot. This will stop the lift motors and end all stability control. This will not power down the SphereBot onboard computer so once the danger is passed you can log back into SphereBot and restart the control system.

b) Move out of the way of falling SphereBot: If you must kill the control system while the SphereBot is in flight then make sure all personnel are out of the way of the falling SphereBot.

6) Troubleshooting:

a) SphereBot does not power on: Check the battery to make sure it is charged. Check the battery connector. Make sure the power switch is in the right position.

b) SphereBot does not connect to router: Make sure the router is on and you can connect to it with the iPad. Check to make sure the SphereBot's onboard Wi-Fi adapter is fully seated.

c) Lift motors will not calibrate: SphereBot is defective, refer to service.

d) Battery dies quickly: Run balance program on charger. Charge the battery. Replace the battery.

e) User interface won't connect to SphereBot: Ensure the router is powered on and that the iPad has a good connection. Power cycle the SphereBot.

XIV. DOC User Manual

A. Introduction

The Deployable Observation Crawler, or DOC, is a semi- autonomous line crawler robot designed to be used for the inspection of high tension transmission lines. It is meant to be deployed onto this line from an unspecified aerial deployment system. It will provide additional information about the line and its current state. This will include, but not be limited to, images of sections of the power line.

B. Setting up the DOC

1) Charging the battery: To power DOC, a 9.6V rechargeable battery is used. Before proper functioning of DOC can be obtained, the user must make sure the battery pack is fully charged. Make sure to use the provided charger to ensure the optimal life span for the rechargeable batteries.

2) Positioning Camera: Initial testing and set-up of the camera itself is necessary by the user. Before sending the line crawler onto the line, the user will need to position the camera to the angle of their choice. This is highly important because once DOC is deployed onto the line, there will be no chance to change this angle. The desired angle can be obtained by running tests and operating the bot while on the ground and viewing images.

3) Test Deployment Spring: Pick up the robot and set it down. When the line crawler is released, the lever switch should be pressed down, at which point the latch should close, and the tires should start turning. If this does not happen, realign the spring and the switch and verify power has been supplied to the microcontroller.

C. The DOC Operations

DOC is designed to be a semi-autonomous robot with minimum human interaction necessary once deployed because of the designed use of the crawler. This implies that after initial set up of the line crawler, DOC operations will be handled by pre-written code. The only user required input dealing with the operation of the system will be in the retrieval of the images.

1) Frame Design: The frame is designed to allow the line crawler to be placed on the line by an aerial deployment device. It keeps the weight of the system balanced and provides a platform and mount for the other mechanical and electrical components of the system.

2) Latch: The latch acts as a catch system to prevent DOC from falling off the cable as it travels down the line and while it is being deployed onto the line by blocking the opening in the frame into which the cable was inserted during deployment. The latch is built from aluminum strips and a rubber wheel. The physical latch is mounted to a high torque retract servo to allow it to open and close based on the input from the microcontroller.

3) IR Distance Sensor: The infrared distance sensor provides an analog voltage level which is then used to determine the distance of an object relative to the front of the line crawler bot. With this distance the robot will know when it is approaching a pole or other obstacle that it cannot overcome and will stop. If this happens, DOC will need to be retrieved (see Section 8) and moved past the obstacle it has encounter.

4) Encoder System: The encoder system on DOC is used to determine the distance traversed on the line by the line crawler bot. It uses a 20 notched encoder wheel, an IR sensor, and an IR light bulb. It uses the number of IR detections to determine whether the wheel has made a full rotation. Every niche sends a pulse to the IR detector indicating that a niche was passed and increments a counter. Once this counter reaches multiples of 20 (i.e. 20, 40, etc.) we then increment a wheel rotation counter. Once the robot stops, the wheel rotation value is multiplied by the circumference of the wheel to give us a distance traveled in inches.

5) Camera System: The camera system is operated and controlled by a Raspberry Pi - Model B microcontroller. There are a couple of items (included in the package) that you will need in order to view the images taken. These two items are a micro USB cable and an HDMI cable.

a) Taking Photos: The code for the camera system is written to begin taking photos as soon as the system is turned on, or in our case as soon power is provided to the circuit. It will continue to take pictures until the system is turned off. Make sure the camera angled correctly and position you want it in before you deploy the system.

b) Retrieving Photos: Proceed to plug one end of the HDMI cable into the Pi and the other into an HDMI monitor of your choice. Power to the Raspberry Pi is required in order to retrieve the images from it. Please make sure that adequate power is supplied to the Pi by either connecting the battery pack or, if your battery pack is charging, plugging one side of the provided micro USB cable into a computer and the other side into the Pi. In order to retrieve the images taken by the camera system, the user will need to sign into the Pi. The username for logging in is ‘pi’ and the password is ‘password’. Once you have successfully logged into the Pi, type the command "startx" in the command prompt window in order to open the operating system of the Raspberry Pi. Once this starts up, you will find the images taken have been saved to a folder with the path name: ‘/home/pi/ImageCapture/’. Once these are located, you may plug a USB flash drive into the Pi and transfer the images from the Pi onto the flash drive.

6) Deployment: The deployment system consists of a wire hook attached to the frame on one side through a small spring which activates a mini lever switch as the weight of the system is taken off the

wire, which occurs and the line crawler is placed down, hopefully on the transmission cable. Once the switch is activated, the latch closes to keep the line crawler on the line and the line crawler will begin to travel down the line.

7) Retrieval: The switch for the deployment system acts in reverse when the line crawler is retrieved. As the line crawler is picked up by the wire hook, the mini lever switch is deactivated causing the latch to open and the motion of the line crawler to stops, if it hasn't already.

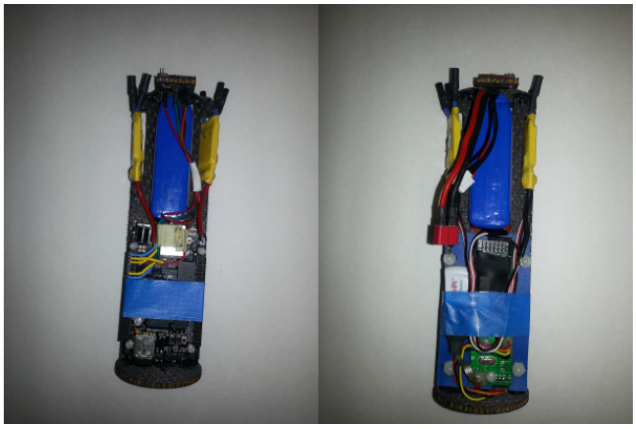


Figure 5: Electronics Sled, front and back

XV. Hardware

A. SphereBot Block Level

The mechanical build comprises the main body of the robot, as well as the electronic components that provide locomotion, control actuation and a delivery platform for sensors and cargo.

The SphereBot's electromechanical and electronic components are comparable to a human's muscles, internal organs and sensory organs respectively. The lift motors, propellers and the control surface servos make up the electromechanical components. Like human muscles they allow the SphereBot to move about in and manipulate its environment. The lift motors used in the SphereBot are contra-rotating DC brushless motors built in a single assembly. The contra-rotating setup results in zero net torque on the robot's frame while flying, so that no other mechanism is needed to keep the robot from spinning. There are additionally four ultra light, ultra fast micro servos that adjust the robot's control surfaces to control its flight. An 11.1V lithium polymer battery powers the robot. The robot's internal and sensory organs are its electronic components. The onboard computer, a 1GHz ARM based BeagleBone Black single board computer, is its brain. A five volt switching power supply, a USB hub and a six channel Pololu Micro Maestro servo controller serve as the robot's digestive and nervous systems. The functions of the eyes and ears are performed via an 802.11b wireless Ethernet adapter.

B. DOC Hardware, Block Level

The line crawler has a number of electrical and electromechanical hardware components which are combined with software to allow the line crawler to travel down and inspect transmission lines reliably. Four electromechanical servos are used in the line crawler, all of which can be safely run by a 5V supply and are controlled using pulse width modulation, PWM, sent from the microcontroller. Two of these servos are continuous rotation servos, for which the PWM signal sets the speed of the servo's rotation, and have been attached to tires to provide controllable propulsion down the cable. Another continuous rotation servo is part of the encoder system, and matches the speed of the propulsion servos. The attached encoder uses infrared light and a counter to calculate the number of times the servo has turned. The last servo is a retract servo, specifically the Hitec 75-BB, attached to the latch arm, and, unlike the other servos, the servo is designed to either open or close by rotating to either 0 degrees or 180 degrees determined by the PWM value sent to it. In addition to these components, a normally open mini lever arm switch is used in the deployment system. The distance to the pole is determined through an IR sensor, which measures the amount of infrared light reflected back to it from the IR led in the sensor and converts it to a voltage which can be read through an analog pin on the microcontroller. A 9.6V Nickle-Metal Hydroxide battery is used to power the system, which is regulated through the microcontroller and the motor shield which powers system shown by the block diagram in Figure 6 An external 5VDC voltage regulator is used to regulate the voltage from the battery to power the camera system.

1) DOC Deployment System: The hardware for the deployment system includes a wire for the aerial device to connect to, a normally-closed switch which is activated by the weight of the line crawler after it has successfully been deployed, and a latch

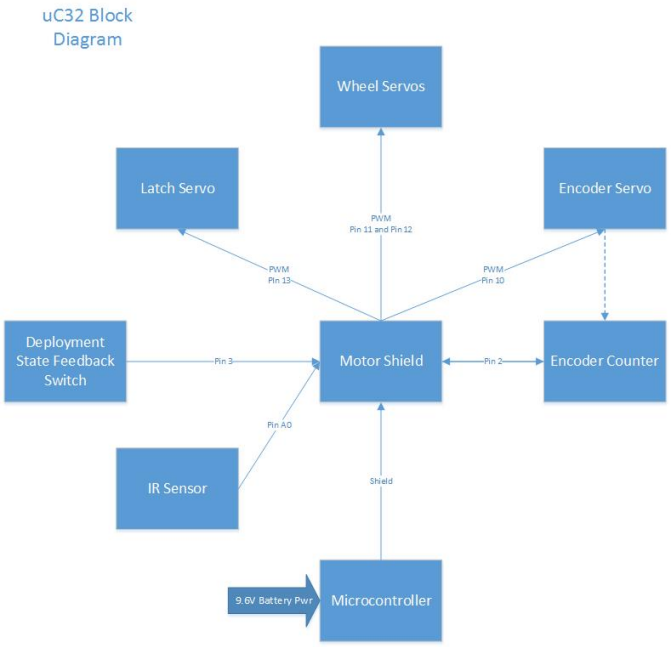


Figure 6: DOC uC32 Block Diagram

system which aids in the transition from the aerial device and the wire. In order to understand this system, a block diagram has been generated for the deployment system and is shown in Figure 7. The deployment device is controlled by the user

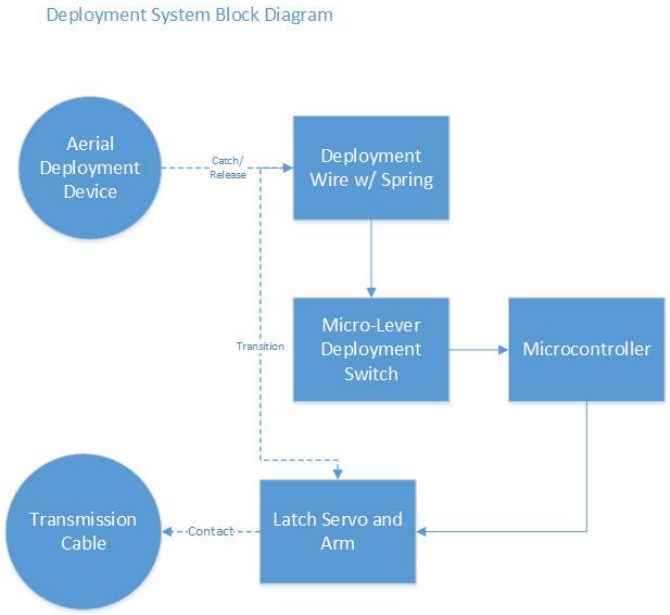


Figure 7: DOC Deployment System Block Diagram

and connects to DOC through a simple wire hook built into the frame. This hook has a small spring on it which causes the wire to activate a lever switch, and sends a high to the microcontroller. After that contact has been made, the servo the latch is attached to will activate to close the latch. Once closed, the latch will maintain near-contact with the transmission cable. The latch is designed to make contact if DOC starts slipping, and provides a way to prevent the line crawler from falling off the line in the deployment process once it has been aligned with the transmission line.

2) DOC Control System: For the control system, the hardware includes a motor shield, two continuous rotation servos with attached wheels for propulsion down the cable, an infrared, or IR, sensor and the lever switch and latch system from the deployment system. The motor shield acts as a shield, and plugs directly into the microcontroller. Once the shield is plugged in, the pins are connected between the two boards. The mini-lever switch is a small switch mounted just below the deployment wire, which is plugged into pin 2 and will send a high voltage once the line crawler is deployed, allowing the latch servo, plugged into PWM pin 13, to close so that the line crawler will not fall off the line while it is traveling down the cable. The continuous rotation servos are directly mounted to the wheels and are physically wired the the PWM, pulse width modulation, pins 11 and 12 on the motor shield on the microcontroller shown in Figure 8. The

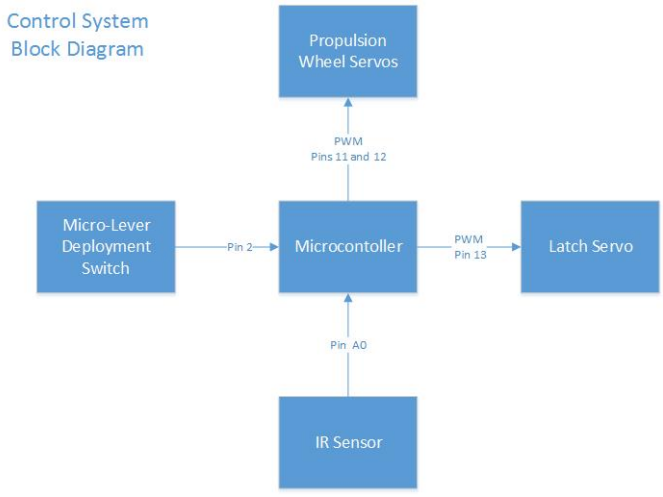


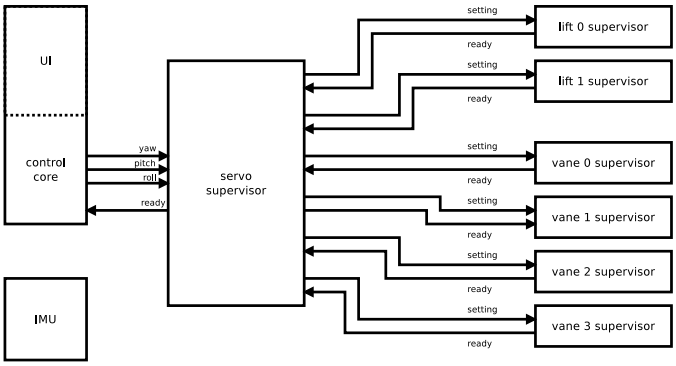
Figure 8: DOC Control System Block Diagram

IR sensor plugged into analog pin 0 acts as sensor to detect a pole as DOC nears it, allowing DOC to stop before it runs into it.

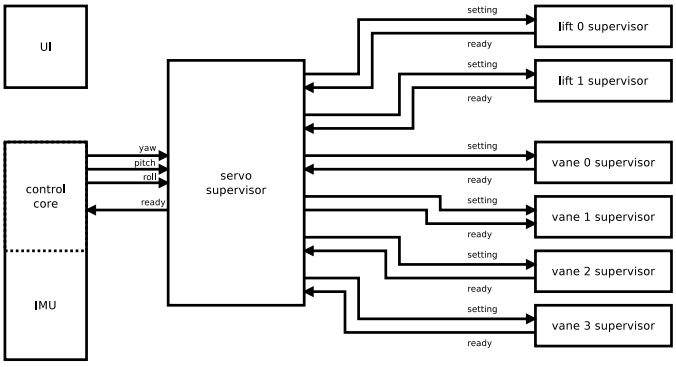
XVI. Software

A. SphereBot Block Level

Because the Electronic Speed Controllers (ESCs) which drive the lift rotors require boot-time calibration, and because boot-time calibration of the vane servos was anticipated, the initial architecture of the servo module (Figure 9) involved separate submodules for each servo and a supervisor submodule to coordinate their operation. Communication between these modules was done via a series of variables in shared memory, using mutexes and condition variables to signal when data was available. This was technically a viable option for the servo module itself, but it caused difficulties when the time came to build the stabilization and user modules. Because any given module could only listen to a single variable, the core could not receive input from both the stabilization and user modules. For testing and development, two versions (Figures 9a and 9b) were created of this monolithic, first-generation control system.



(a) Monolithic core/user unit



(b) Monolithic core/stabilization unit

Figure 9: First generation Control System, with monolithic implementation of user *or* stabilization functionality

The second generation of the control system eliminated this difficulty by replacing the intermodular communications with message queues. The servo module retained its shared memory architecture internally, but receives settings from the core module via its message queue. Rather than waiting on either of the two input modules, the core module waits only on its message queue for input. After handshaking with the core module, the user and stabilization modules wait on a network socket and a serial port, respectively.

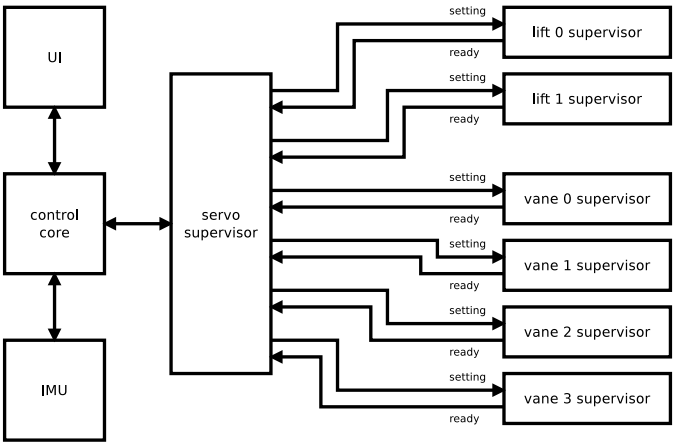


Figure 10: The second generation control system achieves intermodular communication with message queues.

The third generation of the control system (Figure 11) overhauls the servo module to condense the seven submodules into a single module. As the initialization of the ESCs is now performed by the Maestro servo controller, the extra complexity was no longer needed. Additionally, an artifact of the first generation of the control system had a portion of the core module's PID controllers implemented in the servo module; this functionality is being restored to the core module.

B. SphereBot Subroutine Level

1) Control System core routine: The Control System's core module starts by initializing its own local variables and its incoming message queue. Next, the core launches the user module, the stabilization module, the servo module, and the heartbeat submodule, passing the core's message queue to each of them. The core then waits to receive messages indicating events, and responds to them. A `HELLO` message indicates that a thread has started and wishes to pass its own message queue to the control core; this information is stored in a local

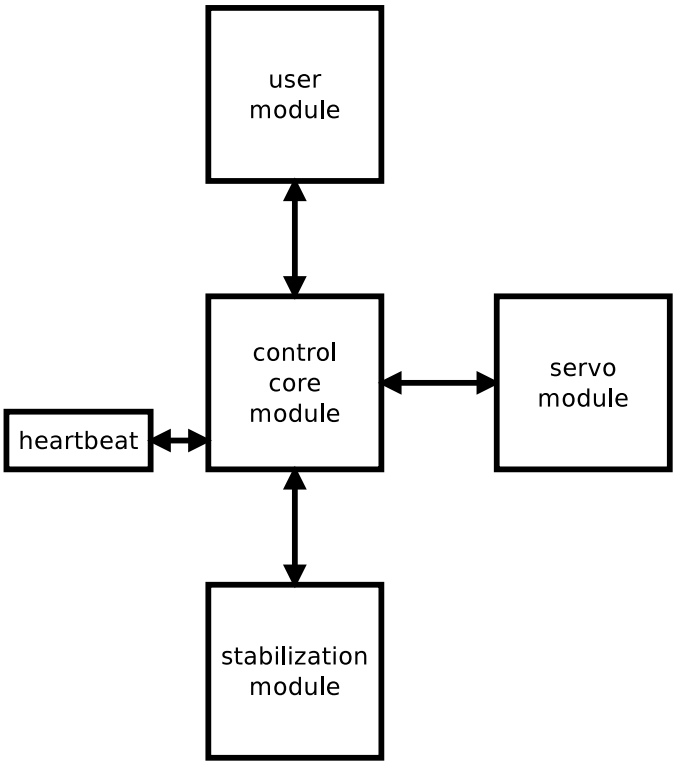


Figure 11: The third generation control system simplifies the core and servo modules

array. A `UI_DATA` or `IMU_DATA` message indicates that the user module wishes to pass settings for the PID controllers, or that the stabilization module wishes to pass plant variables. A `TICK` message indicates that the heartbeat module has measured a fixed interval, configured at compile time. When this message is received, the control core runs the PID controller calculations and issues the result to the servo module. Lastly, when the control core receives a `QUIT` message, it passes the message to all running modules, joins each of the threads as the modules exit, and then exits itself.

2) Control System PID Controllers: The pitch and roll channels have the most useful plant variable data from the Stabilization System, and they are most vital for the flight stability of the SphereBot. Therefore, these two channels have robust PID controllers. Fundamentally, they implement the set of equations,

$$e_{kT} = \text{SP}_{kT} - \text{PV}_{kT} \tag{1}$$

$$\mu_{kT} = P_{kT} + I_{kT} + D_{kT} \tag{2}$$

$$P_{kT} = k_P e_{kT} \tag{3}$$

$$I_{kT} = I_{kT-T} + \frac{k_P}{T_I} e_{kT} \tag{4}$$

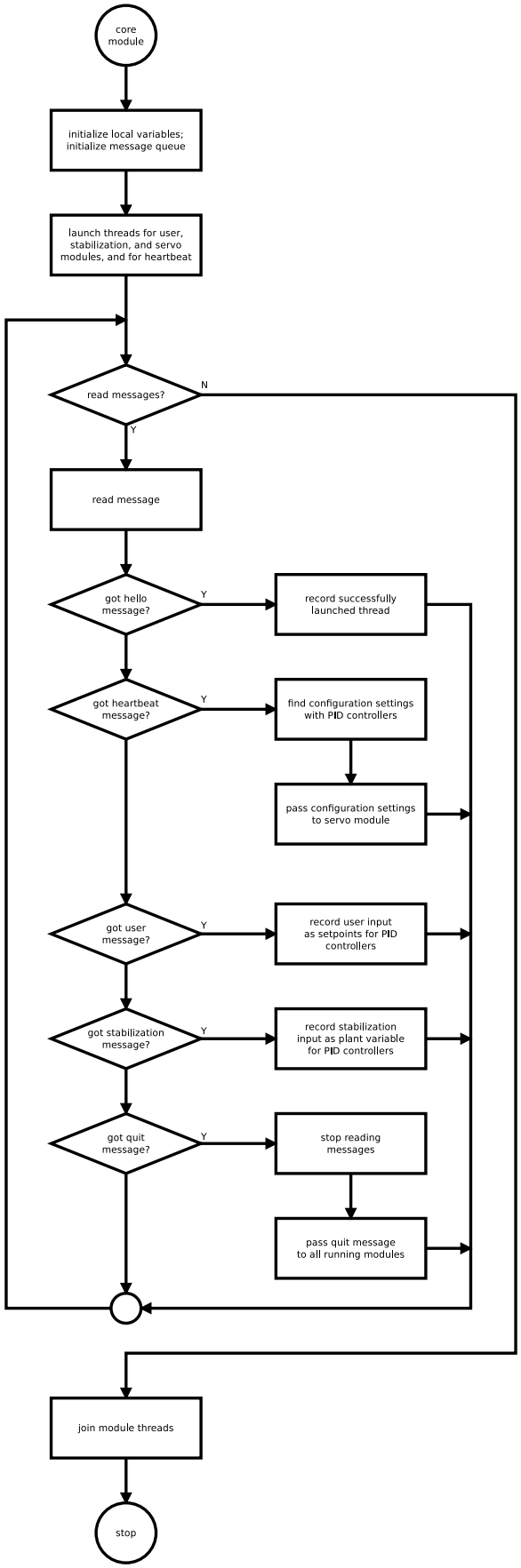


Figure 12: Control system core module simplified flowchart

$$D_{kT} = k_P T_D \left(\mathrm{PV}_{kT} - \mathrm{PV}_{kT-T}\right) \quad (5)$$

where:

e_{kT} is the instantaneous error between the setpoint and the measured process (or plant) variable,
μ_{kT} is the control variable produced by the PID controller,
k_P is the proportional constant, a tuning variable,
T_I is the reset time, a tuning variable, and
T_D is the derivative time, a tuning variable.

The derivative component of the controller (Eqn. 5) uses the process variable, rather than the error, to eliminate large changes in the output due to changes in the setpoint. In actual implementation, the PID controllers for pitch and roll start by updating a moving average filter in order to reduce noise from the IMU. Window sizes from 1 to 50 samples have been tried; 15 samples appears to provide a good balance with sufficiently low noise and sufficiently high responsiveness. The individual proportional, integral, and derivative components are calculated. The integral portion is clipped to within the output range allowed, and the integrator is quenched entirely when the lift rotors are not running. These measures prevent a condition referred to as windup, in which uncorrectable errors accumulate and lead to overcorrection. Finally, the three components are summed, clipped to within the output range allowed, and returned.

Because the process variable for yaw remains erratic and the process variable for altitude has insufficient resolution, the yaw and lift channels are provided with skeleton PID controllers which perform proportional calculation only.

3) Control System input modules: When the user module is launched, it initializes its local variables, passes a `HELLO` message to the control core, and opens a listening TCP socket on all available interfaces. It accepts the first available connection and begins reading data from the socket. If a message from the User Interface System indicates that the user has pressed the Kill button, a `QUIT` message is passed to the control core; otherwise, the setpoint data is packed into a `UI_DATA` message and passed to the core for processing. The operation of the stabilization module is similar. After initializing, it opens a serial connection to the Stabilization System and begins reading IMU data. If the checksum passes after a record is read, the process variable data is packed into an `IMU_DATA` message and passed to the control core. In each module, after each input record is processed, the module checks its own message queue, solely to determine if it has received a `QUIT` message. If not, the module waits for its next input record.

4) Control System output module: When the servo module is launched, it likewise initializes its local variables and passes a `HELLO` message to the control core. It then initializes the Pololu Micro Maestro servo controller and waits for messages from the core. Upon receiving a `SET_VANE` message, the servo module records the desired control vane settings, to control pitch and roll. For a `SET_ROTOR` message, the module records the desired lift rotor settings, to control lift and yaw. When the servo module receives a `SRV_COMMIT` message, the module computes the individual servo settings required to achieve the attitude configuration settings, and commits those servo settings to the Maestro. Finally, if the servo module receives a `QUIT` message, it exits. Otherwise, it waits for the next input message.

C. DOC Software, Block Level

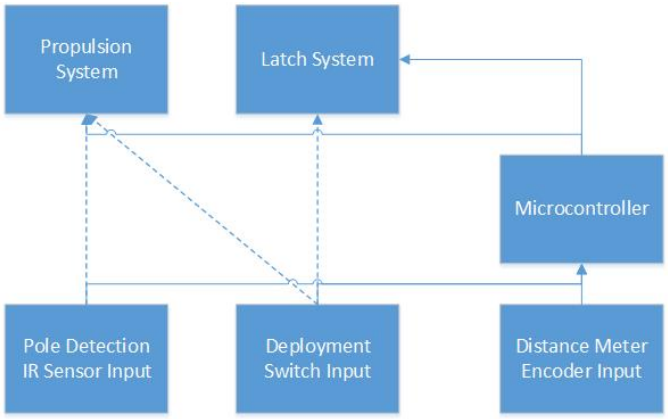


Figure 13: DOC software block diagram

DOC's software is split into two sections: the image system and the microcontroller system. The microcontroller system consists of five different block sections interconnected through the programming of the chipKIT uC32, the microcontroller used in DOC. One block is the sensor used to detect a pole, and the feedback from this sensor directs the propulsion block to stop as DOC nears the pole signifying the end of that section of the transmission line. The deployment switch block activates when the line crawler is set on the line and is used as an indicator that the latch system should close and

the propulsion system should start moving the line crawler forward. Finally, the distance input block acquires information from the encoder's counter to calculate information about the location of the line crawler on the line. This information is intended to be used to correlate the images taken of the line to the specific location on the line, but does not impact the operation of the robot on the transmission cable.

The image system consists of only one block that is controlled by the Raspberry Pi Model B microcontroller. One block is used to control, and also receive data from, the HD camera being used to take images of the line. This block simply records video of the line and and receives it from the camera. Once the video is received and saved, the block then splits the video into frames and saves these frames as images. Each image is time stamped and then saved again to a specified location. This time stamp is necessary in order to correlate the encoder's distance traveled calculations with each image using the time stamp of each. The images themselves are going to be used for later viewing in order to determine the condition of the power line.

1) DOC Vision System: The most important thing obtained from a human inspector up on the high tension power line is their sight, giving them the ability to see the line up close and determine potential faults on the line. With this in mind, the first type of data acquisition tool implemented was a camera system. The camera will be mounted on the bot and allow the system to take images of the power line to be viewed later by a trained professional while they are safe on the ground.

To implement these sensors developers identified a few different options for microcontrollers to choose from. The three that ideal candidates were the Microchip chipKIT uC32, which is used to control DOC, the BeagleBone Black, and the Raspberry Pi. The uC32 is based on the popular Arduino open source hardware prototyping platform [25] and was the most familiar to the assigned developer, but did not provide the processing power of either of the other two options nor did it have the amount of library options or functionalities for video/image processing as the BeagleBone Black or the Raspberry Pi. The developers decided to use the Raspberry Pi because it provided the option of programming the camera system in Python using OpenCV, which has a good number of libraries for image processing already written.

The Pi has graphic capabilities that are roughly equivalent to the original Xbox's level of performance. The Graphics Processing Unit (GPU) is capable of 1G $^{\text{pixels}}/_{\text{s}}$, 1.5G $^{\text{texel}}/_{\text{s}}$ (texture element) or 24G FLOPs (Floating-point Operations per Second) of general purpose computing with a 512MB RAM CPU and many other graphics/image processing features. It also gives the ability to save the images to an SD card in order to be viewed later by trained personnel.

The options for the camera came down to first deciding whether to use a standard or a high definition camera. Developers decided to go with a high definition camera to maximize image clarity: it was necessary to obtain images with enough resolution for the viewing technician to be able to effectively determine whether the power line contained a fault or not.

Developers chose the Logitech C615 high definition web camera because it was relatively inexpensive and provided full HD 1080p video capture (3) . It also had the feature of auto-focus (3), which was somewhat vital because the operator will not be able to focus the camera by hand. The final feature that was looked at was the universal clip that allowed for a rotating camera and adjustable camera angles and made the camera more easily mountable to DOC.

D. DOC Subroutine Level

The software for DOC makes it possible for the line crawler to meet its feature set and is separated into two sections: the microcontroller software and the Raspberry Pi software. The microcontroller is responsible for taking the input from the mounted sensors, controlling the DOC's motion, and acquiring the distance DOC has traveled. On the other side, the Raspberry Pi stores the image data from the line. Neither process ends until the line crawler is powered off.

The line crawler initialized the system to clear the variables, set the servos that the tires are mounted to stop, and set the latch servo to open the latch as shown in Figure 15.

In order to keep the line crawler from rocking on the line, the speed of the line crawler could not change rapidly. Ideally, an IMU or other sensor would be incorporated to detect rocking so that it could be compensated for directly, but too much time would be required to fully incorporate the sensor. The alternative solution was to ramp the

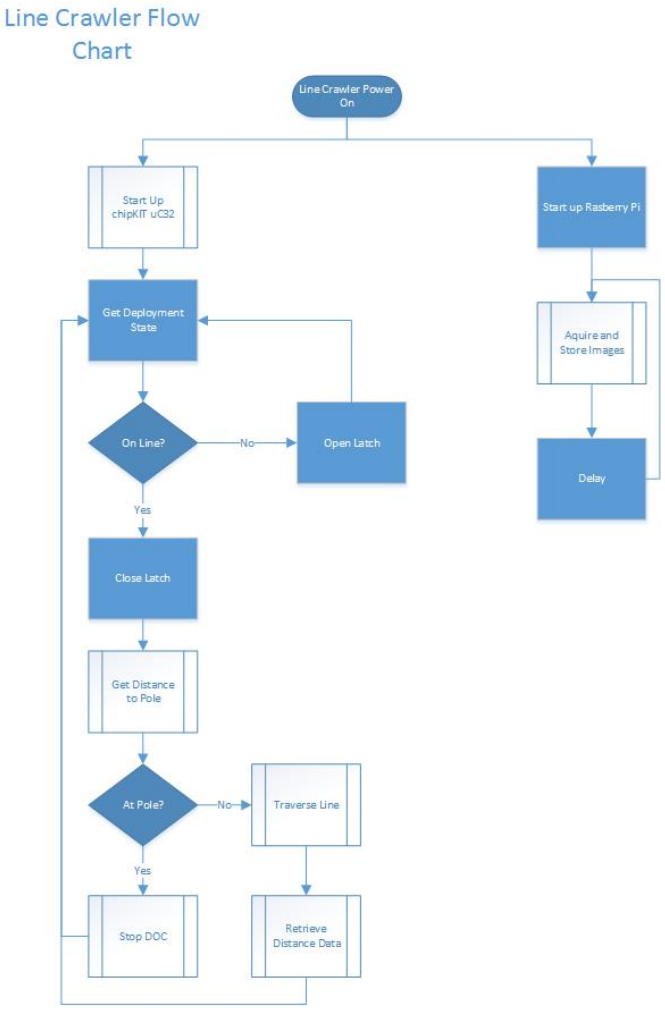


Figure 14: DOC Control System Flowchart

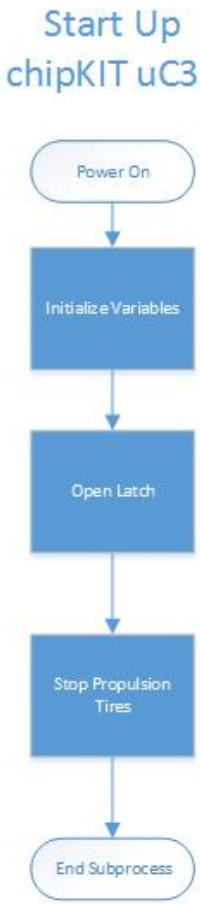


Figure 15: DOC Startup Flowchart

speed so that the system did not begin to rock as a result of known forces, such as changing directions. The ramp deration is a result of testing and has dependence on the design of the overall system, but the values are stable once they are determined. The flow chart shown in figures 16 and 17 show the processes of ramping up the speed as the line crawler starts moving and decreasing the speed of the tires as the line crawler stops respectively. The variables of the ramp are related between the two subsystems, as it is used to determine the current speed of the crawler.

XVII. Mechanical

A. SphereBot Mechanical Build

As mentioned in Section XV-A, the mechanical build includes the main body of the robot. The relationship of the frame to rest of the system is comparable to the relationship of the human skeleton to the rest of the human being. The frame provides a mounting point for the lift motors, the control surface actuating servos, and a housing for control, communication and power supply electronics. The spherical frame, like a human's rib cage, provides protection for the robot's vital internal components and also protects the power lines from damage caused by the robot's propellers.

The SphereBot's frame is comparable to a human's skeleton. Its control vanes and lift rotors, then, are its legs and feet, although wings would be a more apt analogy. The contra-rotating setup of the lift rotors results in zero net torque on the robot's frame while flying, so that no other mechanism is needed to keep the robot from spinning.

1) Options and Solutions: Three different materials were considered for the frame, each with its own benefits and drawbacks. The first material considered was carbon fiber. Carbon fiber is very light, very strong and its dark color and thatched pattern are very aesthetically pleasing. Carbon fiber, therefore, would seem like the perfect choice for a

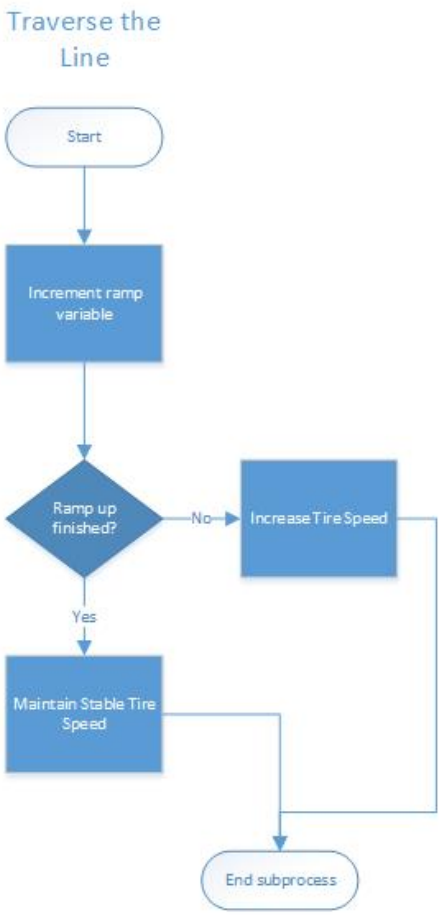


Figure 16: DOC Line Traversal Flowchart

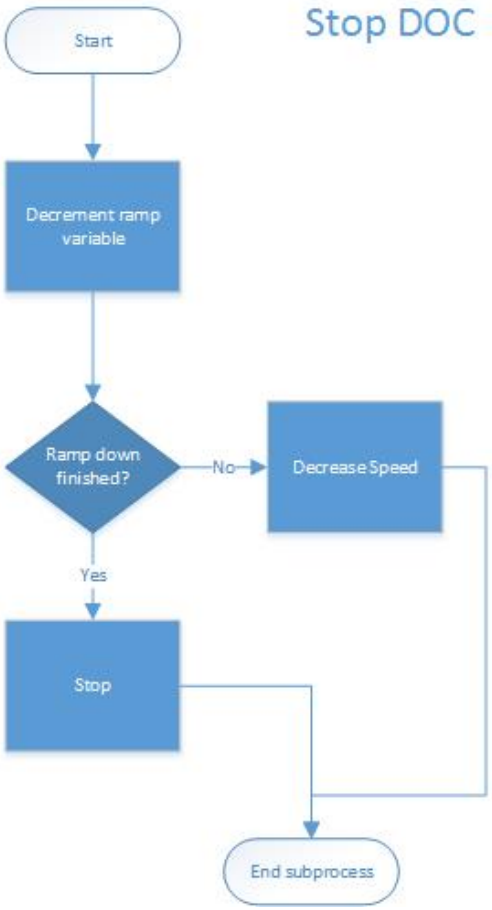


Figure 17: DOC Stop Flowchart

frame material, however, the cost of carbon fiber fabrication was found to be very prohibitive. The high cost, coupled with the fact that none of the team members had carbon fiber on experience, led to the decision to forego carbon fiber for the initial frame build. Similar to carbon fiber is the fiberglass material used to make circuit boards and this material was considered in planning the SphereBot's frame construction. Circuit board material is not as light as carbon fiber and is far more flexible, so was not as desirable. Its light yellow color and lack of any surface pattern also make it far less visually appealing. Like carbon fiber circuit board material is also hard for laymen to work, requiring special cutting machines and fabrication experience, therefore, proved impractical for the initial build. ABS plastic is considerably heavier than either carbon fiber or circuit board material and not nearly as impressive, however, it was chosen for the initial frame build primarily for the ease with which it can be cut and assembled. ABS's drastically less expensive cost also provided the opportunity to make mistakes, while nailing down the general design of the frame, that would have been too costly, in both time and money, if either of the other two materials had been used. The resultant frame is pictured in Figure 18.

2) Mechanical Improvements: Each phase of testing revealed shortcomings in the preliminary mechanical build. By far the most conclusive tests were the weight and thrust tests. Because the weight of the SphereBot was determined to be 12.05N and the maximum thrust was found to be only 10.09N, it was determined that a new lighter frame would be needed to improve the thrust-to-weight ratio. Without a lighter frame or stronger lift motors the SphereBot would never fly. Center of gravity testing revealed that weight would have to be shifted from the bottom of the robot towards the middle in order to increase the influence of the control surface directed thrust and improve response to control surface adjustment. Thrust pattern testing showed that while the disturbances caused by the control surface link arms were minor, airflow could be improved and the design could be simplified

Figure 18: Second-generation ABS frame

by removing the link arms and having the control surfaces driven directly by the servos. Accessing the electronics for maintenance and modification, as well as battery charging, in the testing phase made painfully evident that a new method must be devised to access the electronic components in order for the SphereBot to be deemed a success.

Several improvements were made to the new frame, each addressing a different shortcoming of the original frame. The first and most important improvement was the adoption of carbon fiber for the build material. Despite being built out of ABS plastic, making it far too heavy, the initial frame build was not a total waste of time because it facilitated the development of a general design that was used as a model for the new frame. The sheets from which the frame parts were cut were fabricated using epoxy pre-impregnated carbon fiber sheets. The sheets were laid on an eighth inch thick cardboard honeycomb material called Nomex core and then heated in a sealed vacuum oven called an autoclave. The vacuum ensures that a uniform pressure is applied across the surface of the part while it is curing. The cure cycle for the epoxy pre-impregnated sheets and Nomex core sandwich is accomplished in three stages. The first stage is the ramp up stage. In the ramp up stage the temperature in the autoclave goes from room temp to 350 F over the course of one hour. The second stage is called the hold stage. In this stage the temperature is held at 350 for one hour. The third stage called ramp down. During ramp down the heater is turned off and the doors to the autoclave are left closed. It is then allowed to cool down to about 100 F. Once the sheets are removed from the autoclave they are allowed to sit for five hours to finish curing. The sheets are shown from the top, side and close-up in Figures 19a, 19b and 19c respectively.

The sheets were then sent, along with CAD drawings of the flat parts, to a facility where the flat parts were precision cut with a high pressure water-jet cutter. A cylindrical center tube was constructed with a cylindrical form and the same pre-impregnated carbon fiber sheets. The center tube and flat parts were then glued in place using high-strength epoxy. The resulting SphereBot with electronics installed weighs approximately one half what the old SphereBot did. Additionally, an electronics sled was created that holds all electronic components and slides right into the center tube from the bottom. The sled greatly improves access to the internal electronics. It also carries the lithium polymer battery and positions it much closer to the center of the SphereBot. This improves the center of gravity and enhances the influence of control surface directed thrust. The improved frame and both sides of the electronics sled are pictured in Figures 20 and 5.

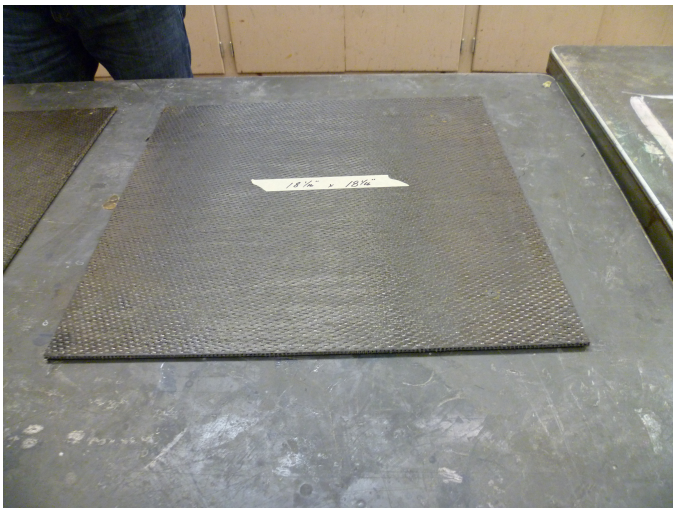
B. DOC Mechanical Build

DOC's mechanical components consist of the frame and the latch. The basic frame consists of an upside-down Parallax Boe-Bot frame mounted to a bent aluminum bar which, in turn, is mounted to a bend piece of aluminum sheet, to which the tires are mounted. This frame can be seen in Figure 21. The latch consists of a three pieces of aluminum cut from a VEX bar. Two are attached perpendicularly to form the "arm" of the latch with a servo arm attached to one end of the arm and the 1 inch rubber tire attached to the other side using two collars and a $^1/_8$ inch shaft. The last piece of metal is used to stabilize the previously unsupported side of the shaft the tire spins on.

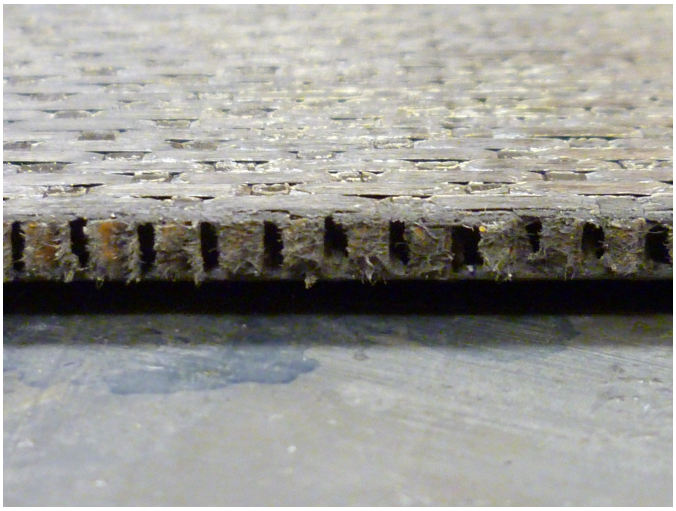
XVIII. Hardware Testing

A. SphereBot Mechanical Build

Five basic tests were performed on the SphereBot's mechanical build. Those tests were: thrust



(a) Carbon fiber sheets before cutting



(b) Carbon fiber sheet side view

(c) Carbon fiber sheet up close

Figure 19: Carbon Fiber used in third-generation frame

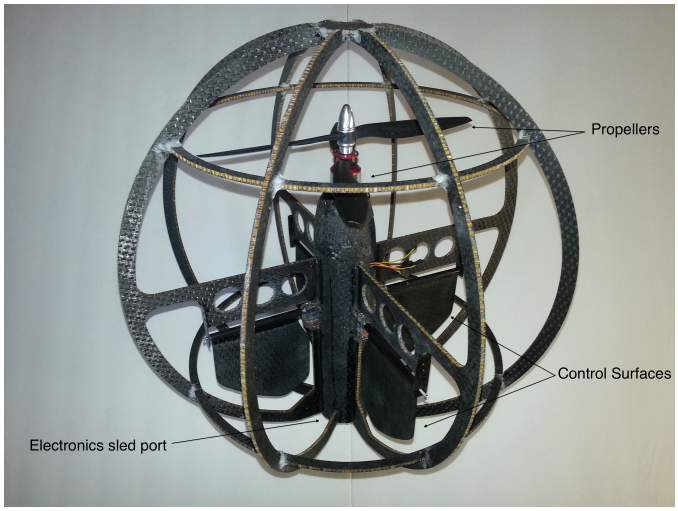


Figure 20: New SphereBot frame

pattern testing, weight testing, thrust testing, center of gravity testing and ease of access to electronics. In thrust pattern testing the directed air flow from the propellers was checked to see how well the air column flowed down through the frame of the robot. It was found during this testing phase that the link arms that actuate the control surfaces cause some disturbance of the air column. Weight testing was done to establish the weight of the robot, so that it could be compared to the available thrust. The testing was performed using a digital hanging scale that gave the equivalent mass measured in kilograms. The mass was then converted into weight in units of Newtons by multiplying by 9.8, the acceleration due to gravity on Earth. The weight was determined to be 12.05N. The same digital hanging scale was used to determine the maximum thrust by hanging the SphereBot upside down on the scale and zeroing the scale. The lift motors were then slowly ramped up to max thrust and the mass reading corresponding to the thrust exerted by the lift motors was recorded and converted to Newtons. The maximum thrust value was found to be 10.09N. The final test, ease of access to internal components, was completed as a consequence of work done on the internal electronic components of the SphereBot. The testing found that accessing the internal components of the SphereBot was exceedingly difficult, requiring the disassembly of the housing bottle to perform simple tasks such as charging the battery. The center of gravity test checked the effect that the low center of gravity designed into the frame had on the performance of the SphereBot. The ability of the SphereBot to right

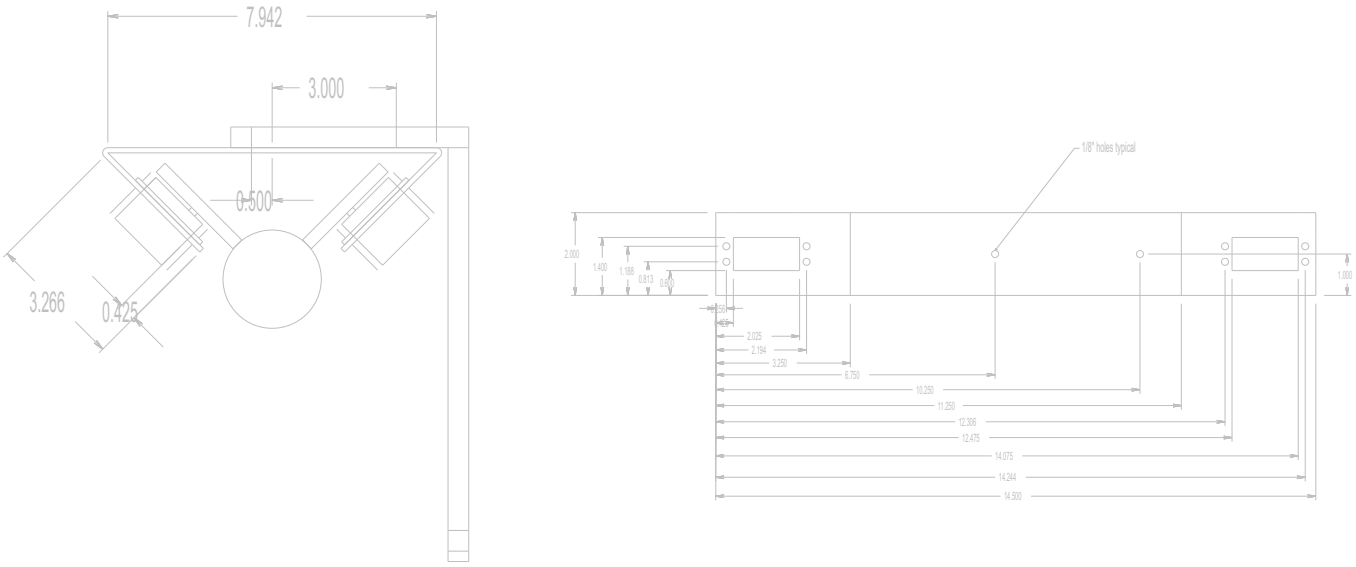


Figure 21: DOC Chassis

itself while sitting on the ground was part of this test and the SphereBot was able to right itself. The combined effect of the low center of gravity and the influence of the thrust directed by the control surfaces was also tested. It was found during this test that the very low center of gravity significantly reduced the influence of the control surface directed thrust. This reduced influence caused slow reaction to attitude adjustment via directed thrust.

B. DOC Frame

Once the frame was manufactured, the servos with their mounted wheels were attached to the frame and it was placed on the piece of 3 inch PVC pipe used as a simulated transmission cable for most of DOC's testing to verify the frame was easy to deploy onto the line and had good contact with the "cable." The frame arm bent slightly to balance the expected weight more evenly, and the frame was ready to go to the next step. A picture of the frame on the PVC pipe as it was for this test is shown in Figure 22.

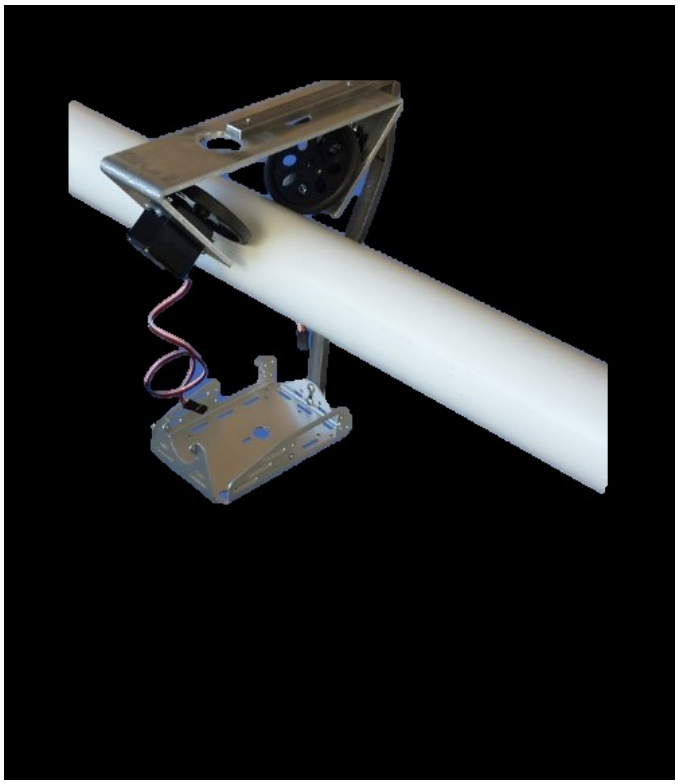


Figure 22: Initial balance testing of the DOC frame on the simulated power line

C. DOC Attachment System

As the parts were acquired, they were tested for functionality as their use in the project required, including the switch, the servos, and the sensors. Once it was determined that the individual parts for the latch system were working, the basic propulsion codes to ramp the speed of the robot up and stop after a set distance were combined to test low-level systems, which were then tested to verify the parts and the code worked and that the logic was sound for each of the subsystems . For example, once the flex sensor and the micro servo known to function, the micro servo was combined with the prototype

latch was tested with the flex sensor to verify the response. This allowed developers to verify that the latch closed when the Line Crawler was on the line and opened as it was picked up. The code was also modified for each subsystem, going back to the previous subsystem, the latch and flex system code was modified to turn the latch far enough to successfully open far enough to easily remove the line crawler from the line, and close far enough to prevent the line crawler from falling. Some problems could not be fixed or adjusted during testing, for example the flex sensor sliding off the line, and those were noted and taken into account for the next version. The other test that was repeated through each step was the weight distribution on the robot with the different systems attached. Once the subsystem had been checked a couple of times, and the behavior of the system was understood and accounted for as well as possible, the system would move onto the next step of integration and retested to insure the individual subsystems still behaved as expected and then that they worked together to meet the requirements of the combined system. Finally, the entire system was integrated and tested to run on the PVC line. After a few modifications to the code of the first prototype latch system, it operated as it was meant to with the exception of the sliding flex sensor, and the need to calibrate the flex sensor as the system was turned off and on, and developers were able to get more robust parts and trade them out to make the revised system, with the exception of the sensor. The testing process followed the same general steps on the revised system. At this point, the HS-75BB retract servo has been tested to ensure it works and verify the PWM signal that activates it. The micro-lever switch was also verified to insure it acts like a switch should and can be activated with the slight pressure the deployment spring provides. Both passed individually, with the servo smoothly going to the correct position and the micro-switch consistently providing a string of "on" or "off" through the serial monitor when it is activated or deactivated with clean switching . However, there were some integration errors in the sub system which includes both the servo and the micro lever switch which have been resolved. Finally, the switch was mounted and tested in response to the deployment system spring activation. The deployment system now passes these tests.

D. DOC Propulsion System

Initial testing of the hardware included testing of the servos to make sure that they worked properly. This was done before the servos and/or wheels were attached to the frame of DOC. This test was also to make sure the developers could make a connection and get a signal to each servo and that they worked correctly. The results were positive for this initial testing and both servos rotated fully 360 degrees and had a smooth flow.

The next test performed was to make sure that the servos continued to work correctly once they and the circuit were effectively mounted to DOC. This test demonstrated that DOC traversed the line, and it tested the stability of DOC while on the line. During this test, the bot traversed the lines multiple times to make sure that the wheels and servos could support the weight of DOC. This included testing of added features like push button start/stop and making sure that it controlled DOC and that DOC remained stable on the line when started and stopped.

Every time a new component/feature was integrated into DOC, the developers retested to make sure this was still true. So far, even with the battery pack (which is the heaviest component on DOC) positive outcomes have been seen from the testing. The servos and wheels have been able to support the weight of DOC and still move up and down the line and stop, when need be, effectively.

E. DOC Camera/Data Acquisition System

The main component of this system was the Logitech High Definition camera itself. The first thing that need to be tested was that the quality of the video/images of this camera was adequate for our purposes. To check this, an operational test was performed by simply connecting the camera and looked over numerous images and photos taken by the camera of various objects and people. This test demonstrated that the camera has excellent resolution and would work perfectly for the needs of the robot.

After the camera was attached securely to DOC, the next step was to make sure that the camera system was effective at obtaining images of the line and provides a clear representation of the entire line by looking all the photos. A number of different runs of the bot were went through while obtaining images (a few images obtained are shown in Figure 25). The different tests consisted of different angles

and positions of the camera to see which gives the best and clearest images.

The next big component of the Camera System that required extensive testing was the voltage regulation circuit. The battery being used to power DOC is a 9.6V Lithium Ion rechargeable battery. The camera system itself takes no more than 5.5V before the components begin to mess up. After the circuit was built the first test ran was to check to see what a fully charged battery pack was outputting. This was determined to be around 10.7 volts and was fairly consistent.

After it was determined what this is outputting, the next step was to directly test the voltage regulation circuit and see if that circuit is outputting acceptable levels for voltages. The original testing gave outputs of no higher than about 5.4V so initially there were very good results. One major issue seen with this testing was that the output voltage of this circuit would start at a certain acceptable level and continuously drop very quickly for some reason. The output voltage ended up dropping to as low as 1.3V and staying constant at this voltage level.

F. DOC Control System

The control system consists of a number of a number of different sensors and components and required the testing of the integration of these things with DOC. One of the first sensors that needed to be tested was the pushbutton that was used to make DOC start and stop. This was needed to make sure that the button was getting a signal to DOC and that DOC responded correctly at the time. This pushbutton was retested every time a new component was added to ensure that it still effectively controlled DOC's start/stop. This testing found that the push button worked well with the system but it was decided to take another route to control the start/stop DOC.

The next sensor tested was the infrared distance sensor that would be used to detect obstacles in front of DOC and stop its movement. To test this sensor, the developer started by ensuring that an ample signal was received from the sensor when faced with an obstacle. Then, calibration was verified by measuring specified distances on a piece of paper and holding objects at that distance. While the object was in front of the sensor, the developer checked to make sure first that a signal was received and secondly that the distance being measured was accurate. These results were very positive and confirmed both that the sensor was working correctly and that the code was doing the correct voltage to distance conversion.

XIX. Software Testing

A. SphereBot Control System

1) Servo Interfacing: One of the appealing aspects of the BeagleBone was the enhanced High Resolution Pulse Width Modulation (eHRPWM) modules. There are three modules, with two channels each, and the SphereBot required six channels of servo control. Per specification, they can be configured to the proper period (20 ms) and pulse width (1500$\pm$500 μs) to be compatible with RC servo signals. The first stage of development was, then, to interface the PWMs with the servos and ESCs. Extensive testing and behavior analysis was performed on the BeagleBone during this phase, using redirection in the `bash` shell environment to pass messages to the MCU's hardware interfaces exposed under the `/sys` filesystem. Through this testing, it was determined that there is a bug in the current version of the PWM drivers. Each module can either drive both channels with an unacceptable period or drive one channel at the proper period. With this failure, the BeagleBone was limited to driving three servos.

Development proceeded past the weakness in the BeagleBone by offloading the servo interfacing tasks to the Pololu Maestro, as mentioned previously. While developing the servo module, operation of the vane servos was coordinated by variously using unmounted servos, the servos mounted to the chassis of the SphereBot, and a test rig (Figure 23) for independent, parallel development. These methods showed that the control vanes operated in the expected direction, based on the input to the servo module.

2) Data Paths: During the data-driven periods of development, there was a need to trace the flow of the program, compare inter-thread timing, verify expected data manipulation, and identify faults. Conventionally, a debugger such as `gdb` would have been the ideal tool for such tasks. However, portions of the program are sensitive to timing, and so extensive use was made of debugging text (e.g. Figure 24) to meet this need. This text was generated primarily by the use of the `puts()` and

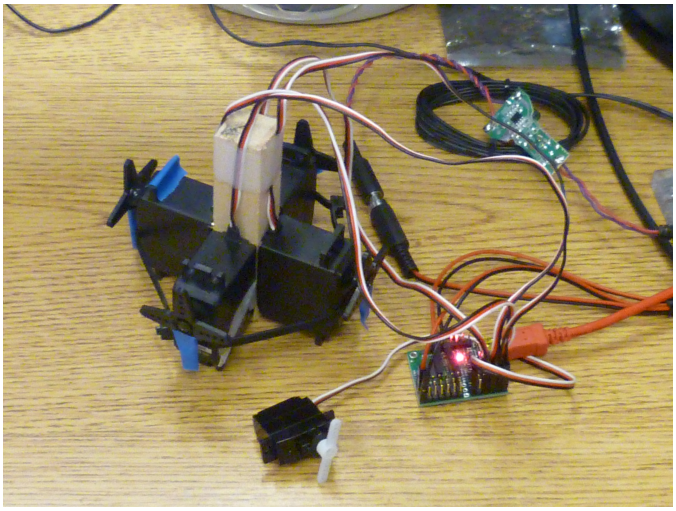


Figure 23: A servo test rig was used to simulate the vanes of the SphereBot during testing.

```
<pts://wortmanr@csgc-pc18:22/~wortmanr/msg_queue> make dep && make
make: Nothing to be done for `dep'.
./SphereBot
core got hello
core got hello
core received heartbeat
core received heartbeat
core received heartbeat
core received heartbeat
core received heartbeat
core received heartbeat
servo got set vane
core received heartbeat
core received heartbeat
core received heartbeat
core received heartbeat
core received heartbeat
core received heartbeat
core received heartbeat
core received heartbeat
core received heartbeat
core received heartbeat
core received heartbeat
core received heartbeat
^Cmake: *** [demo] Interrupt

<pts://wortmanr@csgc-pc18:22/~wortmanr/msg_queue>
```

Figure 24: Debugging text displayed to the console is used to monitor the running program.

`printf()` standard C library calls, and were monitored via the same `ssh` connection that was used to launch the control program. Through the course of development, this method of program monitoring successfully identified the causes of erratic behavior for adjustment, and demonstrated correct behavior when a portion of code was complete.

B. DOC Propulsion System

The first tests ran were to make sure the coding compiled correctly and effectively made the servos rotate for a set number of rotations before stopping. Ultimately, this testing was directly related to the testing of the hardware aspect of the propulsion system, which included the servos and wheels. Once the servos were attached to the frame, a testing code was written to have DOC move up the line, stop and then reverse directions. Every time a new component/feature was integrated onto DOC, the developers reran this test to ensure it compiled and that this feature and its variables or functions were not affected.

C. DOC Camera/Data Acquisition System

The software written for this aspect was designed to obtain video feed from the webcam and separate this video into its individual frame. Once they are separated, each frame will be time stamped and saved to a file with different names for later viewing. The testing involved with this included running through the code multiple times and checking to make sure that the video was obtained. The quality of the video was of less concern than the quality of the images taken of the line. It was also verified that the code was saving the images to the correct directory. The images came out to be very clear, as can be seen from Figure 25.

After this was confirmed then the next tests run were to ensure that an accurate timestamp is being printed onto each image. Played around with different colors and font of the time stamp. A brighter color seemed to work most effectively because images had dark backgrounds while others had a lighter background and with a bright color like red it could show up on both. Hundreds of images were taken throughout numerous testing days and a couple examples are shown with a time stamp on them in comparison to various backgrounds in Figure 26.

D. DOC Control System

At one point the developers added a remote infrared sensor that would control the start/stop of DOC. When this code was written, a number of different test compiles were required to effectively obtain signals from the remote to the sensor. This testing included a great deal of rewriting, specifically for the parameters of the input signal being received by the IR sensor. Ultimately these tests were unsuccessful because the sensor could not provide a consistent enough connection with the remote signal.

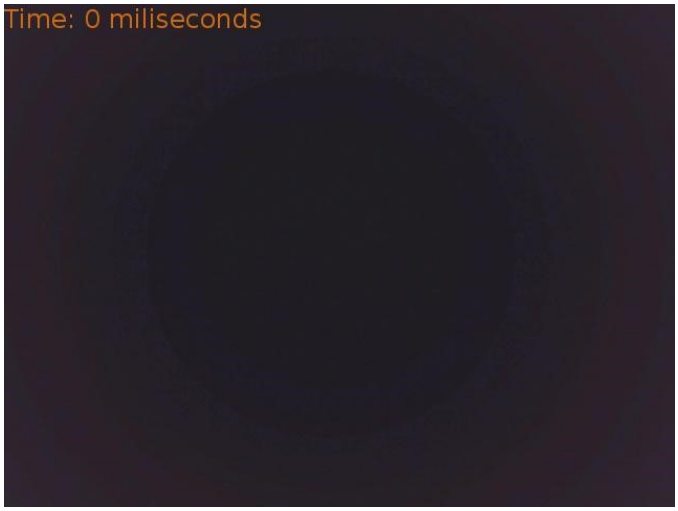
When the IR sensor was implemented, original testing of the code was performed to make sure it worked correctly. After confirming that voltage was actually received, then the next step was to test and

(a) Example 1

(b) Example 2

(c) Example 3

Figure 25: Example images of simulated line



(a) Example 1

(b) Example 2

Figure 26: Example timetamps overlaid on images

rewrite the conversion function based off the data obtained being compared to physically measured data. This code didn't have too much debugging included in it. The main thing was after calculating the correct multiplication factor for the conversion, tweaking that number to get a more accurate measurement reading. Once this was accurate, the developer integrated this with the other code and ran multiple test to make sure it still worked effectively.

XX. Conclusion

Electrical energy is becoming increasingly important to modern society. To maintain a reliable supply of electrical energy, the infrastructure that carries electrical energy to the consumer must be

well maintained. High tension power lines are a major component of that infrastructure. Power line inspection is dangerous because it presently requires manual inspection. Recent technological advances make the creation of robotic inspection platforms that can reduce the danger of power line inspection feasible.

From the time the developers first saw the need for a system that could help reduce the number of workers sent into hazardous environments to inspect transmission cables, they have been working to realize an idea that would achieve that goal. That idea, originally conceptualized last year, has been refined, expanded and finally settled through the last year as the system was developed and tested. Initially a spherical airborne inspection robot was proposed. Dubbed The SphereBot, a laboratory prototype was designed and built over the course of a semester. The SphereBot was composed of a thrust-vectored spherical frame, a user interface, a control system, an attitude stability sensing system, and an onboard camera. Testing of the SphereBot laboratory prototype revealed deficiencies in the original design that would need to be corrected to make the SphereBot a viable inspection platform.

The first deficiency identified was a poor thrust to weight ratio due to the weight of the laboratory prototype's ABS plastic frame. To improve the thrust to weight ratio the ABS frame of the SphereBot was replaced with a new carbon fiber frame, designed using the original frame as a model. The onboard camera, originally meant for line inspection, was removed from the SphereBot and mounted on a proposed addition to the inspection system. The proposed addition, a line crawler system, was developed to be deployed by an aerial deployment device. The revised SphereBot, with the carbon fiber frame, was tested and the reduction in the weight was found to have greatly improved the thrust to weight ratio, which allowed the SphereBot to achieve flight. Despite the improved thrust to weight ratio additional problems with the stability of the system appeared once flight was achievable. Experimentation found that the instability in the SphereBot was being caused by vibration from the mechanical components. This vibration caused errors in the values obtained from the internal measurement unit. This issue was reduced by securing loose components and by enabling filters on the accelerometer and gyroscope devices onboard the IMU. Despite the reduction in system vibration, the IMU instability has not yet been fully solved. The user interface was also tested and proven to control the movement of the SphereBot.

In the second semester an addition to the line inspection system was proposed to expand the available sensors and, therefore, the inspection capability. The addition consists of a line crawler, later named DOC, and allows for more detailed line inspection. To ease development of the line crawler, two additional team members were brought onto the team to tackle the task of building the line crawler. The requirements for the line crawler included the ability to travel on the transmission line independently, the ability to acquire and store images of the line, and the ability to be deployed onto the line. This system was rapidly developed from the initial design to a frame with components mounted on board to allow the line crawler to meet its feature set. Once a prototype had been developed, DOC's features were tested and minor engineering solutions were made as problems were identified that prevented DOC from fully meeting its feature set. The solutions, to testing deficiencies, that were employed allowed the line crawler, like the SphereBot, to fully meet its original feature set.

Despite the difficulties encountered during the development of the line inspection system, the SphereBot has met the goals of Senior Design set out by the team at the beginning of the fall 2013 semester. DOC is also fast approaching the end of its development with all of the basic features met. More development is required for real world application, but the project, in its current state serves as a solid foundation on which to build a robust system that can greatly reduce the number of lives put at risk in the course of inspecting high voltage transmission cables.

ACKNOWLEDGMENT

The SphereBot team would like to thank professors Tatro and Belkhouche for their invaluable mentorship during this experience. The team would also like to thank Tyler Anderson and Jim Ster for their assistance in producing the enhanced carbon fiber frame.

APPENDIX A
GLOSSARY

AHRS Attitude and Heading Reference System, a device which provides indication of rotational and translational position; often referred to as an IMU, but distinguished from an IMU in that the AHRS has integrated computational capability.

Arduino A family of single-board microcontroller systems which provide a consistent platform for rapid prototyping and hobbyist development.

ARM A family of system-on-chip RISC architectures targeted at embedded microcontroller needs in small devices. Formerly an acronym for Acorn RISC Machine, but now recognized by the given name.

C A compiled programming language originally developed circa 1970 by Dennis Ritchie. Provides common programming structures and little abstraction from the execution platform. Allows small, fast, and efficient programs, at the cost of higher-level features.

condition variable A mechanism in programming languages to indicate freed or available resources shared between concurrent processes. In C, provided by the `phreads` library.

debugger A development tool which allows a user to monitor the state of a program, generally while stopped at a breakpoint.

eHRPWM enhanced High-Resolution Pulse-Width Modulator, a feature of many ARM processors which produces PWM output

ESC Electronic Speed Controller, a motor control device which regulates the speed of a motor as a servomechanism, often using a standard RC servo interface.

FLOPS Floating-Point Operations Per Second — a rough measure of the computational throughput of which a processor is capable

FPGA Field Programmable Gate Array, a logic device which which can be user-configured to implement arbitrary logic circuits. Used to allow flexibility and rapid development of hardware-implemented logic.

FTDI A series of adapters which provide serial connectivity over a USB interface. Formerly an acronym for Future Technologies Digital Interface, but now recognized by the given name.

GPU Graphics Processing Unit — a purpose-specific processing unit optimized for massively parallel operations required for graphics computation

IMU Inertial Measurement Unit. A device which uses a combination of accelerometers and gyroscopes to determine the attitude and velocity of a system.

LiPo Lithium Polymer — a category of rechargeable battery with high energy density, high power capability, and low memory effects

MCU Microcontroller Unit.

mutex A mechanism in programming languages to ensure mutually exclusive access to resources shared between concurrent processes. In C, provided by the `pthreads` library.

PID Proportional Integral Derivative, a control system which uses a feedback mechanism to minimize the error between an applied setpoint and the measured output of a process.

PWM Pulse Width Modulation, a signal processing method which regulates the duty cycle of a periodic rectangular waveform; required for interfacing with RC servos.

RISC Reduced Instruction Set Computing, a class of processor architectures which provide a minimal instruction set in order to reduce the transistor count, silicon size, and power dissipation of the processor. Contrasts with Complex Instruction Set Computing (CISC), which trades a larger, hotter processor for a potential reduction in the number of machine instructions required for any given task.

servo servomechanism; a device which acts to minimize error between a measured parameter and a setting; in the case of RC servos, the setting is applied by a PWM signal with a period of 20 ms and a pulse width which varies about a nominal 1500 µs from a minimum of 1000 µs to a maximum of 2000 µs.

`bash` Bourne Again SHell, a popular descendant of the original Bourne shell used on Unix and Unix-like systems

`/sys` A virtual filesystem which exposes device and kernel state to userspace. Used in Linux and some other Unix-like operating systems to provide a standard interface to hardware and other features.

windup a condition in a PID controller in which uncorrectable errors accumulate, leading to overcorrection.

APPENDIX B
VENDOR CONTACTS

May 4, 2014

To: The Intel Corporation

Re: Thank you for the grant

Dear Intel,

We of Team SphereBot from the 2013–2014 Senior Design class at California State University, Sacramento would like to extend our heartfelt thanks for the grant that you provided our team. The grant money was used to help develop a lightweight carbon fiber frame for our spherical inspection robot. Without the lightweight frame our robot would not have been able to fly. Your grant, therefore, was instrumental in completing our Senior Design successfully. Your generosity reflects greatly on your company and sets you apart from the rest of the industry. Thank you.

Sincerely,

Aaron Diab, Robert Wortman,
Darrell Cahaill, Rebecca Wingo, and
Emmanuel Dupart

May 4, 2014

To: James Ster, Equipment Technician, CSU, Sacramento

Re: Thank you for your assistance.

Dear Mr. Ster,

We of Team SphereBot from the 2013–2014 Senior Design class at CSU, Sacramento would like to extend our heartfelt thanks for your advice and assistance in designing both the SphereBot and Line Crawler robots as well as your recommendation of Tyler Anderson to help redesign the SphereBot frame. Your help greatly facilitated the creation of both of our systems and was instrumental in completing our Senior Design successfully. Your extensive experience, knowledge, and willingness to help reflect greatly on yourself, the college of engineering and computer science, and the California State University, Sacramento. Thank you.

Sincerely,

Aaron Diab, Robert Wortman,
Darrell Cahaill, Rebecca Wingo, and
Emmanuel Dupart

May 4, 2014

To: Tyler Anderson, ME student, CSU, Sacramento

Re: Thank you for your assistance.

Dear Mr. Anderson,

We of Team SphereBot from the 2013–2014 Senior Design class at CSUS would like to extend our heartfelt thanks for your hard work in helping us design and manufacture of our new super light and strong SphereBot frame. Your help was crucial in our success in creating the SphereBot and in successfully completing our Senior Design.Your professionalism, competence in your field, and generosity reflect greatly on yourself, the mechanical engineering department, and the California State University, Sacramento. Thank you.

Sincerely,

Aaron Diab, Robert Wortman,
Darrell Cahaill, Rebecca Wingo, and
Emmanuel Dupart

APPENDIX C
RESUMÉS

A. Robert Wortman

Robert Wortman

Education

1996–1998 **Graduate**, *Naval Nuclear Training Program*.
- *Naval Nuclear Power Training Command*, Orlando, FL
- *Nuclear Power Training Unit*, Ballston Spa, NY

2005–2011 **transferred**, *American River College*, Sacramento, CA, *3.967 GPA*.

2011–present **B.S. (in progress)**, *California State University, Sacramento*, *3.895 GPA*.
- major in Computer Engineering

Experience

1998–2001 **Nuclear Electronics Technician**, *United States Navy*, U.S.S. Tucson (SSN-770).
- As qualified Reactor Operator and Reactor Technician, operated and maintained Reactor Controls Division equipment, including reactor and nuclear instrumentation, steam generator control, and the reactor plant which provided ship's electrical energy and propulsion.
- Assisted in preventative and corrective maintenance on Electrical division equipment, including various motors, generators, and distribution panels.
- Assisted in Nucleonics Division activities, including reactor coolant and steam generator chemistry sampling.
- As Logroom Yeoman, tracked and maintained Engineering Department procedural and systems documentation, and organized manual data acquisition and verification for reactor, electrical, and propulsion plant systems.

2002 **Nuclear Electronics Technician**, *United States Navy*, Pearl Harbor Naval Shipyard.
- As Physical Security Office front desk clerk, performed personnel authorization and identification services in accordance with shipyard security procedures.
- With shipyard information technology division, implemented secure decommissioning procedures to allow outgoing equipment to be released to local educational activities.

2003–2004 **Sales Clerk**, *HSC Electronic Supply*, Sacramento, CA.
- Maintained sales floor displays.
- Operated point-of-sale system for customer purchases.

2012–present **Student Researcher**, *California Smart Grid Center*, Sacramento.
- Create data acquisitions systems to relay data from sensors and other devices to a relational database.
- Create programming interfaces for gathered data to support other researchers.
- Lead Mobile Application Team in developing tools to provide context for consumer decisions.
- Lead Dispatchable Load Team in developing system to store energy during periods when generation from renewable sources exceeds energy demand.

Skills

- Linux system administration
- Network administration
- C and C++ programming
- Unix shell scripting
- analog and digital electronic theory

✉ *wortmanr@ecs.csus.edu*

B. Aaron Diab

AARON DIAB

OBJECTIVE: To obtain a computer engineering career position

EDUCATION

American River College
Associate in Science • Computer Science • Highest Honors • May 2010

California State University, Sacramento
Bachelor of Science • Computer Engineering • 3.68 GPA • June 2014

Related Courses:

• Advanced Computer Organization	• Computer Interfacing	• Robotics
• Operating Systems Principles	• Signals & Systems	• Data Structures
• Advanced Logic Design	• Operating System Pragmatics	

SKILLS

Languages: • C • C++ • Java • Objective C • bash • Verilog • VHDL • Python • LaTeX
Design: • MIPS ISA • CPU • iOS • Android
Software: • MATLAB • Multisim • MPLAB • Xcode • Eclipse • gcc • gdb
Hardware: • Microchip • Xilinx • Raspberry Pi • ATmega • Parallax Propeller • x86

WORK EXPERIENCE

Lead Biomedical Equipment Support Specialist, VA Medical Center, Mather, CA **08/05 - Present**
Lead technician and special project manger in charge of work distribution and providing additional technical troubleshooting support to subordinate technicians and interns. Plan, purchase, and implement advanced medical equipment and related computer workstation and server systems.

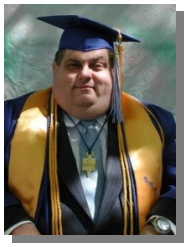
Student Researcher, California Smart Grid Center, CSUS, Sacramento, CA **06/12 - Present**
Student researcher charged with developing mobile applications on iOS and Android platforms to monitor real time energy usage and predict future usage. Designed and coded prototype iOS application to monitor real time energy usage. Developed plans to extend current application to provide projected energy usage with sensor input and to provide feedback to the consumer.

Medical Equipment Repairer 91A10, U.S. Army **04/98 - 08/04**
Assembled, configured and maintained a wide variety of advanced medical devices and related computer systems. Provided educational support and training to medical equipment end users. Coordinated lodging and service activities while managing team of mobile service technicians.

Medical Equipment Repairer 91A10, U.S. Army Natl Guard, CA **11/95 – 04/98**
Assembled, configured and maintained a wide variety of advanced medical devices and related computer systems. Provided educational support and training to medical equipment end users.

Darrell Cahail

3/22/2013



1830 T Street #4, Sacramento, CA 95811
Cell: (916) 969-4358
dlc88@saclink.csus.edu

INTRODUCTION

Goal-focused professional offering years of experience in the computer industry with excellent technical and communication skills. I am motivated and enthusiastic by new challenges and tasks and take a proactive approach to achieve success in all projects. I enjoy working in a complex projects with significant technical and knowledge challenges. Significant experience in working with different operating system and platforms including Windows, UNIX, Linux and DOS.

OBJECTIVES

Engage in a challenging and high performance oriented role in the field of Computer Engineering and implement the expertise and experience gained in this field to develop complex project with efficiency and quality.

EDUCATION

Modesto Junior College

May 2009
Associate of Science, Industrial Electronics
Associate of Arts, General Education
Certificate of Achievement, TRIO Program
TRIO Student Support Services, Student of the Year
Dean’s List Summer 2006 – May 2009
Scholarship Award, Faculty Emeritus, 2009-2010

California State University, Sacramento

Spring 2014

EXPERIENCE

Student Researcher | **California Smart Grid Center**
November 2011 - Current

I was responsible for system designing, code review and test review. I managed development team issues. I also took the task of writing complex programs or modules; while maintaining the research computers. I also took the responsibility of imparting training to new team members as assigned by project manager of my project. I was involved in the generation of research documentation and planning.

STRENGTHS

- ☐ Excellent communication skills - present points precisely and clearly
- ☐ Exceptional problem solving ability and analytic skills to solve problems efficiently
- ☐ Good team player with excellent interaction skills to coordinate and work within a team
- ☐ Superior Technical Skills
- ☐ Expertise in working with various operating systems

SKILLS

- ☐ Languages: C, C++, Java, .NET, JavaScript, HTML, CSS, JDK, Verilog, VHDL
- ☐ Assembly: x86, MIPS, Microchip, Atmel
- ☐ Databases: MySQL, Access, Microsoft SQL
- ☐ Operating System: UNIX, Linux, Windows, DOS
- ☐ Electronics Tools: Multisim, LabVIEW, MATLAB, Spice
- ☐ Design: UML
- ☐ Productivity Tools: Microsoft Office Suite, Libre Office Suite
- ☐ Have sound knowledge of analog and digital circuit design
- ☐ Have sound knowledge in networking protocols and device programming
- ☐ Have experience in working with C and C++ compiler programming and system level programming

D. Rebecca Wingo

Rebecca Wingo

rebeccawingo@csus.edu

OBJECTIVE

An entry position in Electrical Engineering.

EDUCATION

In progress: BS, Electrical and Electronic Engineering[+], CSU Sacramento, GPA 3.640

[+]Expected May 2014

Related Courses

Product Design Project II*	Introduction to Microprocessors
Product Design Project I	Introduction to Feedback Systems
Digital Control Systems*	Signals and Systems
Robotics	Network Analysis
Electronics II	Introduction to Logic Design
Modern Communication*	Engineering Mechanics
Applied Electromagnetics	Electromechanical Conversion

* Current Semester

SKILLS

Communication/Organization:

*Strong written and oral communication skills: able to work with multi-discipline teams
*Able to keep track of multiple deadlines

Leadership/Management:

*Strong management skills: able to organize a team of people to teach groups of young students
*Organized in crisis

Tools:

MATLAB * PSpice * Multisim * Oscilloscope

Programming:

C/C++ * VHDL * Assembly Language

WORK EXPERIENCE

Student Researcher	Smart Grid Center	4/13-Present
File Clerk	Quilt N' Home	6/09- Present
Trigonometry/English Tutor	Cerro Coso Community College	9/11- 12/11

PROJECTS

High-Tension Power-Line Inspection Robot*

Member of a five person multi-discipline computer and electrical engineering team designing, building, and testing an airborne robot with a deployable line crawler to perform hazardous inspections remotely.

Solar Regatta- Green Flash*

Member of an eight person multi-discipline mechanical and electrical student team designing and building a solar-power boat, named the Green Flash, to win the Northern California Solar Regatta put on by SMUD.

The Frankenbot

Member of a two person team which designed and built a multi-purpose wheeled robot to compete in a series of challenges, including line following, a maze, a robot convoy, and a race.

The Top-Hat Robot

Member of a three person team which designed and built a robot driven by the user via remote control or autonomously using an ultrasonic sensor and bumper switches.

***** Current Project

ACCOMPLISHMENTS and ACTIVITIES

- Dean's Honor List at CSU Sacramento
- Tau Beta Pi member
- DEI Honor's Society

Emmanuel George Dupart

EDUCATION

Bachelors of Science in Electrical & Electronic Engineering, GPA: 3.103
California State University, Sacramento, *Expected Graduation Date: Dec. 2014*

Related Courses

Introduction to Microprocessors	Electronics I w/ Lab	Robotics w/ Lab
Physical Electronics	Program Concept+Method I & II	Engineering Economics
Signals and Systems Analysis	Engineering Economics	*Digital Control Systems**
Electromechanic Conversion	Electromechanics Lab	*Introduction to Machine Vision**
Probability and Random Signals	Electronics II w/ Lab	*Senior Design I & II w/ Lab**
Network Analysis w/ Lab	Introduction to Feedback Systems	*Modern Communication Systems**

Spring 2014

Affiliations

- Cooper-Woodson College Student Association, 2013/14, **Vice-President**
- National Society of Black Engineers (NSBE)

EXPERIENCE

Student Assistant, Sacramento State University Ethnic Studies Dept., Sacramento, CA 2012-Present
Job Tasks

- Coordinated multiple large scale lectures, seminars, meetings, and events that helped educate the campus community and promote awareness of resources in the community
- Provided excellent customer service by greeting visitors, helping them make appointments, getting them the information they need, and taking care of their needs
- Organized supplies, materials, paperwork throughout the office, and confidential information in a program database

Summer Orientation Leader, First Year Experience Program, Sacramento, CA Summer 2012
Job Tasks

- Developed understanding of individual majors and all graduation requirements specific to each major
- Explained complex information in a clear and concise way, to people with varying levels of communication skills
- Led daily tours of campus, explaining different organizations, clubs, and services and answering questions
- Coordinated and facilitated daily group exercises for group morale and introductions

Electronics Sales Floor Associate, Wal-Mart, San Leandro, CA 2007-2009
Job Tasks

- Listened to customers in order to better assist them and provide them with their needs in the most effective manner and provide excellent customer service
- Organized displays with movies, games, music, and more in order to maintain a safe and organized work environment
- Coordinated and oversaw motivational team building exercises

PROJECTS

- **Solar Tracker** (from *Intro to Microprocessors)*
- **Pendulum Stabilization Micro-Mechtronics/Controls Project*** (from *Digital Control Systems)*
- **High Tension Power Line Crawler Bot*** (from *Senior Design*) *In-Progess

SKILLS

- C and C++ Programming
- MATAB
- Parallax Propeller
- x86
- Python on OpenCV
- ChipKIT uC32 & WF32
- Analog Discovery
- Arduino
- Raspberry Pi Model B

REFERENCES

[1] F. Time, "The salary of a high voltage cable inspector," *Houston Chronicle*, accessed Sep. 8, 2013. [Online]. Available: http://work.chron.com/salary-high-voltage-cable-inspector-21086.html

[2] G. Stix, "Working hot: life at 765 kv," *IEEE Spectrum*, pp. 54–56, Sep. 1988, accessed Sep. 8, 2013. [Online]. Available: http://simson.net/ref/1988/IEEE_Working_Live.pdf

[3] (2012, May) 49-9051 electrical power-line installers and repairers. Bureau of Labor Statistics. Last modified March 29, 2013. [Online]. Available: http://www.bls.gov/oes/current/oes499051.htm

[4] (2013) Fatal occupational injuries, total hours worked, and rates of fatal occupational injuries by selected worker characteristics, occupations, and industries, civilian workers, 2012. Bureau of Labor Statistics. From Census of Fatal Occupational Injuries, 2013. [Online]. Available: http://www.bls.gov/iif/oshwc/cfoi/cfoi_rates_2012hb.pdf

[5] (2012, Mar.) Line installers and repairers. Bureau of Labor Statistics. From Occupational Outlook Handbook. [Online]. Available: http://www.bls.gov/ooh/installation-maintenance-and-repair/line-installers-and-repairers.htm

[6] (2006, Apr.) Burn injury facts: Arc flash / blast. Washington State Department of Labor. [Online]. Available: http://www.lni.wa.gov/Safety/Research/Files/ArcFlashHazardReport.pdf

[7] P. Giovinazzo, "Electrical safety: The fatal current," *Bulletin*, vol. 2, no. 13, Feb. 1987, provided to Ohio State University by Elmwood Electric. [Online]. Available: http://www.physics.ohio-state.edu/~p616/safety/fatal_current.html

[8] (2009, May) Transmission facts. American Electric Power. [Online]. Available: http://www.aep.com/about/transmission/docs/transmission-facts.pdf

[9] N. F. Marks, H. Jun, , and J. Song, "Death of parents and adult psychological and physical well-being: A prospective u.s. national study," *Journal of Family Issues*, vol. 28, no. 12, pp. 1611–1638, Dec. 2007. [Online]. Available: http://jfi.sagepub.com/content/28/12/1611

[10] J. Katrašnid, F. Pernuš, and B. Likar, "A survey of mobile robots for distribution power line inspection," *IEEE Transactions on Power Delivery*, vol. 25, no. 1, pp. 485–493, Jan. 2010. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5345712

[11] *IEEE Guide for Reducing Bird-Related Outages*, IEEE Power & Energy Society Std. 1652-2010, Feb. 22, 2011. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5724322

[12] Y. Kobayashi, G. G. Karady, G. T. Heydt, and R. G. Olsen, "The utilization of satellite images to identify trees endangering transmission lines," *IEEE Transactions on Power Delivery*, vol. 24, no. 3, pp. 1703–1709, Jun. 2009, xnote. [Online]. Available: Xurl

[13] L. Tavares and J. ao Lilva Sequeira, "RIOL – robotic inspection over power lines," in Proceedings from 6th IFAC Symposium on Intelligent Autonomous Vehicles. Toulouse, France: Institute for Sytems and Robotics, Lisbon, 3–5 2007. [Online]. Available: http://welcome.isr.ist.utl.pt/pub/?accao=showpublication&id_publication=1704

[14] M. Bühringer, J. Berchtold, M. Büchel, C. Dold, M. Bütikofer, M. Feuerstein, W. Fischer, C. Bermes, and R. Siegwart, "Cable-crawler: robot for the inspection of high-voltage power lines that can passively roll over mast tops," *Industrial Robot: An International Journal*, vol. 37, no. 3, pp. 256–262, 2010. [Online]. Available: http://publications.asl.ethz.ch/files/buehringer10cable.pdf

[15] R. Tatro, "Problem statement," Internally published on Hive, College of Engineering and Computer Science, CSU, Sacramento, May 30, 2011. [Online]. Available: http://ecshive.ecs.csus.edu/

[16] ——, "Design idea contract," Internally published on Hive, College of Engineering and Computer Science, CSU, Sacramento, May 30, 2011. [Online]. Available: http://ecshive.ecs.csus.edu/

[17] NetMBA.com. Work breakdown structure. Web page. Internet Center for Management and Business Administration. [Online]. Available: http://www.netmba.com/operations/project/wbs/

[18] D. L. Cook, "Program evaluation and review technique - applications in education," U.S. Department of Health, Education and Welfare, Report, 1966. [Online]. Available: http://www.eric.ed.gov/

[19] R. Tatro, "Work breakdown structure," Internally published on Hive, College of Engineering and Computer Science, CSU, Sacramento, May 30, 2011. [Online]. Available: http://ecshive.ecs.csus.edu/

[20] ——, "Project timeline," Internally published on Hive, College of Engineering and Computer Science, CSU, Sacramento, May 30, 2011. [Online]. Available: http://ecshive.ecs.csus.edu/

[21] ——, "Bread board proof," Internally published on Hive, College of Engineering and Computer Science, CSU, Sacramento, May 30, 2011. [Online]. Available: http://ecshive.ecs.csus.edu/

[22] ——, "Midterm technical design review," Internally published on Hive, College of Engineering and Computer Science, CSU, Sacramento, May 30, 2011. [Online]. Available: http://ecshive.ecs.csus.edu/

[23] ——, "Laboratory prototype documentation," Internally published on Hive, College of Engineering and Computer Science, CSU, Sacramento, Nov. 20, 2012. [Online]. Available: http://ecshive.ecs.csus.edu/

[24] ——, "Laboratory prototype presentation," Internally published on Hive, College of Engineering and Computer Science, CSU, Sacramento, Nov. 19, 2012. [Online]. Available: http://ecshive.ecs.csus.edu/

[25] "chipkit uc32 board reference manual," Technical Manual, Digilent, Pullman, WA, Jul. 17, 2012. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/chipKIT%20uC32_rm.pdf

Robert Wortman has extensive knowledge of the Linux development environment, and programs fluently in C. He is academically familiar with aeronautics and flight dynamics, and has a professional background in thermodynamics, fluid flow, and electronic theory.

Rebecca Wingo is an EEE major focusing in control systems and electronics. She has experience using microcontrollers and sensors to control robots as well as some experience in analog and digital circuitry. In addition, her experience as a student researcher at the California Smart Grid has exposed her to a number of different engineering projects and communication styles.

Aaron Diab has been professionally employed as a biomedical equipment/electronics technician for 17 years. Much of this time was as a soldier in the United States Army where he learned to strive for professionalism and attention to detail in all pursuits. This professional experience has provided him with a rich background in constructing, troubleshooting and repairing mechanical and electro-mechanical assemblies and equipment. This background coupled with his studies in computer engineering in general and iOS programming in particular provide him with a unique skill set that make him a valuable asset to Senior Design Team 1.

Emmanuel George Dupart is a senior Electronic/Electrical Engineer with a special interest in Controls and Feedback Systems. He has worked on numerous projects in his academic career ranging from control system projects, to full robotic projects, and even machine vision projects. These projects include, but are not limited to, implementing a digital control system to stabilize a pendulum, building a fully automated robot which included various sensors, and automatic detection of specified line faults using a camera system and a line crawler robot.

Darrell Cahail has extensive knowledge in Linux, Java, C/C++, programming IDEs, and hardware interfacing with a focus being on control and feedback systems. He programs most fluently in C/C++ with Java coming in a close second.