SARS:
Search and Retrieval System

Senior Design II Spring 2015

Group 4

Matt Bahr
Brian Crabtree
Brendan Hall
Erick Makris

Sponsored By:

Boeing

SoarTech

Table of Contents

1. Executive Summary

The decision to build a Search and Retrieval System (cleverly abbreviated as SARS) was made after a significant amount of brainstorming on the part of Group 4, also referred to in this document as the SARS Group. All group members wanted to take on a challenging project that would provide valuable experience and catch the attention of potential employers. Almost all of the ideas that were thrown around during these initial brainstorming sessions had to do at least in part with self-guided vehicles or artificial intelligence. The fact that SoarTech offered a sponsorship to groups with projects displaying principles of artificial intelligence was a major incentive to implement a system that involved smart robots. The team chose to build SARS because it offered the opportunity to program communications between two autonomous vehicles: a quadcopter in the air and a rover on the ground.

A basic explanation of the system is that it will involve the interfacing of three separate subsystems. The first of the subsystems is a quadcopter, or SARS Copter, which will hover in the air and use a camera to scan the ground for a target object to be retrieved. The second is a rover, or SARS Rover, which will receive the GPS coordinates of the target object from the SARS Copter once it has located the object. The SARS Rover will then travel to the coordinates to retrieve the target object. After most of the design of the SARS Rover subsystem had been completed, the SARS Group decided upon a tennis ball covered in hook and loop fasteners and lying on top of a white poster board as the target object to be retrieved. The third subsystem is a Linux-based console application, called the Command Distribution Center (CDC), which shall be used to monitor the functioning of SARS and to display diagnostic information on both the SARS Rover and the SARS Copter. The application shall also display a video feed from the camera used to scan the ground.

Early in the research phase of the project development, the SARS Group identified the main concerns in implementing this system and distributed the tasks involved in each among the group members. These challenges were as follows:

- Image processing to identify the target object from the air
- Wireless communications using both Wi-Fi and radio communications
- Automated GPS navigation for both the SARS Copter and the SARS Rover
- Object retrieval by the SARS Rover
- Power distribution in both the SARS Copter and the SARS Rover
- Application development

Over the course of the past two semesters, the SARS Group has managed to build a proto-type which satisfies almost all of the original specifications and successfully functions as a proof of concept. With more time and a larger budget, a more robust prototype could be produced; however, due to the lack of time and funds, certain sacrifices and considerations had to be made. The SARS Rover's search algorithm which handled the search for the target object once the rover reached its GPS coordinate did not perform as well as was originally intended; likewise, weather-proofing the SARS Copter and SARS Rover to the point where they could perform in the rain proved too difficult.

2. Project Narrative Description

Group 4's senior design project, SARS, is a multi-faceted aerial and ground object detection and retrieval system composed of three primary components: a quadcopter, a ground rover, and a Linux-based console application. All three aspects of the system work in conjunction to autonomously locate an object, retrieve it, and provide real-time diagnostics and a live video stream of the mission to the user.

The first part of our project, the SARS Copter, is responsible for gathering images of the search area. Mounted with a high quality camera, the SARS Copter is fed a series of waypoints from the Mission Planner application for controlling UAVs, dictating the area which it will need to search for the object. As the copter pauses at each waypoint, the CDC pulls frames from the camera's video feed. These images are then processed to determine at which waypoint the target object is located. Once the object is identified, its waypoint is relayed to the SARS Rover over Wi-Fi. The SARS Copter is composed of six primary components: the copter itself, an Internal Measurement Unit which analyzes real-time telemetry, allowing the quadcopter to stay in flight, a camera which provides live video streaming and images to be processed for object detection, a telemetry device which communicates with the CDC, a GPS chip which relays the quadcopter's location to the user via the CDC and allows for automated flight to mission waypoints, and finally the Pixhawk flight controller.

The SARS Rover is the aspect of the project which physically retrieves the object at its location. Once the object's waypoint information has been transmitted from the CDC, the rover travels to the object's location and picks it up using an attached arm. After retrieving the item, the rover returns to its original location. Throughout the entire mission, the rover constantly relays information about its position and speed to the user via the CDC. The main components of the rover are the rover itself, the retrieval arm, the microcontroller, the Wi-Fi communication module, the GPS and compass, and an ultrasonic sensor for locating the object once the SARS Rover has reached the object's approximate GPS location.

The final piece of the project is the CDC console application. The CDC displays real-time diagnostics to the user about the two vehicles while providing a live aerial video stream from the SARS Copter. The only necessary user interaction is the takeoff and landing of the SARS Copter and the power on and off of all subsystems. Aside from these actions, the system performs in a completely autonomous manner.

An ambitious project, SARS has provided all four group members with valuable experience in many different arenas, from application development and UI design to wireless communications, object detection, hardware/software integration, and artificial intelligence. Special thanks go out to SoarTech and Boeing for funding the majority of the project budget. The development of SARS would not have been possible without the help of both companies.

3. Project Goals and Specifications

3.1. Project Goals and Objectives

After careful consideration and much discussion, Group 4 decided upon the following goals and objectives for the SARS project. The main consideration in coming up with these goals was the setting in which the project is being developed. As SARS was being implemented in a classroom setting rather than a market setting, its main objective were to meet all specifications and to perform in a manner that would secure each member of Group 4 an A grade in Senior Design. The following list specifies all project goals and objectives.

- To pass Senior Design with an A grade.
- To attract potential employers.
- To implement an effective proof of concept for a search and retrieval system with multiple real life applications at an affordable price.
- To use computer vision as a means of locating objects.
- To program effective communications between SARS subsystems.

3.2. Project Specifications

SARS consists of three main subsystems: a quadcopter, a land rover, and a Linux-based console application. Successful communications between these three subsystems were vital to the project implementation.

3.2.1. SARS Copter

Group 4 decided upon the following specifications for the SARS Copter subsystem. These specifications were intended to allow the SARS Copter to complete the tasks of identifying an object on the ground and relaying its GPS location to the CDC subsystem. The only specification for this subsystem that went unmet was the weather-proofing requirement. Being able to run a mission in bad weather ended up being pointless as none of the project committee members could be expected to watch the project demonstration in the rain.

- Capable of interfacing wirelessly with a Linux-based console application.
- Capable of taking high quality videos/photos.
- Has camera stabilization to facilitate image processing.
- Capable of identifying and locating an object on the ground and calculating its GPS coordinates accurate to within 3 m.
- Capable of hovering at a constant height between 5 ft. and 10 ft. with a variation in height no greater than 3 in.
- Has battery life up to 10 minutes.
- Basic weather-proofing.

3.2.2. SARS Rover

Group 4 decided upon the following specifications for the SARS Rover subsystem. These specifications were intended to allow the SARS Rover to complete the tasks of receiving waypoint information from the CDC subsystem, traveling to the GPS coordinates of the received waypoint, and retrieving a target object located at those coordinates. Once again, the only specification that was not met was that of basic weather-proofing.

- Capable of interfacing wirelessly with a Linux-based console application.
- Capable of travelling up to 1000 ft. on a single battery charge.
- Has a retrieval subsystem for picking up the target object off the ground.
- Has a subsystem that uses a sensor to find the target object once it has reached the approximate GPS coordinates.
- Able to return to within 3 m of its starting location with the retrieved target object.
- Able to carry a load of 5 lbs.
- Basic weather-proofing.

3.2.3. Command Distribution Center (CDC)

Group 4 decided upon the following specifications for the CDC application. These specifications were intended to allow a user to interface with the SARS subsystems. The application had to provide a live video stream from the SARS Copter and display diagnostic information for the SARS Rover and the SARS Copter. Of the following specifications, the only one that went unmet was the provision of stop commands to the rover and quadcopter as well as the provision of a start command to the quadcopter. The reason for the lack of quadcopter commands was that at some point, the copter's flight controller was damaged, limiting its capabilities for autonomy. The only way to get the SARS Copter to travel to a series of waypoints was to take off and land manually. A stop command for the SARS Rover was not implemented as a result of the time constraints of the project.

- Provide start commands to the SARS Rover and SARS Copter
- View live video stream from quadcopter camera
- View GPS and other sensor data from quadcopter and rover
- Provide live updates of missions as they progress.

3.3. System Overview

SARS consists of 3 major subsystems: the SARS Copter, the SARS Rover, and the CDC. Each subsystem is diagrammed below. Each subsystem was assigned to a primary engineer, but various tasks relating to that subsystem were delegated to other members as needed.

Figure 3-1 below displays the system overview for SARS. Brendan Hall was responsible for the SARS Copter subsystem. Brian Crabtree was responsible for the SARS Rover subsystem. Matthew Bahr and Erick Makris worked jointly on the CDC application subsystem.

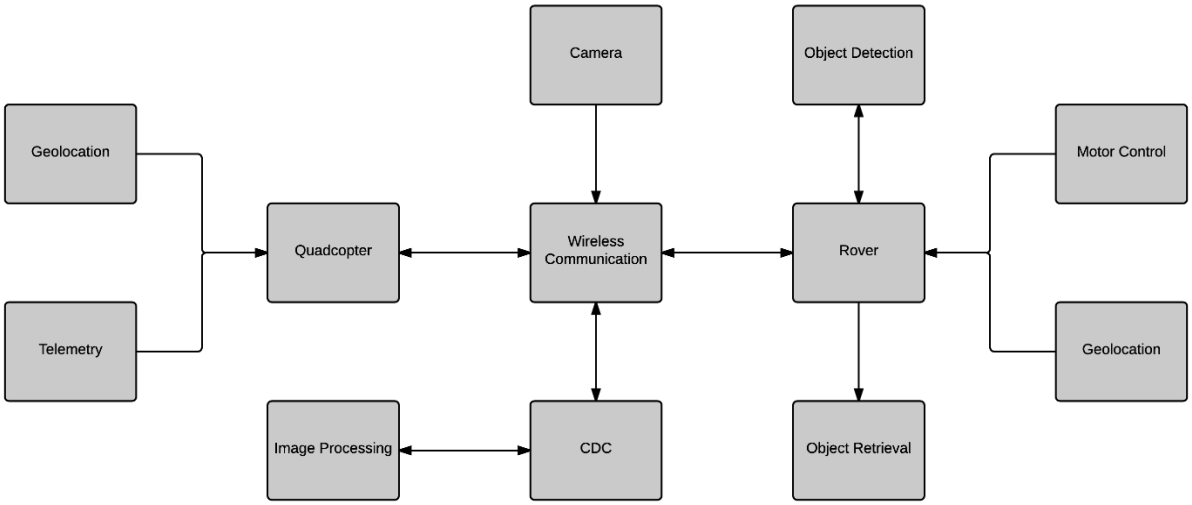

Figure 3-1: System Overview

Figure 3-2 below displays the portion of the block diagram corresponding to the SARS Copter's subsystems. As previously stated, Brendan Hall was primarily responsible for the SARS Copter's hardware and software components, and he was given support from both Matt Bahr and Erick Makris.

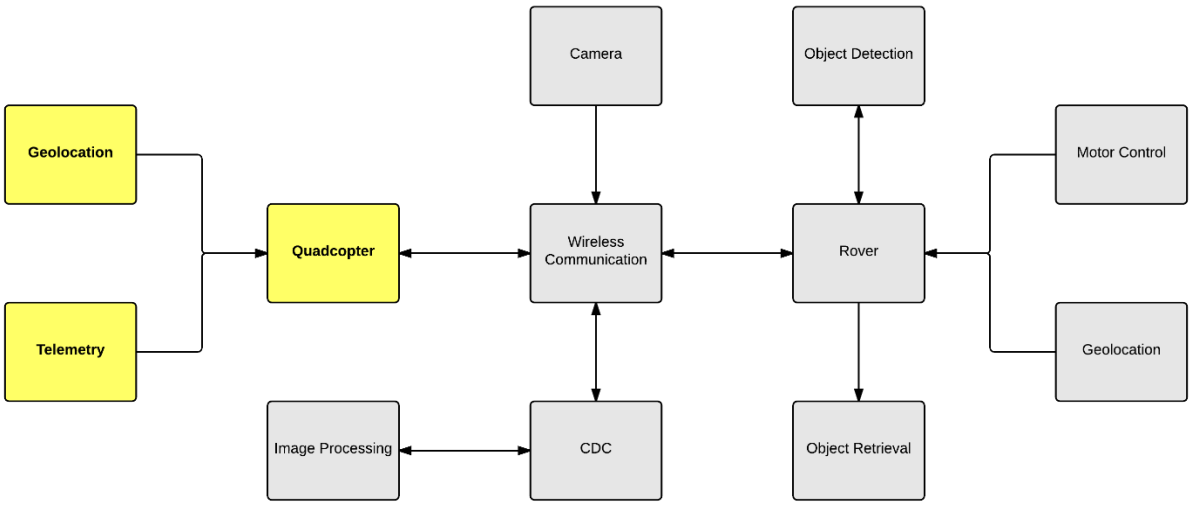

Figure 3-2: SARS Copter Subsystem

Figure 3-3 below displays the portion of the block diagram pertaining to the SARS Rover's subsystems. Brian Crabtree was primarily responsible for the SARS Rover's hardware and software components, and he was given support from Erick Makris.

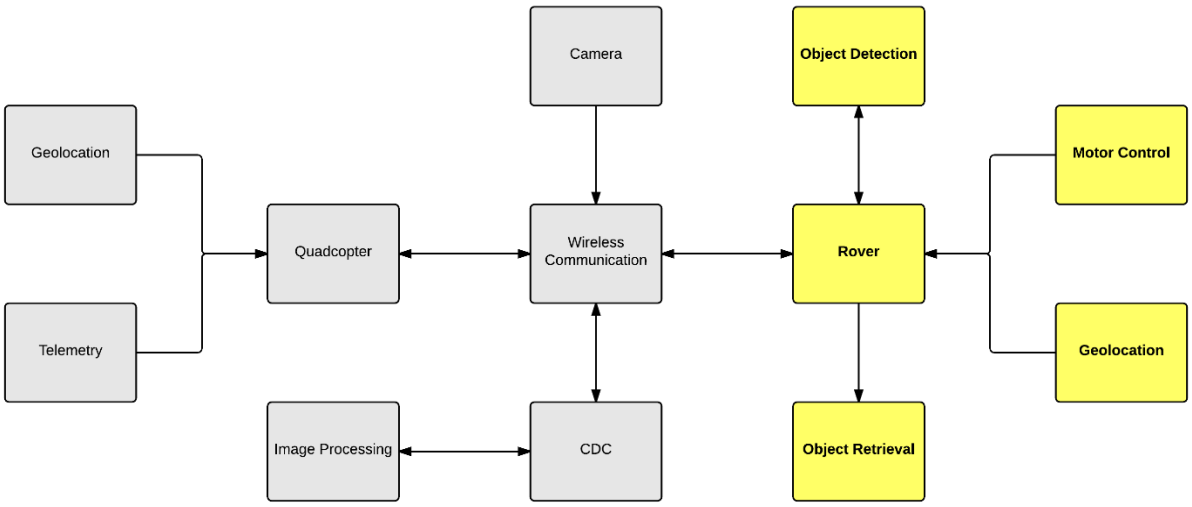

Figure 3-3: SARS Rover Subsystem

Figure 3-4 below displays the portion of the block diagram pertaining to the CDC application subsystems. Erick Makris was primarily responsible for the CDC subsystem, and Matthew Bahr was primarily responsible for the Image Processing and Camera subsystems. All four engineers were significantly involved with the Wireless Communication subsystem.

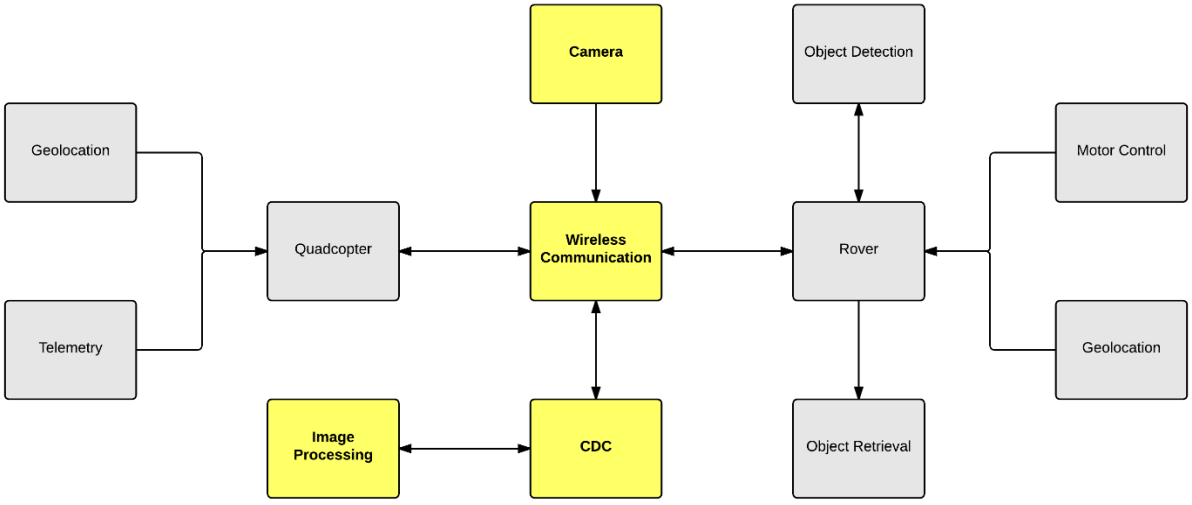

Figure 3-4: Application Subsystem

4. Research

4.1. Quadcopter

4.1.1. Quadcopter Overview

A quadcopter is an unarmed aerial vehicle (UAV) whose lift is generated by four rotors, spaced equally at the corners of a square body. The copter is able to fly by having two pairs of rotors that rotate in opposite directions: one pair turns clockwise and the other counter-clockwise (Figure 4-1). By adjusting the amount of torque and thrust produced by each rotor, the copter is able to move freely in a three dimensional space with six degrees of freedom, three of which are translational (up/down, forward/back, left/right) and three which are rotational (pitch, yaw, and roll).

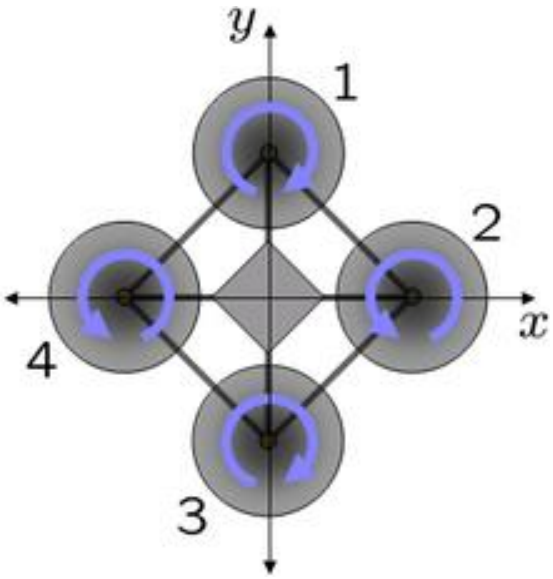

Figure 4-1: Rotational direction of quadcopter blades. Permission from Gabriel Hoffma. See Appendix B for details

Starting out, the biggest decision on the front end of our project is deciding the level of prefabrication. Quadcopter kits can arrive Ready To Fly, All-Inclusive, or Almost Ready To Fly. They can also arrive as a Frame Kit or they can be Scratch Built. Components of the Flight controller can be purchased individually (such as the accelerometer or microcontroller), or they can arrive already connected and ready to plug in. When deciding on how much of the copter we want to come already assembled and ready to be integrated, the primary factors that were considered were cost, customization, and the amount of time it will take to get the copter off the ground.

4.1.2. Camera

SARS uses a camera mounted below a quadcopter to identify the target object on the ground. The camera is angled so that the field of vision is directed straight down at the ground. The camera had to be capable of interfacing with a computer so that image processing algorithms may be run on the individual frames from its video feed. The camera had to produce high enough quality images that a reasonably large, brightly colored object could be clearly identified from an approximate height of 10 ft. The camera had to have stabilization software or a stabilized mount so that the

video feed was clear. SARS had to include video streaming from the camera to the CDC application.

The three cameras that were most suitable for SARS were the GoPro Hero3 White, the Raspberry Pi camera module, and the HTC RE Camera.

4.1.2.1. GoPro Hero3 White Edition

The GoPro Hero3 White Edition was the first camera considered by Group 4 for the SARS image processing subsystem. Listed below are the pros and cons of using this camera. The decision to use the GoPro to implement SARS was based upon but not limited to these factors.

Pros:

- Wi-Fi Communications; may interface with a BeagleBone
- Weather-proof
- 5MP still photos with wide field of view and 2592x1944 screen resolution
- See Table 4-1: GoPro Hero3 White Video Modes below for specs on video resolution
- Rechargeable lithium-ion battery; see Table 4-2: GoPro Hero3 Battery Life below for battery life information
- Many prefabricated quadcopters have mounts compatible with GoPro cameras

Cons:

- Weight (4.8 oz)
- Costs $199.99

Table 4-1 below displays the video modes of the GoPro Hero3 White Edition. More than likely, if the Hero3 is selected as the camera for the SARS image processing subsystem, the 1080p video resolution was used because for the limited scope of the project, a medium sized field of view was all that was necessary, so the SARS Group did not want to sacrifice video quality.

Video Resolution	1080p	960p	720p	WVGA
Frames per Second (fps) NTSC/PAL	30 25	30 25	60 50 30 25	60 50
Field of View (FOV)	Medium	Ultra Wide	Ultra Wide	Ultra Wide
Screen Resolution	1920x1080	1280x960	1280x720	848x480

Table 4-1: GoPro Hero3 White Video Modes

Table 4-2 below indicates the approximate continuous recording time (hr:min) expected when shooting in various video modes using a fully-charged battery based on tests performed by the GoPro engineers.

	With Wi-Fi Off	With Wi-Fi On + Using Wi-Fi Remote	With Wi-Fi Off + Using LCD Touch BacPac™
Video Mode	Estimated Time	Estimated Time	Estimated Time
1080p 30 fps	2:15	2:00	1:30
960p 30 fps	2:45	2:30	1:45
720 60 fps	2:15	2:00	1:30
720p 60 fps	3:00	2:30	1:45

Table 4-2: GoPro Hero3 Battery Life

The open source project, GoProController, written in Python allows SARS to interface between the GoPro camera and the CDC via Wi-Fi communications. This project opens the camera's live stream and saves a single frame using OpenCV. Object detection algorithms are then run on the images saved by the application, enabling SARS to identify the target object.

4.1.2.2. Raspberry Pi Camera Module

The Raspberry Pi Camera Module was the second camera considered by Group 4 for the SARS image processing subsystem. Listed below are the pros and cons of using this camera. The decision not to use the Raspberry Pi to implement SARS was based upon but not limited to the following factors.

Pros:

- 5MP still photos
- 1080p30, 720p60, and VGA90 video modes
- Connects directly to the Raspberry Pi[1] via the CSI port
- Existing API's (MMAL and V4L) for accessing the camera
- Lightweight and small
- Costs $24.99

Cons:

- No prefabricated quadcopter mounts for this camera
- A Raspberry Pi might be a more versatile processor than is necessary for SARS

The MMAL API framework provides an interface to multimedia components such as the Raspberry Pi camera module. By defining components, ports, and buffer headers, the API enables the transfer of data from a component to a client. When a component is created, the input and output ports are exposed, enabling the streaming of data through buffer headers. A buffer header

[1] Raspberry Pi is a trademark of the Raspberry Pi Foundation.

points to the memory location where the transferred data is stored. This API framework allows for the transfer of data from the Raspberry Pi camera module to the Raspberry Pi itself.
The V4L (Video for Linux) API also provides for the transfer of video and audio from a component to a client.

4.1.2.3. HTC RE Camera

The HTC RE Camera was the third camera under consideration for the SARS image processing subsystem. Listed below are the pros and cons of using this camera. The decision not to use the RE Camera to implement SARS will be based upon but not limited to the following factors.

Pros:

- 1080p, 30fps FHD video
- 146 degree super wide angle lens with f2.8 aperture for low light usability
- 820mAh rechargeable battery
- 1hr 50mins of continuous FHD video recording
- Wi-Fi capability

Cons:

- Weight (2.35 ounces)
- Costs $199
- New product; may include bugs
- No prefabricated quadcopter mounts

The HTC RE Camera is a newly released product; therefore, it is quite possible it may contain some significant faults. As it does support Wi-Fi communications, having to write the code from scratch to hack the video feed and extract frames would be an incredibly challenging task. The fact that open source software exists to do exactly this with a GoPro makes the GoPro a more favorable choice. Furthermore, no prefabricated quadcopter mounts exist for this camera, and because of its circular shape, building a homemade mount would be far more challenging for this camera than it would be for the Raspberry Pi Camera module. In light of these facts, the HTC RE Camera was not seriously considered for use in SARS; however, the SARS Group felt that it merited at least some research.

4.1.2.4. Final Camera Decision

After some discussion, the SARS Group decided to use the GoPro Hero 3 White Edition. The availability of quadcopters with prefabricated GoPro mounts was the most compelling factor in the decision of which camera to use. The team was confident that this setup will yield high quality images for object detection. Ironically, the SARS Group still ended up fabricating its own GoPro camera mount out of neoprene and several hook and loop fasteners. This saved a significant amount of money for the development other SARS subsystems and also provided adequately clear images for the image processing. The compatibility of the GoPro with the CDC was also a major convenience as the CDC performed all of the necessary image processing.

4.1.3. Image Processing

SARS uses its own image processing algorithm, written in Python, to detect the object once the CDC has access to the GoPro images; however, during the initial stages of design, the SARS Group intended to use OpenCV to perform the image processing. OpenCV is an open source framework with C++, C, Python and Java interfaces. It supports Linux; therefore, it is compatible with the CDC. The code for the image processing was to be written in Python to keep it consistent with the open source code found which will allow the CDC to access the GoPro via Wi-Fi.

OpenCV allows for two different methods of object detection: Cascade Classification and Latent SVM. The team originally decided to use the Cascade Classification method as Latent SVM only supports C/C++ interfaces whereas Cascade Classification can be performed in Python and Java as well as in C or C++. The Cascade Classification method involves training a classifier or list of classifiers to detect a specific object and then using the classifiers to determine if any region within the image is likely to contain the specified object. If an image region passes all of the classifiers, the object has essentially been detected. For each classifier, there are two types of error, false positives, when an image region passes as containing the object when it actually does not, and false negatives, when an image region containing the object fails a classifier. Each classifier is trained to pass as close to 100% of the true positives as possible, minimizing false negatives. They are not as adept, though, at preventing the false positives. Minimizing the false positives is the point of cascading the classifiers. For any classifier by itself, because the number of objects in an image is typically so small, the total number of passing image regions is roughly equivalent to the number of false positives. The goal is that a future classifier has been trained to catch the false positives admitted by one of the initial classifiers. Figure 4-2: Cascade Classifier, reprinted pending permission from Paul Viola and Michael Jones, illustrates this process.

The technique for training a classifier was derived from the famous AdaBoost algorithm. It involves initially selecting a classifier which identifies the highest number of positive image regions (regions actually containing the object). The next classifier is selected based on its performance with the false positives which passed the first classifier as well as on its performance with the true positives. In this way, all subsequent classifiers are trained to weed out the false positives of previous classifiers while still passing the true positives.

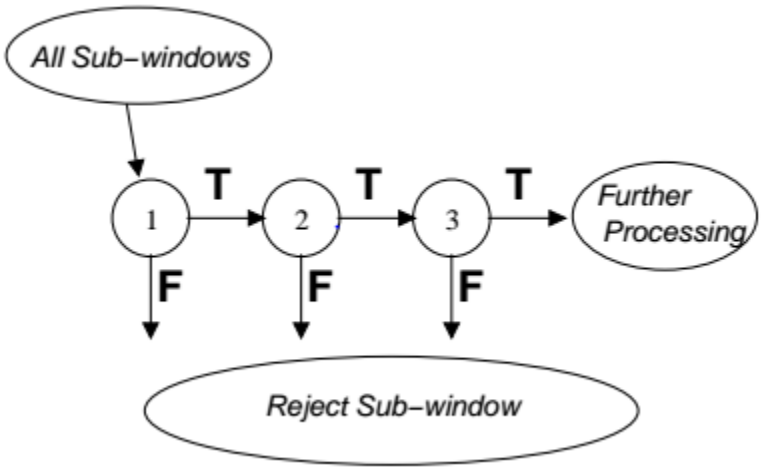

Figure 4-2: Cascade Classifier

Each classifier is composed of multiple features. Features are varying patterns made up of white and black rectangles (see Figure 4-3: Classifier Features – reprinted pending permission from Paul Viola and Michael Jones). The features must be smaller than the sample image being processed. They are dragged across the sample image, and the weighted sum of the region covered by the white rectangle is subtracted from the weighted sum of the region covered by the black rectangle. Based on this difference, the feature passes the sample image region as containing the object or fails it as not containing the object. The cutoff difference is selected based on the desired false positive rate and detection rate.

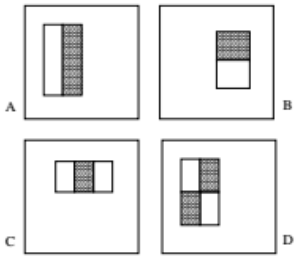

Figure 4-3: Classifier Features

The process of finding the weighted sums is sped along by computing the integral image for the sample image. Each pixel value in the integral image is equal to the corresponding pixel value in the original image plus all of the pixel values to the left and above the corresponding pixel value. This relationship is described by the Equation 1: Integral Image Equations, reprinted pending permission from Paul Viola and Michael Jones, below.

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x', y')$$

$$s(x,y) = s(x, y-1) + i(x,y)$$

$$ii(x,y) = ii(x-1, y) + s(x,y)$$

Equation 1: Integral Image Equations

In these equations, s(x, y) is the cumulative row sum, ii(x, y) is the integral image value at coordinates (x, y), and i(x, y) is the original image value at coordinates (x, y). With the integral image computed, any rectangular sum can be computed with only four array references, drastically expediting the process of calculating the weighted sums.

The process for training the classifiers and selecting the optimal combination for detecting an object involves testing each classifier across two test image databases, one containing the object and one not containing the object. The classifiers are selected based on their performance as described above in relation to a user selected target false positive rate and target detection rate. Once the classifier cascade has been trained, it can then be used on new images to determine with certainty dependent on the established target rates whether or not the images contain the object. OpenCV includes an application called opencv_traincascade which provides the framework for training the classifiers. Once the cascade has been created, the object detection code can be run.

While the SARS Group originally intended to use the Cascade Classifier method of object detection, later in the development stage, the decision was made to switch to OpenCV's SimpleBlobDetector. The SimpleBlobDetector does not require the collection of thousands of sample images to train the application to search for a specific object; it simply identifies blobs of a specific color, size, shape, etc. The SimpleBlobDetector proved to difficult to implement, though, so Matt Bahr wrote an image processing algorithm specifically for identifying a white blob in an image. This algorithm works best in broad daylight when the object is lying on a grass background. The pseudocode for this algorithm is provided later on in this document.

4.1.4. GPS Module

A GPS module needed to be mounted on both the SARS Copter and on the SARS Rover. The two subsystems have slightly different requirements of their respective GPS modules. The SARS Copter module had to be small and lightweight, and it had to consume little power. The SARS Rover module does not have the same size or power limitations as the quadcopter module. The main concern with the GPS for the rover subsystem is that it had to be capable of tracking an object in motion; however, as the rover is not required to move at high speeds, this has not been an issue. Adafruit and Sparkfun both produce GPS units which meet these requirements. The other option considered for SARS was the Ublox LEA-6H module, which was recommended for the Pixhawk flight controller. This was the GPS that was selected for the SARS Copter subsystem.

The Adafruit Ultimate GPS Breakout is a small, lightweight GPS module with a built-in microcontroller. It has a weight of 0.3 oz. and a size of 25.5mmx35mmx6.5mm. The Ultimate

Breakout supports UART communications and is RTC battery-compatible, which means that it will not require a heavy power source. The module's start time is 34 seconds; it has 66 channels for searching for satellites. Having so many channels speeds up the process of locating the satellites. The Ultimate Breakout also has an update frequency of 1 to 10 Hz. Finally, the built-in microcontroller supports datalogging; it stores GPS information in its FLASH memory. The Ultimate GPS Breakout costs $39.95 for orders of fewer than 9.

The Copernicus II is a GPS module produced by Sparkfun. This module is also small and lightweight, having dimensions of 19mmx19mmx2.54mm. It supports UART communications and has a start time of 38 seconds. The Copernicus II only has 12 channels where the Ultimate Breakout has 66. It does not have a built-in microcontroller for datalogging. The official Sparkfun website does not have information on the update frequency of the Copernicus II; however, because the rover is not required to move at high speeds, the capabilities of this module should easily meet the requirements for both the quadcopter and land rover subsystems. The Copernicus II costs $44.95 for orders of fewer than 9.

The Ublox LEA-6H is a module designed by 3D Robotics Inc. It has a 5 Hz update rate. It has a weight of approximately 0.6 oz. and a total size of 38mmx38mmx8.5mm, and it also comes with a protective case, which is a major plus because the quadcopter will be running out in the elements. The main advantage to this GPS unit was that it came included with one of the quadcopter kits that was considered for SARS. It has an APM and Pixhawk-compatible 6-pin DF13 connector which allowed it to interface with the Pixhawk microcontroller. The LEA-6H also includes an LNA and SAW filter to reduce the noise in the received signal. If this microcontroller is selected for SARS, then more than likely the Ublox LEA-6H will be the chosen GPS unit.

Considering the decision to use the Pixhawk microcontroller, the Ublox LEA-6H module was selected as the GPS unit used by the SARS Copter. This unit has all of the functionality necessary for the geolocation of the target object to be retrieved. While it lacks some of the extra functionalities of the other two modules that were considered, these capabilities, independent FLASH memory and data logging, are unnecessary for the basic task being performed. The fact that it comes recommended for use with the Pixhawk used to run ArduCopter was the critical factor in the decision to use this module, and the GPS proved to be incredibly reliable through the development and testing of SARS.

The Adafruit Ultimate GPS Breakout was selected to be used for the GPS navigation in the SARS Rover. This device was chosen over the other two because it is slightly cheaper than the Copernicus II and because it offers several functionalities that the Copernicus II does not, such as the FLASH memory datalogging. Furthermore, it was difficult finding pricing information for purchasing a Ublox LEA-6H module separately from a quadcopter kit. Upon device testing, the Ultimate GPS Breakout proved itself to be more than sufficient for the SARS Rover's navigation to its target retrieval object.

4.1.5. Internal Measurement Unit

The internal measurement unit (IMU) is a device that measures data related to the quadcopter's velocity, orientation, and gravitational forces. The IMU is composed of at least an accelerometer and a gyroscope, but can also contain a compass to measure its relative geographic position as well as a magnetometer. The IMU registers all of this data and adjusts the rotation of the quadcopter's rotors to stabilize it and ensure that it flies correctly.

The accelerometer determines the acceleration of the copter in either meters-per-second-squared (m/s^2) or in G-force (g) which is approximately 9.8 m/s^2. One of the primary purposes of the accelerometer is tilt-sensing, or determining the objects orientation with respect to the earth's surface. The accelerometer is also used to sense motion. The gyroscope measures angular velocity, or how fast the quadcopter is spinning around an access. The gyroscopes measurements are made independent of gravity, so it is able to accurately calculate the copter's rotation in rotations-per-minute (RPMs) or in degrees-per-second (deg/s). These calculations are used to help adjust and correct the pitch, yaw, and roll of our device. A magnetometer is used to orient the copter accurately along its Z-axis with respect to the earth and counteract the copter's drift. A global positioning system (GPS) device can also be used in lieu of a magnetometer. The IMU can also include a barometer to provide more accurate altitude stability and positioning.

For the quadcopter, there are several efficient and easy-to-integrate IMUs that could be implemented in the system. When deciding on an IMU, the first option to consider is whether to buy each of the components separately or buy a senor unit that already integrates each individual sensor. Because our project has so many different parts that need to be interfaced, reducing the more tedious communications within subsystems is a high priority, so we will be buying an all-in-one sensor unit.

In addition, there are several other factors to consider, such as: power consumption, interface (analog, digital, or pulse-width modulation (PWM)), range, axes of reference, and other bonus features such as GPS integration. We centered on three primary options for the IMU: the 9DOF Razor IMU, the DIYDrones ArduIMU V3+, and TI's SensorTag. Each has features that give it a potential advantage over the other. The Razor IMU is Arduino compatible and the outputs of the sensors are processed by an on-board Atmega328 and output over a serial interface. The ArduIMU has a GPS port as well as the on-board microprocessor. Below is a table summary of the various IMU's considered during research.

Gyroscope	ITG-3200 - triple-axis digital-output gyroscope	Tri-Axis angular rate sensor	IMU-3000
Accelerometer	ADXL345 - 13-bit resolution, ±16g, triple-axis accelerometer	Tri-Axis accelerometer	KXTJ9
Magentometer	HMC5883L - triple-axis, digital magnetometer	HMC5883L - triple-axis, digital magnetometer	MAG3110
Voltage Level	3.3-16 VDC	3.3 VDC	3.3 VDC
Dimensions	1.1"" x 1.6"	1.5" x 1.0"	71.2 mm x36 mmx
Serial Interface	I^2C	SPI	I^2C

Table 4-3: IMU comparison table

4.1.6. Flight Controller

For the purposes of our project, we will purchase our flight controller pre-fabricated to minimize the time spent building the quadcopter so we can focus on interfacing the copter with the other components of our project, as well as implementing the image processing from the camera. There are three primary open-source flight controllers available: AeroQuad, ArduCopter, and AutoQuad. Each one has extensive documentation to provide support for building our UAV, as well as nicely packaging the primary microcontroller as well as the IMU and, in some cases, a GPS unit. Below in Figure 4-4 is a comparative table displaying the different functional capabilities of the different flight controllers.

	AeroQuad 32	Arducopter	AutoQuad v6.6
Open Source	Yes	Yes	Yes
Gyro Stabilization	Yes	Yes	Yes
Self-Leveling	Yes	Yes	Yes
Care Free	N/A	Yes	Yes
Altitude Hold	Yes	Yes	Yes
Position Hold	Add-on	Yes	Yes
Return Home	Add-on	Yes	Yes
Waypoint Navigation	Add-on	Yes	Yes

Figure 4-4: Flight Controller Comparison

AeroQuad provides multiple levels of packaging that can simplify the design process depending on the level of customization we are seeking. There is the option to purchase the flight control board on its own, to purchase a kit which includes a shield to allow easy connection to the 9DOF IMU, or a full kit that comes with a pre-determined quadcopter frame, motors, speed controllers, and propellers. Obviously the more that is provided in the kit the more expensive it is, so we will determine what level of customization is necessary to keep our project on track. As far as price, the AeroQaud network is incredibly reasonably priced, with the controller kit coming in at $199.95 and the controller itself at $149.95.

The AeroQuad 32 is run by a 32-bit ARM processor at 168 MHz and has both Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I^2C) compatibility. If connected to the MPU6000, the SPI mode has fast sensor sampling and flight stability. It has 8 PWM receiver inputs and outputs, both 3.3V and 5V outputs, serial wire debug, and 3 USARTs. Overall, it is an incredibly flexible flight controller that would allow a multitude of options for connecting to the IMU as well as our GPS device and wireless communication module.

ArduCopter, much like AeroQuad, has a vast array of information that will make building our copter seamless and painless. Arducopter is powered through the APM 2.6 flight controller which includes a built-in barometer, external I^2C port, and GPS and USB ports. The board is powered at 2.25 A at 5.37 V and can convert power from the main flight battery up to 18 V. Also like the

Aeroquad, it comes with a built-in IMU composed of an MPU6050 6-axis gyro, the HMC5883L 3-axis digital magnetometer and a MS5611-01BA01 Barometric pressure sensor for better altitude control. At $180 ArduCopter's APM is another viable option, though it is less flexible than AeroQuad.

The third primary FCB is produced by AutoQuad. The AutoQuad 6 contains a Ublox LEA-6T precision timing GPS module, 9DOF analog IMU and 1 pressure sensor. It has 14 PWM controllers/receivers, built-in bi-directional telemetry radio compatible with Bluetooth and XBee, and a STM32F407 32bit Cortex M4 microcontroller @ 168Mhz. However, it has an operating input voltage of around 9V, which compared to the other boards is extremely high, and would not be sufficient from a power consumption perspective.

	AeroQuad	Arducopter	AutoQuad
Processor	32-bit ARM	APM 2.6	32-bit Cortex M4
Processor Speed	168 MHz	168 MHz	168 MHz
Flash Memory	32 KB	2 MB	3MB
Serial Interfaces	SPI, I2C	I2C	I2C
UARTs/USARTs	3 USARTs	4 UART ports	1 UART port
Voltage Level	3.3V or 5V	5 V	9V
Price	$199.95	$180.00	$467.00

Table 4-4: Flight controller specifications table

In addition to a flight controller that includes an IMU, we could also choose to buy a MCU on its own and connect a separate IMU ourselves. With this route, there are two popular MCUs that have a tremendous amount of resources and information already documented and available online: the BeagleBone Black and the Arduino Mega 2560.

The ATMega 2560 has 54 digital I/O pins, 16 analog inputs, 4 UARTs, a USB connector and I^2C as well as SPI serial communication capabilities. It has an operating voltage of 5V, as well as 256 KB of flash memory and a clock speed of 16 MHz. Arduino's integrated development environment (IDE) runs in C/C++. The biggest advantage of this MCU is that there is more information and more resources about how to build a quadcopter with this as the primary flight controller than any other board on the market, which would help limit the time spent getting the copter to fly so we can spend more time interfacing all of the different components.

The BeagleBone Black is the second large flight controller MCU used in quadcopters, and though there is some information available it is significantly less than the Arduino board. One of the advantages, though, is that is has native Python support which will make it easier to code, and is compatible with Ubuntu. The BeagleBone Black has 512 MB RAM, UART pins as well as I^2C and SPI compatibility.

4.1.7. Frame

Overall, there are several incredibly important factors to consider when deciding on a frame. First, the level of prefabrication: because of the scale of our project, we are trying to spend only a small amount of time getting the quadcopter to fly because there are so many other components that need to communicate correctly for the project to run. Second, the weight of the frame needs to be able to support the flight controller as well as the BeagleBone and GoPro without being so heavy that it significantly reduces the flight time. Third, the copter needs to have a mount for the GoPro that allows the camera to survey the terrain perpendicular to the ground.

Each of the quadcopter systems has a wide range of possible frames, each with different prices and compatibility specifications that provides a wide range of options. The Aeroquad Cyclone is a relatively cheap frame coming in at $124.95, containing a FCB mount plate that easily supports all AeroQuad boards (as well as standard 45mm output holes). The biggest bonus feature of this frame though is a built-in goPro camera mount, making it simple to attach the camera we will be using for the image processing. Although the frame does not come pre-assembled, AreoQuad's website does have a detailed walkthrough of how to assemble the frame, as well as connect the flight controller and PCB.

An incredibly cheap option is the Spider Quadcopter frame. At only $40.00, this frame provides most of what we would need (GoPro stabilization mount, space to mount 1-4 45mm control boards, as well as a power distribution board as well as an Electronic Speed Control (ESC) unit). In addition, the frame is very light at 486 g, and it will be important to minimize the weight of the copter in order to maximize flight time. Below is a comparison of the two frames.

	Cyclone Frame	Spider Frame
Weight	650 g	486 g
Camera Mount?	Yes	Yes
Frame Material	Aluminum	Glass Fiber
Motors	BP 2217	35-series
Average Flight Time	10 minutes	8 minutes
Price	$124.95	$40.00

Figure 4-5: Frame Comparison

ArduCopter has several frames that come pre-assembled and with many of the boards already assembled. However, most of the frames lack flexibility and although much of the software is open-sourced, copters that are already that put-together will provide less engineering experience.

4.1.8. Processing Microcontroller

Initially, our group planned to run the flight controller, image processing, and wireless communication through a single MCU. However, after realizing the sensitivity of the flight controller and how any interference with its processing can cause the copter to crash very easily, it was decided to attach a second MCU to the copter that would handle the image processing and wireless communication separately while communicating with the flight controller via a serial interface.

Much like the research for the flight controller, the two biggest candidates for the additional MCU were the BeagleBone black and the ATMega 2560. The biggest task that the second MCU would have is image processing, and during our research we were able to find some open source GoPro controller Python code, and because of the BeagleBone's native Python support it was decided that this would be the best environment to handle the image processing. In addition, we were able to receive a free BeagleBone Black from the TI Innovation lab, making our project more cost efficient. Below (Figure 4-6) is a comparison of the two microprocessors.

	BeagleBone Black	Arduino ATMega2560
Operating System	Android, Linux, Windows CE, RISC OS	N/A
Development Environment	Cloud9 and Node.JS	Android IDE, Eclipse
Programming Language	C, C++, Java, Python, Perl, Ruby	Wiring Based (~C++)
Architecture	32-bit	8-bit
Processor	TI Sitara AM3359 ARM Cortex A8	ATMega2560
Speed	1 GHz	16 MHz
RAM	512 MB	8 KB
I/O Protocols	22	14
ADC	Internally Used	6
USB	1x2.0	N/A
Cost	$55.00	$30.00

Figure 4-6: Microcontroller comparison

The BeagleBone (Figure 4-7) has an AM335x 1GHz ARM® Cortex-A8 processor, 512MB of DDR3 RAM, as well as USB clients for power and communications. It also has 2x46 pin out locations.

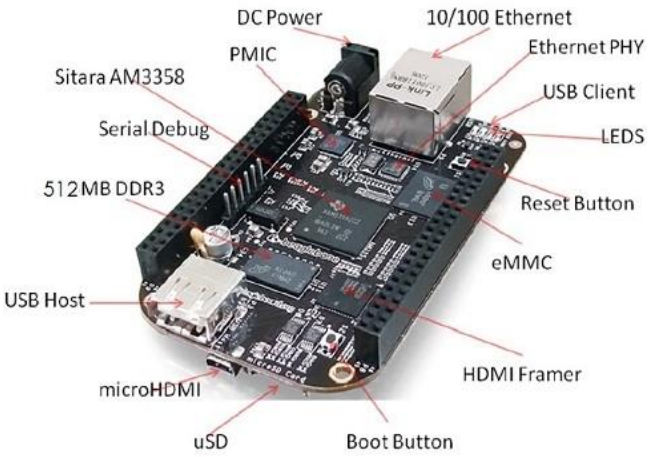

Figure 4-7: BeagleBone Black. Permission from jkridne. See Appendix B for details.

4.1.9. Communication Interface

One of the big questions about the FCB is how all of the different devices will communicate. For quadcopters, there are typically three primary methods of communication: SPI, I^2C, and UART/USART. All three are serial interfaces, meaning they are time division multiplexed so the data is sent over a certain period of time. Each has certain advantages and disadvantages that can be applied to transferring information between the IMU, the flight controller, and the wireless communication module.

SPI (Figure 4-8) is a protocol dictated by a master sending a clock signal to one or two slaves. At each clock signal, the master shifts one bit out to the slaves and receives one bit in, known as MOSI for Master Out Slave In and MISO for Master In Slave Out. The master is able to control which slave to send and receive data to via a Slave Select (SS). This can also be achieved by daisy chaining the slaves together though this makes the software extremely difficult.

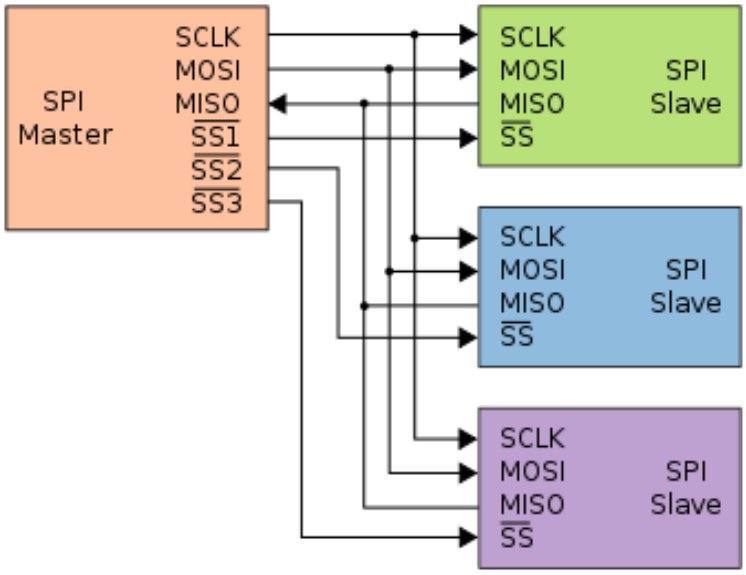

Figure 4-8: Example of Typical SPI Bus. Permission from en:User:CBurnett. See Appendix B for details

I^2C (Figure 4-9), like SPI, is also a synchronous protocol. More advanced than the SPI, this communication interface is comprised of two wires: one that sends a clock signal (SDL) and another which transfers the data (SDL). Unlike SPI which uses SS, the first byte of the data transfer contains a 7-bit address followed by one bit dictating whether the next block of information will be read/write, so the interface is able to communicate with up to 127 different devices. If the master is sending data, it sends out the data bit-by bit with each clock pulse; if it is receiving data, it simply provides the clock pulses and reads in the data off of the data bus.

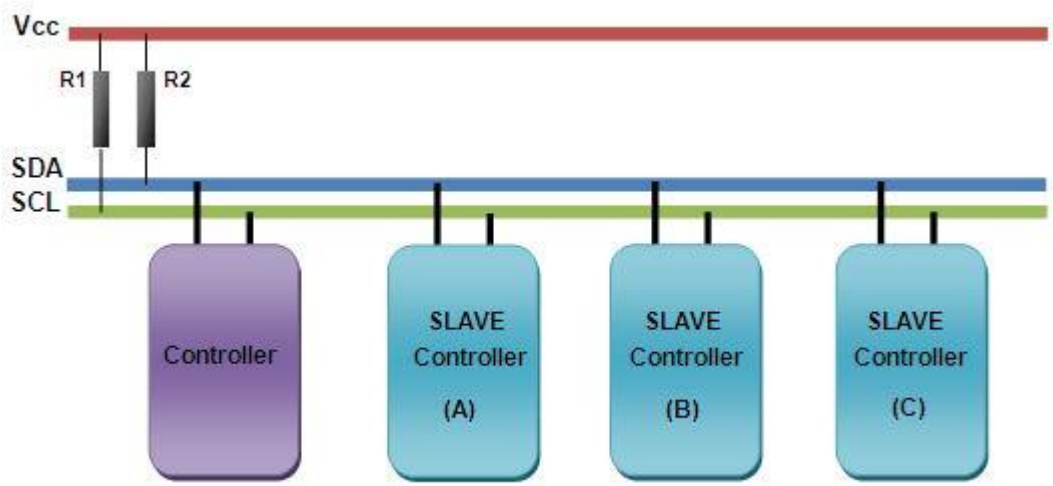

Figure 4-9: I2C communication protocol. Permission from www.engineersgarage.com. See Appendix B for details.

Finally, UART/USART is the third possible serial protocol used to transfer data.8-bit data is transferred typically using a start bit, the single byte of data, and a stop bit. The start bit is a low-

level bit and the stop bit is high-level, meaning that there is no specific voltage level needed so it can be run at 3.3V or 5V, whichever the microcontroller runs at. However, the devices communicating via UART have to agree on transmission speed and bit-rate since it is an asynchronous communication protocol.

4.1.10. Power Distribution

4.1.11. Quadcopter Kits

While it is possible to purchase different parts of the copter separately, there are several cost-efficient kits that include a large level of prefabrication either Ready-To-Fly (RTF) or Almost Ready-To-Fly (ARTF). Ready-To-Fly kits require no assembly and arrive with most of the underlying components shielded from the user to reduce complexity. However, because an additional microcontroller will be connected to modify the flight path, a RTF copter does not provide the level of flexibility or customization necessary to integrate the other subsystems of our project. Thus, the most amount of pre-fabrication we can consider is an ARTF kit.

There are two ARTF kits on the market that have an extensive library of available information and support online. The first is the DIYDrones Quad Kit (Figure 4-10). A do-it-yourself kit, it includes all of the parts needed to assemble a quadcopter with a Pixhawk autopilot system, including motors, propellers, electronic speed controllers (ESCs), a power module and GPS. This kit already comes with all of the part needed to build the copter and get it in the air quickly, with added space to install other parts such as our image processing microcontroller and additional power systems. The kit comes in at a very reasonable $550, which covers the frame and flight controller. The kit is also compatible with ArduCopter, providing access to the largest online open-source multi-copter UAV control platform that will allow us to easily control the copter and send commands to accomplish the goals we need. Below are all of the components of the kit:

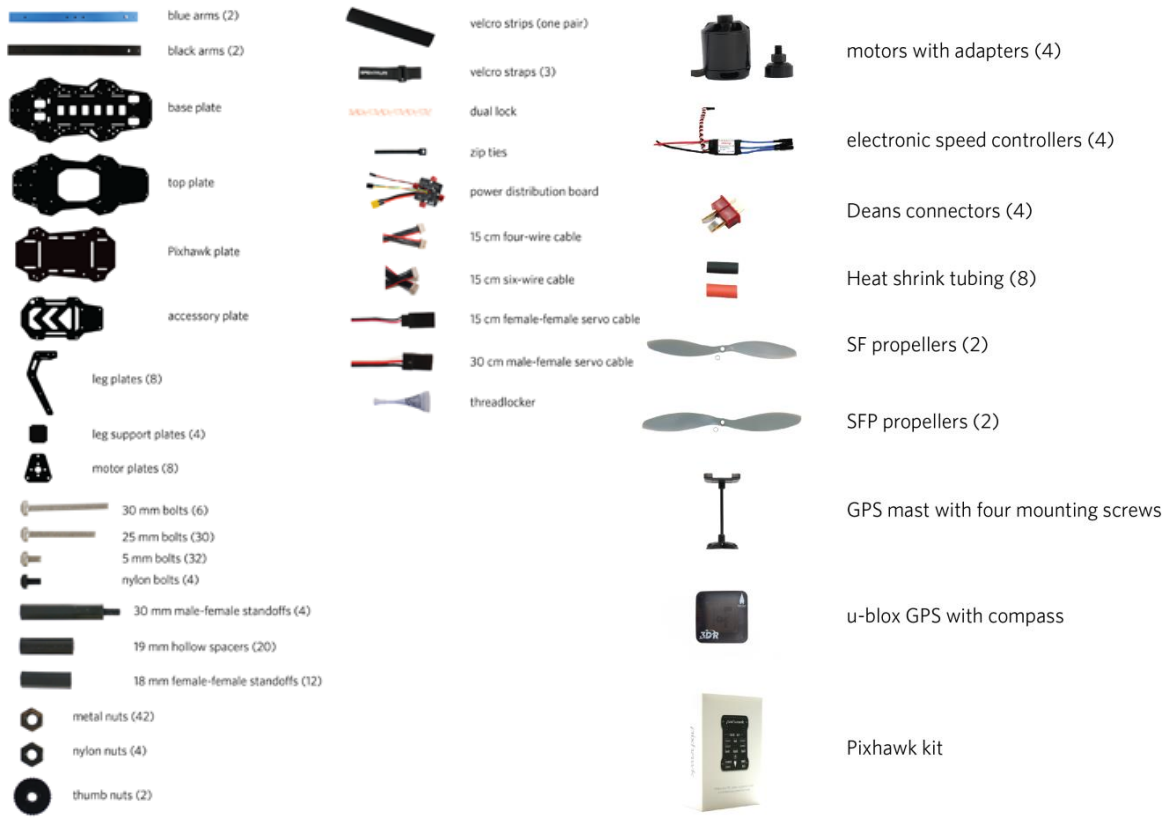

Figure 4-10: Quad kit components

4.2. Rover

4.2.1. Rover Travel Speed

In the selection of motors to power the wheels of the rover, travel speed is in an important consideration. Motors with too much power will lead to the rover being difficult to control, especially with varying terrains. A rover that moves too slowly will make object retrieval take too long and will lead to issues with battery life of both the rover and the quadcopter. Most DC motors are rated in RPM both under load and with no load. In order to determine the linear speed a rover will travel based on the RPM of the motors, some calculations must be made. The only variable other than the RPM of the motor is the radius of the wheels. This is the equation for linear velocity in meters/second based on RPM and wheel radius:

$$v_{linear} = RPM * \frac{1}{2\pi} * R_{wheel}$$

Based on that equation, we can make a comparison of motor speeds by RPM based on a wheel with a radius of 42mm. Table 4-5 shows a comparison of these different RPMs to their corresponding linear speed.

RPM	M/S	Time to Travel 100 M (Seconds)
50	0.334225	299.1993003
75	0.501338	199.4662002
100	0.668451	149.5996502
125	0.835563	119.6797201
150	1.002676	99.73310011
175	1.169789	85.48551438
200	1.336902	74.79982509
225	1.504014	66.48873341
250	1.671127	59.83986007
275	1.83824	54.39987279
300	2.005352	49.86655006
400	2.673803	37.39991254
500	3.342254	29.91993003
600	4.010705	24.93327503
700	4.679155	21.3713786
800	5.347606	18.69995627
900	6.016057	16.62218335
1000	6.684508	14.95996502

Table 4-5: RPM to M/S comparison chart

It was decided that it should not take more than 100 seconds to travel 100m for object retrieval, which eliminates any motors operating below 150 RPM. The upper limit for motor speed is more difficult to determine. It would be beneficial to select a motor with a slightly higher top speed than the speed we need because its speed can be scaled back using the motor controller. Experimentation allowed us to determine the proper travel speed of the rover. It was also beneficial to use on board motion sensing on the rover to determine proper speed on the fly and lower speed if necessary to prevent the rover from possibly flipping over due to moving too quickly over volatile terrain.

4.2.2. Terrain Traversal Capabilities

The rover should be able to travel over most grassy surfaces, concrete, and possibly some dirt. The main point of decision is between wheels and tank style treads. There are advantages and disadvantages to both tank treads and wheels.

Tank style treads are very advantageous on rough terrain. Their overall design was intended to make them extremely versatile in terrain covering capabilities. They are very capable of covering dirt, wet surfaces, and inclines. The other key advantage to treads is that they have a very high total weight capacity, allowing for the rover as well as the object retrieval capacity to be large.

Unfortunately, treads also have down sides. Steering is significantly more difficult with treads than compared to wheels. Also, treads have a much lower speed capability than wheels. The final downside is that if the treads break, repair can be very difficult.

Wheels also have their own set of advantages and disadvantages. Wheels are very cheap. They also can be used for significantly faster speeds than treads. Along with faster speeds, wheels also are much more maneuverable than treads. Finally, any repair issues with wheels can be easily handled by swapping out the wheels. On the downside, wheels have a very difficult time on uneven or slippery terrain. Also, wheels are more susceptible to issues with excessive weight than treads are.

After some consideration, it was determined that wheels will be more beneficial for the SARS project. Speed, maneuverability, and reparability will be more beneficial to SARS than the weight and terrain abilities of the treads.

4.2.3. Motor Controllers

The motor controller is an integral part of the rover system. The first and most important part of motor controller selection is the choice between a 2 wheel drive controller and a 4 wheel drive controller. The first step in that decision is the debate between a 2 wheel drive and a 4 wheel drive rover. A 4 wheel drive system gives the rover the ability to navigate more effectively as well as in place navigation. On the other hand, a 2 wheel drive system is slightly cheaper and uses less battery power. For the SARS rover, a 4 wheel drive system is be more beneficial due to its ability to navigate various terrains as well as the finer turning abilities.

After the selection of a 4 wheel drive system, motor controllers can be investigated. After looking into available motor controllers, it has become apparent that single board to control 4 motors is a rare product. The more popular configuration is to use two 2 channel motor controllers instead of a single 4 channel motor controller. There are some other important features of various motor controllers that must also be considered in the selection process. One feature important to this project is that the controller has stepping capability. This feature allows for the motors to be directly accelerated rather than using pulsing to accelerate. This will allow for more controlled movement as well as smoother motion. Using a pulse style acceleration method will lead to much jerkier acceleration as well as more difficulty decelerating. If the motor controller selected is stepping capable, smooth acceleration and deceleration is possible and motion will be much smother. This is also beneficial for fine movements and prevent unnecessary input to any sensors on the rover. Another key motor control feature is maximum current and voltage throughput. Voltage control is the key to DC motors and therefore the motor controller must allow for at least 10V to achieve maximum performance from the connected motors. Current handling is also important for the motor controller to ensure that the circuitry is not damaged by possible excessive current from the motors. DC motors have no internal current regulation, and therefore have a chance of damaging the circuitry. Ideally, the controller will have a current regulation method, such as a fuse, for protection.

4.2.4. Weight Capacity

For this system, weight capacity was not very high up on the design priority list. The rover is not intended to carry a large amount of weight. In the design specifications it is mentioned that the rover should be able to carry at least 5lbs, which should not have much impact on the overall design. Assuming the rover can handle its own weight, an additional 5lbs will not be difficult to achieve. The only impact the required weight capacity will have on the rover is that it influences the design towards using a metal frame, as opposed to a plastic frame, to maintain structural stability. The metal frame should have no issues withstanding carrying a small amount of weight. The other aspect of weight limitation is that the motors may have difficulty maintaining high RPMs with excessive weight. As mentioned briefly when discussing the overall travel speed of the rover, under load DC motors will have a significantly lower maximum RPM. Therefore, when selecting appropriate motors, the load RPM should be the main deciding factor to ensure effective performance while the rover is carrying weight. There is one important aspect of the weight capacity when determining the design for the rover, which is the amount of torque the rover can handle without tipping over. In order to retrieve the desired object, an arm must be extended from the rover to reach out and grab the object and return it to the rover. This will create a significant amount of torque on the rover and may cause the rover to tip over. In order to counter this issue, counterweight may have to be added to the rover to prevent this, adding to the overall weight of the rover.

4.2.5. Microcontrollers

The microcontroller is perhaps one of the most important aspects of the rover. The microcontroller must be able to interact with the motor controller, object detection sensors, the wireless communication module, and the GPS module. At first glance, it is obvious that the selected microcontroller must have lots of options for communications protocols. It should be compatible with various protocols such as GPIO, UART, SPI, or I^2C. Additionally, the microcontroller selected should operate quickly enough to handle all of the necessary tasks without noticeable performance lag. This controller will be responsible for retrieving current location from the GPS, detecting obstacles with a sensor, determining any changes to the path or speed of the rover (to be communicated to the motor controller), control the object retrieval apparatus, listen for information from other modules (Linux app or quadcopter), and transmit data to the Linux app.

Upon initial research, it was an easy decision to select a microcontroller from Texas Instruments (TI). TI controllers have the widest set of supported standards as well as the largest capability for I/O expansion. TI also offers a large enough product base to meet any of the potential needs of the rover system. The strongest argument for TI controllers is that all of the products are open source, meaning that the schematics are readily available for easy implementation on a printed circuit board. TI offers a range of products from which to choose the most appropriate microcontroller. There are six major product lines from which to select the appropriate microcontroller. For low power and low frequency applications, there are three major divisions of the MSP430 microcontroller. The next product line is the C2000 microcontroller which has the highest frequency and is intended for real time control. The next series for automation and control is based on the ARM Cortex M4 controller and has middle frequency operation. The final product line, the Hercules line, is designed for security applications. At a quick glance, the MSP430 and Hercules

product lines can be eliminated from contention. The MSP430 lines do not have the necessary processing power to control the rover and the Hercules line has unnecessary features whose resources can be better focused elsewhere. The next product to be removed from the running is the C2000 series. The C2000 series is focused on minimal input systems, which is not what the rover is based around. This leaves the automation and control series which has two subseries, the Tiva C and the Concerto. There are a few major differences between these two series, and the most important details to this project have been outlined in Table 4-6.

	Tiva C	Concerto
Processing Core(s)	ARM Cortex M4	ARM Cortex M3, TI C28X (FPU)
Operating Frequency	120MHz	100MHz
Flash Memory	1024KB	512KB
RAM	256KB	64KB
UART	8	5
SSI/SPI	4 (Bi, Quad, Advanced)	4
I^2C	10	2
GPIO	140	64
Average Chip Cost	~$20	~$35

Table 4-6: Tiva C vs. Concerto microcontroller comparison

This comparison makes the controller selection for the SARS Rover simple. The Tiva C series controllers have significantly more I/O capability as well as more storage and a higher operating frequency. The key feature of the Concerto chips is the dedicated floating point unit, which will not be necessary for SARS. The Tiva C series chips are also less expensive, which is always beneficial.

4.2.6. Target Object Retrieval

The SARS Rover's most daunting task is the actual retrieval of the target object once it has reached the object. There were many different approaches to picking up objects with a robotic grasping apparatus that were considered for SARS. The most common options include a two or three pronged clamp or a design that attempts to mimic the human hand. Clamps can be very effective for regularly shaped objects with well-defined edges that can withstand some amount of pressure. As the target object varies in shape and rigidness, however, the clamp approach becomes increasingly ineffective. The other common approach is attempting to mimic the human hand. The human hand is perhaps one of the most versatile object grabbing tools in the world, but it is very difficult to mimic robotically. The human hand has so many degrees of motion that most attempts to copy it do not produce effective results, or they require significant amounts of expensive hardware.

At first glance, a clamp of some sort is the best choice for the SARS system, but after some brief research, a unique concept in robot grasping was discovered. A gripping system developed by the University of Chicago and Cornell University involves using the jamming property of some granular substances. Generally, granular substances act nearly as fluids in a normal environment. When the environment is changed to a vacuum, the particles stiffen in place. This in turn makes

for an extremely effective robotic gripper. This gripper can pick up almost anything as long as the vacuum is being created, and once the vacuum is released, the object will be released as well.

This was originally decided upon as the most beneficial design for the SARS system. This will allow for any desired object, regardless of shape, to be retrieved. Implementation of this design should also have had minimal effects on the rover. Most of the required hardware could have been stored directly on the chassis. The heaviest component of the system is the vacuum pump which could have been housed on the chassis. The actual retrieval apparatus would have consisted of an arm with a balloon filled with coffee grounds (the granular substance) and a tube running along the arm from the vacuum pump to the balloon to create the vacuum and grab the object. The arm would have needed servos to control its motion as well. The arm would have required roughly 90 degrees of up and down motion where it would have connected to the rover and at the end of the arm where the gripper would have been. The hand would have required close to 270 degrees of rotation to adjust for object pickup and then also to allow rotation to drop off the object in the storage bin which would have been mounted on the rover.

Another key factor of the universal gripper is the range of objects which it is able to pick up. Research by John R. Amend, Jr. et al., shows that the gripper can successfully pick up objects 100% of the time that are up to slightly over 75% of the gripper's size. This is illustrated in Figure 4-11. For example, to successfully pick up a tennis ball (official maximum diameter of 6.86cm), the gripper would have to have a diameter of roughly 9.15cm.

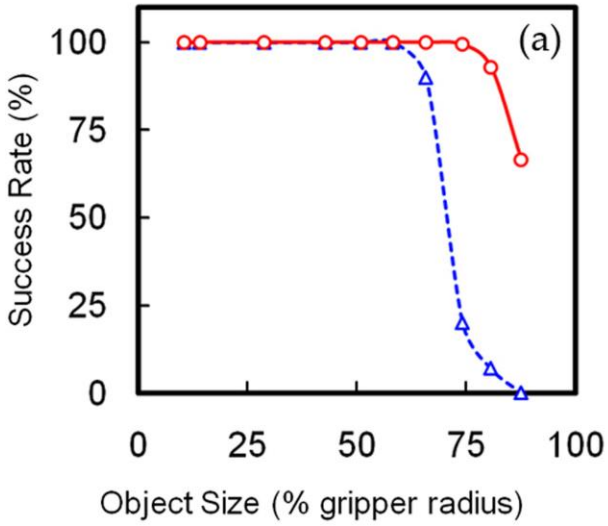

Figure 4-11: Grabber gripping success

Another option that was considered for SARS was to use an electromagnet to lift some metal object. This magnet could have been mounted on the end of a robotic arm and lowered overtop of the target object. If this ended up being the final retrieval method implemented for SARS, the target object would have needed to be modified since a tennis ball cannot be lifted with a magnet. In this case, a large, brightly colored sheet would have been placed on the ground so as to facilitate aerial detection. On top of this sheet a small, three-dimensional metal object would have been placed. This would have been the object for which the rover would have been searching. Upon

detection, the arm would have lowered, the electromagnet would have been turned on, and the object would have been lifted off the ground.

One pivotal concern when designing a robotic arm is the degrees of freedom required. Essentially, each degree of freedom is an arm joint containing a servo motor and an encoder. It is important to try to use as few degrees of freedom as possible because the extra hardware can be very costly. High resolution encoders can cost upwards of $50. Encoders are necessary to implement a PID feedback control system for each arm joint. Using optical sensors to measure servo rotation, the encoder sends electrical pulses to the microcontroller whenever the servo rotates some arbitrary number of degrees. The frequency of these pulses can be used to calculate displacement, velocity, and acceleration, among other variables.

More than likely, if a magnet had been selected as the final retrieval method for SARS, this system would require fewer degrees of freedom than would the Universal Gripper. A moderately powerful electromagnet would not require as much accuracy in its placement overtop of the target object; however, depending on the size of the magnet, its weight could potentially be too much for the arm to lift. Also, a powerful magnet might damage a metal arm if the degrees of freedom of the arm joints are not carefully monitored.

Since the robotic arm would only have been performing one function, it would not have needed extra sensors for finding the target object besides those already mounted on the rover. Essentially, the arm only would have needed to switch between two positions: picking up the object and dropping it in the storage unit. Because the task being performed is so simple, using a robotic arm to perform it probably would have ended up being a waste of time and resources, and due to time and power constrains, this idea was dismissed before any significant design progress was made, and the SARS Group decided to use a hook and loop fastener attached to the end of metal arm extending from the front of the SARS Rover. The arm was designed to lower to grab the object and raise up to lift it off the ground, but because of the power distribution limitations of the SARS Rover motor controller, the arm ended up remaining static, and the SARS Rover would simply drive forward until the fastener attached to the target object.

4.2.7. Non-Contact Obstacle Detection and Avoidance

If the rover encounters an obstacle on route to object retrieval, it must be capable of detecting and avoidance these obstacles before contacting them so that it does not crash. Two sensor technologies have been considered for this purpose: infrared distance measurement sensors and ultrasonic distance measurement sensors. Each respective technology has its own advantages and disadvantages which will be briefly overviewed before discussing specific sensor models that have been considered for use with the SARS rover.

4.2.7.1. Infrared Distance Sensors

Infrared sensors work by triangulating the distance to the object. A pulse of infrared light is emitted from an infrared emitter, it reflects back off of the object, and then it strikes an infrared detector. When the light reaches the detector, it arrives at an angle that is dependent on the distance from the object from which it was reflected. The distance to the object is then determined from this

angle using basic trigonometry, and the output voltage is varied according to this distance. Figure 4-12 illustrates how the infrared sensors work.

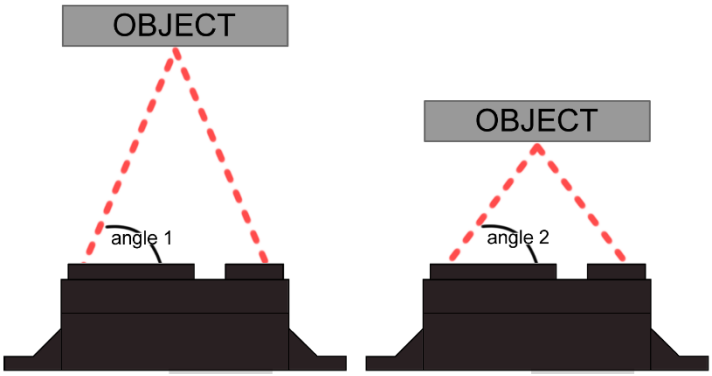

Figure 4-12: Triangulation with Infrared Sensors
Image Courtesy of ROBOTC

Infrared sensors are good for precise detection of objects and are relatively cheap and easy to implement. Precise detection may become important if the rover must perform a close proximity search of the intended retrieval target using the object detection and avoidance system. Infrared sensors also have several very limiting disadvantages. First, they are extremely susceptible to influences from other light sources. SARS is designed to operate in outdoor environments, where sunlight will provide a significant amount of interference for the sensors. If the interference is too great for the infrared sensors to operate effectively, then they will be rendered useless. Second, the output voltage vs. object distance behaves according to a non-linear relationship (see figure 10), which makes accurate calculation of the object distance more complex and less reliable. The combination of these two disadvantages makes infrared based object avoidance a less desirable option than an ultrasonic based solution. Nevertheless, it is still considered in the event that precise objection detection becomes important to the system.

4.2.7.2. Sharp GP2Y0A21YK Proximity Sensor

A popular infrared sensor is the Sharp GP2Y0A21YK. It has an effective range of 10cm to 80cm with an output voltage that ranges from 3.1V to 0.4V. The relationship of target distance to output voltage is illustrated below in Figure 4-13.

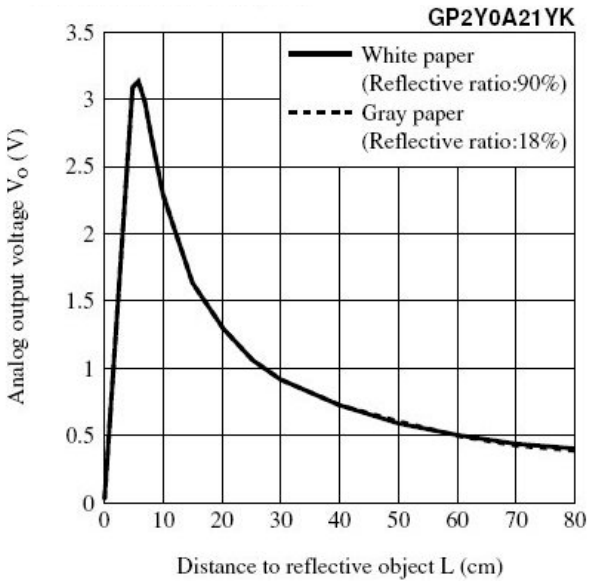

Figure 4-13: Sharp GP2Y0A21YK V-L Relationship
Reprinted with permission from Sharp (Pending)

The sensor outputs are terminated with a 3-pin Japanese Solderless Terminal (JST), which consists of the 5V supply line, ground, and the output line. The sensor does not require an external clock or signal to operate. It continuously pulses its infrared emitter and reports the results via voltage to the output line, which requires around 30mA of current to operate. Table 4-7 summarizes the sensor's specifications.

Sharp GP2Y0A21YK	
Cost	$13.95
Distance Output Type	Analog (Voltage)
Detecting Distance	10cm to 80cm
Supply Voltage	+4.5V to +5.5V
Output Terminal Voltage	+0.4V to +3.1V
Operating Current	30mA to 40mA
Operating Temperature	-10°C to +60°C

Table 4-7: Sharp GP2Y0A21YK Specifications

4.2.7.3. Ultrasonic Distance Sensors

Ultrasonic sensors work using sound instead of light. They operate using a principle similar to radar or sonar, whereby the distance to the target is determined by measuring the time it takes for a sound wave to travel to the target and back to the sensor. Knowing the travel time and the speed of sound in the medium through which the sound wave was traveling, the distance to the target can be calculated. Figure 4-14 illustrates how this process works.

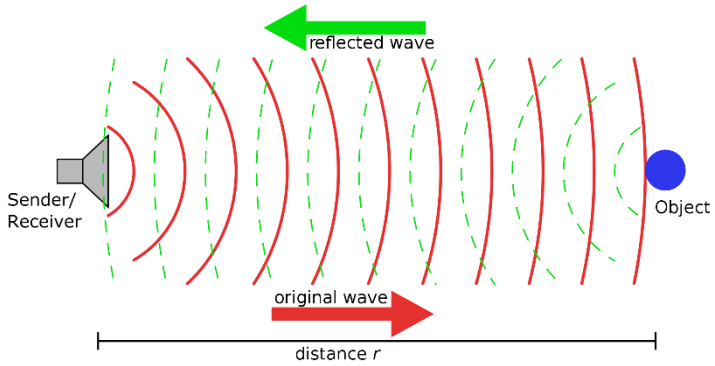

Figure 4-14: Ultrasonic Ranging
Image Courtesy of Wikipedia

Unlike infrared sensors, ultrasonic sensors are not susceptible to influences of sunlight, which makes them very suitable for outdoor applications such as SARS. They also project the sound waves in a wider beam than a typical infrared sensor projects its light, so they are better at sensing objects in a general direction since they do not need to fire the sound wave directly at the target. They do have some disadvantages, though. First, they do not work well when measuring distance to sound absorbing objects such as sponges. If the rover encounters objects like this, it will not be able to detect and avoid them. Second, if the sound waves emitted by the sensor encounter an object or wall with sharp and/or irregular angles on its surface, the sound wave could be reflected at such a sharp angle that it never returns to the sensor and is lost. The sensor would then fail to report the obstacle, and the rover would be unable to avoid them. In addition to "lost" echoes, ultrasonic sensors are susceptible to "ghost" echoes, where sound waves reflected at sharp angles from an irregularly shaped object or wall can return to the sensor as a false positive. The rover would then attempt to avoid an object that isn't actually there. Figure 4-15 illustrates a scenario where this situation could occur. The third disadvantage of ultrasonic sensors is cost. They are considerably more expensive than infrared sensors, costing upwards of $20-$30 per sensor whereas infrared sensors are typically in the range of $10.

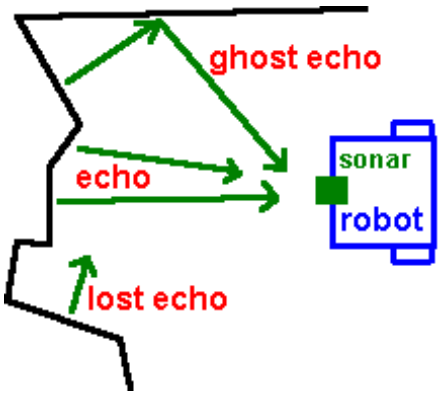

Figure 4-15: "Ghost" and "Lost" Echoes

4.2.7.4. Parallax Ping))) Ultrasonic Distance Sensor

A popular ultrasonic distance sensor is the Ping))) by Parallax. It has an effective range between 2cm and 3m. The Ping))) operates on 3 pins: a 5V supply pin, a ground pin, and a single pulse in/ pulse out IO pin. The sensor is triggered via the IO signal pin and returns the time it takes for the ultrasonic echo to return as a digital pulse via this same pin. An LED indicates when the sensor has triggered an ultrasonic burst and is awaiting return. The sensor requires 30 to 35mA to operate when active. Table 4-8 summarizes the sensor's specifications.

Parallax Ping)))	
Cost	$29.99
Distance Output Type	Digital (TTL Pulse)
Detecting Distance	2cm to 3m
Supply Voltage	+5V
Operating Current	30mA to 35mA
Output Terminal Voltage	+3.3V or +5V
Operating Temperature	0°C to +70°C

Table 4-8: Parallax Ping))) Specifications

4.2.8. Power

Last but not least is the power source for the rover system. Last but not least is the power source for the rover system. There is a large host of options regarding battery selection for the rover system.

The first aspect of choosing an appropriate battery for the rover system is choosing which type of battery the system should use. There are 6 major types of batteries: nickel cadmium, nickel metal hydride, reusable alkaline, lithium-ion, lithium-polymer, and sealed lead-acid. The comparison of these different batteries is in Table 4-9. The information presented helps begin the process of narrowing down which type of battery is ideal for the SARS Rover. A few of the options can be crossed off the list of possibilities fairly quickly.

	NiCd	NiMH	Lead Acid	Li-ion	Li-ion polymer	Reusable Alkaline
Gravimetric Energy Density (Wh/kg)	45-80	60-120	30-50	110-160	100-130	80 (initial)
Internal Resistance (includes peripheral circuits) in mΩ	100 to 200 6V pack	200 to 300 6V pack	<100 12V pack	150 to 250 7.2V pack	200 to 300 7.2V pack	200 to 2000 6V pack
Cycle Life (to 80% of initial capacity)	1500	300 to 500	200 to 300	500 to 1000	300 to 500	50 (to 50%)
Fast Charge Time	1h typical	2-4h	8-16h	2-4h	2-4h	2-3h
Overcharge Tolerance	moderate	low	high	very low	low	moderate
Self-discharge / Month (room temperature)	20%	30%	5%	10%	~10%	0.30%
Cell Voltage(nominal)	1.25V	1.25V	2V	3.6V	3.6V	1.5V
Load Current - peak - best result	 20C 1C	 5C 0.5C or lower	 5C 0.2C	 >2C 1C or lower	 >2C 1C or lower	 0.5C 0.2C or lower
Operating Temperature (discharge only)	-40 to 60°C	-20 to 60°C	-20 to 60°C	-20 to 60°C	0 to 60°C	0 to 65°C
Maintenance Requirement	30 to 60 days	60 to 90 days	3 to 6 months	not req.	not req.	not req.

Table 4-9: Battery Type Comparison (From BatteryUniversity.com)

Reusable alkaline batteries will not even be considered due to their relatively low energy density of 80 Wh/kg, which drops off with every charge. Alkaline batteries will very quickly drop off to roughly 50% of their original capacity after just 50 cycles. This would be detrimental to the development process for the SARS Rover because 50 charge cycles could occur just during the testing process. The next battery to be eliminated is the lead acid battery. While these batteries have many useful features, they are too heavy to be useful for this system. Similar batteries of a different type will have 2-3 times the capacity at the same battery weight. The biggest advantages of the lead acid are the high overcharge tolerance, the low internal resistance, and the low self-discharge rate. These features are not available with any other type of battery, but are not beneficial enough to warrant use of a lead acid battery. The next battery to be eliminated from the running is the NiCd battery. Despite having a phenomenal fast charge time, excellent discharge rates, and great cycle life, these batteries have the same problem as the lead acid of fairly low energy density. They also have a maintenance requirement that can be as frequent as every 30 days, which will cause lots of difficulty. The last of the preliminary eliminations of battery type is the NiMH. These batteries have many favorable features. Energy density, discharge rate, cycle life, and charge time are all respectable values, but there are still some large downsides. These batteries have a whopping 30% self-discharge rate which is a major detriment. They also, like the NiCd batteries, have a maintenance requirement. It is not quite as frequent as the NiCd, but requiring maintenance every 60 to 90 days is still an unnecessary hassle.

At this point, we are down to the last two major battery types and final selection becomes more difficult. Based on the information presented, li-ion and li-po batteries have very few differences between them. Both have similar discharge loads, operating temperatures, and self-discharge rates. Li-ion batteries have a slightly higher energy density and cycle life with slightly less internal resistance, but most of these differences are negligible. There are a few other key details that must be considered in choosing between these two types of battery. The li-po batteries are actually much smaller than the li-ion batteries making them easier to fit wherever they are needed. They are also safer than li-ion because they are not as likely to malfunction due to overcharging. These are key features for selection for the SARS Rover.

Aside from battery selection, the other important aspect of the power system of the Rover system is the gauge of wire that must be used to ensure that current can travel safely without risk of any fires or damaging equipment. Table 4-10 illustrates the current capacity of various gauges of wire. The selected gauge of wire should be roughly 20% higher than the maximum anticipated current through the system to ensure that the safety requirements are met. Preliminary research suggests that the system power will peak at 50A per channel, for 2 channels, so wiring must support roughly 60A.

AWG gauge	Conductor Diameter Inches	Maximum amps for chassis wiring	Maximum amps for power transmission	AWG gauge	Conductor Diameter Inches	Maximum amps for chassis wiring	Maximum amps for power transmission
0000	0.46	380	302	14	0.0641	32	5.9
000	0.4096	328	239	15	0.0571	28	4.7
00	0.3648	283	190	16	0.0508	22	3.7
0	0.3249	245	150	17	0.0453	19	2.9
1	0.2893	211	119	18	0.0403	16	2.3
2	0.2576	181	94	19	0.0359	14	1.8
3	0.2294	158	75	20	0.032	11	1.5
4	0.2043	135	60	21	0.0285	9	1.2
5	0.1819	118	47	22	0.0254	7	0.92
6	0.162	101	37	23	0.0226	4.7	0.729
7	0.1443	89	30	24	0.0201	3.5	0.577
8	0.1285	73	24	25	0.0179	2.7	0.457
9	0.1144	64	19	26	0.0159	2.2	0.361
10	0.1019	55	15	27	0.0142	1.7	0.288
11	0.0907	47	12	28	0.0126	1.4	0.226
12	0.0808	41	9.3	29	0.0113	1.2	0.182
13	0.072	35	7.4	30	0.01	0.86	0.142

Table 4-10: Wire Gauge Current Capacity Comparison

4.3. Android Development

SARS was initially designed with to be used with an Android application. It was supposed to serve as the communications hub through which the quadcopter and rover reported their current status and system diagnostics. The application was also going to be responsible for sending the commands to both begin a search and terminate a search early. It was also going to provide a live video stream of the camera mounted to the quadcopter so that users had a real time view of the SARS system as it performs its search and retrieve operation. The following Android hardware and software were initially planned for use, but the final system utilized a Linux based application instead.

4.3.1. Hardware

The Android application would be initially designed for use with a Nexus 5 smart phone which is part of Google's Nexus family of Android devices. The Nexus devices are designed, developed, marketed, and supported by Google but manufactured by original equipment manufacturers (OEMS) such as Samsung and LG. The Nexus devices are considered Google's flagship Android devices, and are specifically designed by Google for use as Android software development devices.

Because they are designed specifically to run on the latest, purest (un-customized) version of Android, they typically are the best suited devices for prototyping new applications. If development proved successful on the Nexus 5, and ample time was allotted after project completion, the team would decide to also support SARS on tablets. In this case, a Nexus 10 tablet would have been used for development Table 4-11 summarizes the specifications for the Nexus 5.

Google Nexus 5	
Cost	$399.00
Screen	4.95" 1920x1080 Full HD IPS
Size	69.17mm x 137.84mm x 8.59mm
Weight	4.59oz (130g)
Memory	32GB NAND Flash 2GB RAM
Processor	CPU: Qualcomm Snapdragon™ 800 @ 2.26GHz GPU: Adreno 330 @ 450MHz
Sensors	GPS, Gyroscope, Accelerometer, Compass, Proximity, Ambient Light, Pressure, Hall Effect
Wireless Communications	2G/3G/4G GSM/CDMA/WCDMA/LTE 802.11 a/b/g/n/ac Dual-Band Wi-Fi Bluetooth 4.0 LE NFC
Battery	2300mAh LiON
Operating System	Android 5.0 (Lollipop)

Table 4-11: Google Nexus 5 Specifications

4.3.2. Software

The Android application would have been designed with the Android Software Development Kit (SDK). The Android SDK is a comprehensive set of development tools that provides the API libraries, debuggers, emulators and other developer tools necessary to build, test, and debug applications for the Android platform. The most recent version of the Android SDK is API version 21, which only supports Android version 5.0 "Lollipop." The target API of the Android SDK that an application is built with is developer configurable. Targeting lower versions of the Android SDK API allows a wider range of Android versions, and thus a larger number of user devices, to be supported. However, lower SDK API versions do not have support for any of the features implemented in newer versions of the API. For example, designing an application with SDK API version 14 allows a developer to support Android devices running version 4.0 "Ice Cream Sandwich" and up, which represents approximately 87.9% of all Android devices active on Google Play Services. Figure 4-16 illustrates the different API versions, and the cumulative percentage of supported devices for that version. As SARS would be a prototype application with very specific purposes, the team would have developed with API version 19. API version 19 supports Android versions 4.4 and 5.0. All members of the SARS team have Android devices running at least Android 4.4 or higher, so initial development would be targeted only for these Android versions. As with device support, if software development with API 19 proved successful, and ample time

was allotted after project completion, the team would have decided to lower the target API down to version 14, which will support all devices running Android version 4.0 or higher.

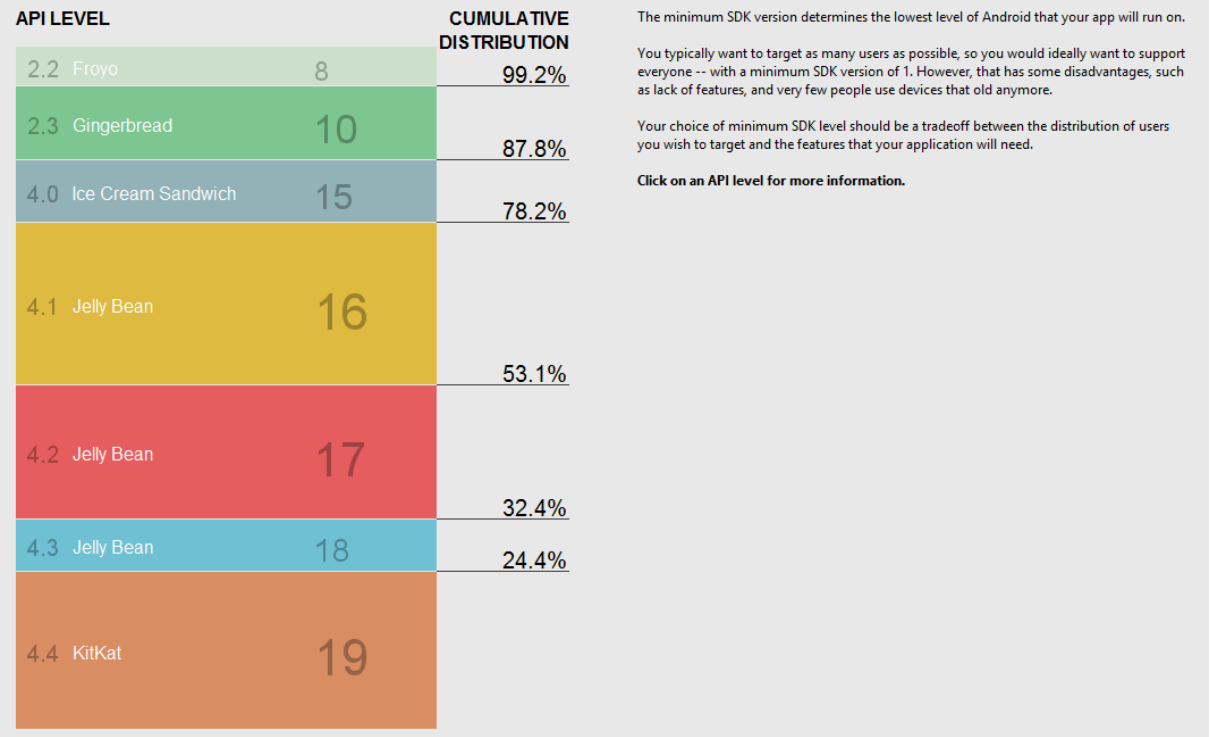

Figure 4-16: Android SDK API Level Distribution

4.3.3. Development Environment

The Android application would have been developed using the recently released Android Studio, which includes the Android Software Development Kit. Android Studio is an integrated development environment produced by Google but based off of the open source Java IDE by JetBrains named IntelliJ IDEA. Android Studio is designed to replace the current official Android development environment – Eclipse with the Android Development Tools plugin – and will become the official Android development environment when it has finished beta testing. Android Studio would have been the preferred development environment for the SARS team for several reasons. First, it is a more robust and easy to use development environment than Eclipse with ADT. Second, it has excellent tools for visualizing and designing the graphical user interface of the Android application. This was important to the SARS team as the application was intended to be as simple and easy to use as possible. Finally, two of the four members of the team have previously developed Android applications using Android Studio, so they are already familiar with the environment and are capable of developing an Android application with it.

4.4. Linux Development

As stated previously, SARS was initially designed for use with an Android application that would serve as our ground station. Due to some technical limitations on the processing power of the microcontrollers intended for use on the quadcopter, as well as additional unforeseen hardware and software requirements, the team decided early on in system development to switch from an Android application running on a smartphone, to a Linux application running on a laptop. The following additional research was performed to facilitate the development of the Linux application which was later dubbed the Command Distribution Center, or CDC.

4.4.1. Hardware

The CDC was designed for use on any laptop capable of running an Ubuntu/Debian based distribution of Linux. Team members Erick Makris and Matthew Bahr were primarily responsible for the development of the CDC, so their personal laptops served as development environments. The use of a Linux laptop was required for development as the CDC needed to be able to interface with additional hardware connected via USB, which a microcontroller on board the quadcopter would not be able to provide. The decision to use a laptop was also influenced by concerns of inadequate processing power for image processing if a Linux powered microcontroller such as the BeagleBone Black was used.

4.4.2. Software

The CDC needed to be developed in Ubuntu Linux, preferably with the Python scripting language because several of the open source libraries used by the system are written in Python and depend on Linux based libraries. As Python is primarily a scripting language, the SARS team had to decide on a method of implementing a GUI for users to be able to easily interact with the application. After some initial research, three GUI toolkits were considered for use: PyQt, PyGTK, and wxPython. All three toolkits are Python implementations of the corresponding GUI frameworks Qt, GTK+, and wxWidgets. All three toolkits have a similar set of features, and the only major difference between them is the amount of available documentation and community support. However, Qt is by far the most widely used and well documented toolkits of the three. These two key factors are what pushed the SARS team to use the Qt/PyQt solution for GUI development.

4.4.3. Development Tools & Environment

As stated before, the CDC was designed using Qt, PyQt, Python, and several open source Python libraries. No Integrated Development Environment was required or used for application development. Code was primarily written in the Vim terminal text editor, and tested on the Linux terminal. A terminal sharing program called Tmux (terminal multiplexer) was also utilized so that SARS developers could utilize pair programming tactics to simultaneously edit and review code together. A screen capture of a typical CDC development environment session appears below in

Figure 4-17 for reference. This same screen is visible and editable by all users currently participating in the session.

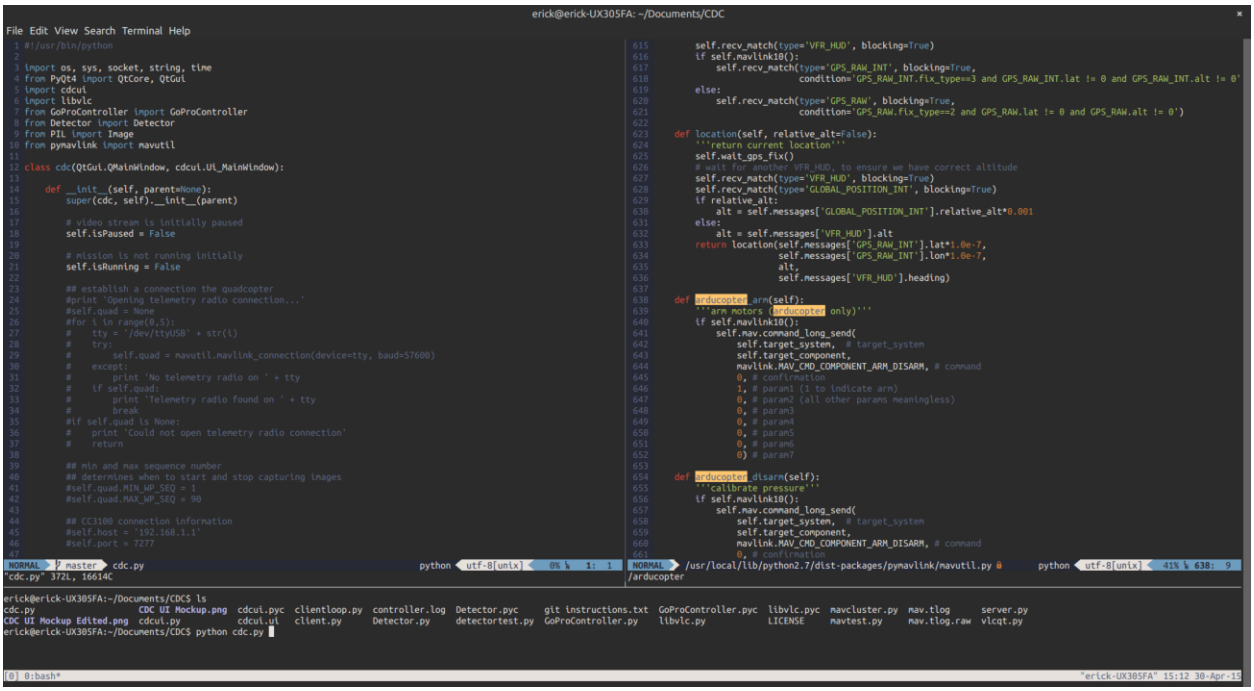

Figure 4-17: CDC Development Environment

The Qt user interface was designed using Qt Creator. Qt Creator is a tool that is provided as part of the Qt development package that allows user to visually design interfaces in a manner similar to other Integrated Development Environments such as Microsoft Visual Studio or Android Studio. The drag and drop interface allows users to place QObjects such as text edit boxes, panels, text input boxes, etc. and automatically generates the XML which can then be interpreted by Qt itself to generate corresponding C++ code or other code generators such as PyQt, which we use to generate the corresponding Python code.

To keep track of additions, removals, and changes of code, a revision control system would need to be utilized for development. The SARS team decided to use the Git revision control system to track code revisions. The Git repository was hosted on GitHub, which is a web-based Git repository hosting service which offers all of the revision control and source code management functionality of Git in addition to other features.

4.5. Wireless Communication

Two different types of wireless communication networks were proposed for use in SARS. The first system that was initially designed to utilize Xbees connected to microcontrollers on both the SARS Copter and SARS Rover which would in turn connect the Android device one at a time. However, as the design of the system evolved and an Android device and SARS copter microcontroller were no longer going to be utilized, the network architecture was also evolved into the second design which utilized the CC3100 and a Linux laptop instead.

4.5.1. Xbee

One of the wireless communication technologies that had been considered is Digi International's Xbee wireless solution. The Xbees are a set of various RF modules that are designed to work with a variety of communication standards such as the IEEE 802.11b/g/n Wi-Fi standard, the IEEE 802.15.4 standard, the Zigbee standard (which extends 802.15.4 to higher protocol layers), and Digi's own proprietary RF communication standard. Xbee's are designed for simplicity and ease of use while still providing low latency, reasonable data transmission rates, and predictable communication timing. Another advantage of the Xbees is that they are also fairly cheap. The Wifi modules cost $35 each, the 802.15.4 modules cost $19 each, and the Zigbee modules cost $17.50 each. As the Wifi module is the version that SARS will most likely use, its specifications have been summarized below in Table 4-12. Since communication with an Android device over Wi-Fi will be required, two configuration scenarios have been considered.

Digi Xbee WiFi S6B	
Cost	$45.00
Serial Communications Interface	UART up to 1Mbps SPI up to 6Mbps
Digital I/O Lines	10
Wireless Communications Standard	802.11 b/g/n WiFi
Wireless Communications Data Rate	1Mbps to 72Mbps
Wireless Network Security	WPA-PSK, WPA2-PSK, and WEP
Wireless Network Channels	13
Transmit Power	+16dBm
Receive Sensitivity	-93dBm to -71dBm
Supply Voltage	+3.14V to +3.46V
Transmit Current	Up to 309mA
Receive Current	100mA
Operating Temperature	-30°C to +85°C

Table 4-12: Digi Xbee Wifi S6B Specifications

The first scenario is to use two Xbee Wi-Fi S6B modules, one for the quadcopter and one for the rover, in an ad-hoc configuration so that each Xbee module can communicate with the other and/or the Android device concurrently. The topology of this configuration is illustrated in Figure 4-18. With this configuration, communication between devices is simplified because all devices can communicate with each other using the same wireless communication standard with no need for conversion between protocols. This is also the cheaper solution of the two, as all that is needed is two Xbee Wi-Fi S6B modules. However, the 802.11 radio requires a significant amount of power (309mA for transmission, 100mA for receipt) which could adversely affect the battery life of the quadcopter and rover.

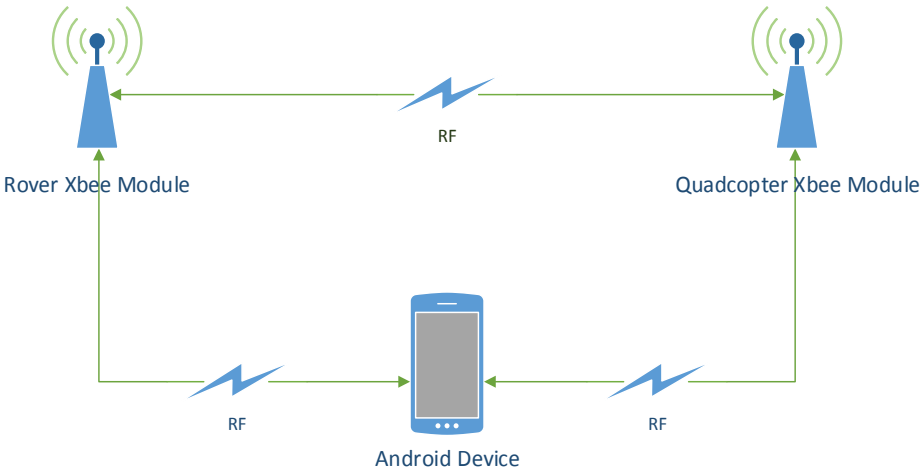

Figure 4-18: Ad-Hoc Topology

The second scenario is to use either two Xbee 802.15.4 or Zigbee modules connected to a Digi ConnectPort X2 gateway which in turn connects to a wireless router. This gateway/router system allows Xbee modules communicating with the 802.15.4 or Zigbee standard to interface with devices communicating over the 802.11b/g/n standard via an HTTP/HTTPS web interface. The Android device could then communicate with the Xbee 802.15.4 modules through an application which integrates this web interface. The topology of this configuration is illustrated below in Figure 4-19.

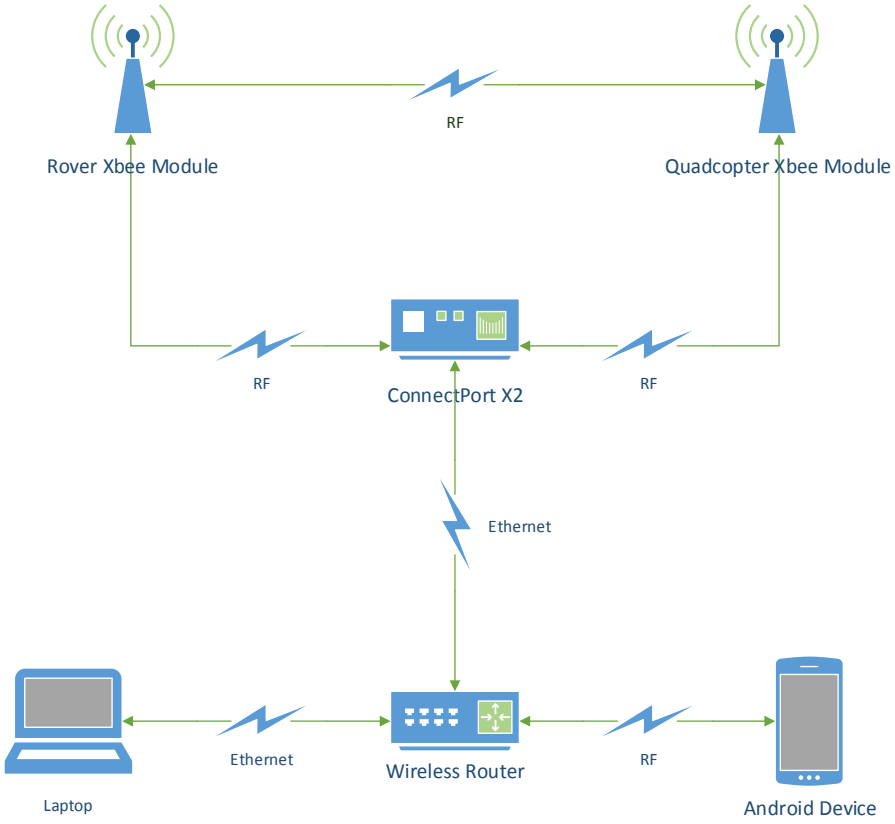

Figure 4-19: Gateway Topology

The advantage of this configuration is that certain versions of the 802.15.4 and Zigbee modules can transmit and receive at much greater range than 802.11. For example, the Xbee-PRO S2 802.15.4 modules can transmit and receive up to 1 mile, whereas the Xbee WiFi modules are limited to around 300 feet. By supporting wireless transmission over long distance in the initial design, SARS could potentially be scaled up later in the design process without concern over wireless transmission range. Integrating the gateway also allows for wider device support. While the initial design will be limited to communication with an Android device, support can easily be extended to any device that is capable of wired or wireless networking. However, this configuration requires both the ConnectPort X2 gateway and a wireless 802.11n router. The ConnectPort X2 is priced at $149.99, and an 802.11n wireless router can cost anywhere from $50-$150 depending on the model. The ConnectPort X2 gateway's specifications are listed below in Table 4-13 for reference.

Digi ConnectPort X2	
Cost	$135.00
Device Management Interface	HTTP/HTTPS Web Interface Device Cloud Management Interface
Protocols	TCP, UDP, DHCP, SNMPv1
Wireless Interfaces	802.15.4, ZigBee, DigiMesh 2.4, 900HP, XSC, 868
Ethernet Interfaces	ZigBee, 802.15.4, DigiMesh (XBee-PRO 900HP)
Supply Voltage	+12V
Operating Current	100mA to 283mA
Operating Temperature	-30°C to +70°C

Table 4-13: Digi ConnectPort X2 Specifications

4.5.2. SimpleLink

Another wireless communication technology that was considered is Texas Instrument's SimpleLink WiFi solution. The TI SimpleLink CC3100 is a self-contained processor that simplifies the implementation of WiFi network connectivity for low-cost, low-power microprocessors such as the TI MSP430 and TI Tiva C Series (both microprocessors that were under consideration for use). The CC3100 consists of a wireless network processor that contains an 802.11b/g/n radio, an embedded IPv4 TCP/IP stack, and a power supply. It is designed to simplify networking by offloading all network-related processing, transmission, and receipt from the host microcontroller thereby reducing the processing and power requirements of the host. Communication between the CC3100 and the host microcontroller is handled via UART or SPI. The CC3100 is slightly more expensive than the Xbee WiFi Module, retailing for $36.99 each as part of a combo pack that is required to enable PC connectivity for programming. Like the Xbee, the CC3100 can be configured for an ad-hoc network, so that the CC3100's on both the quadcopter and rover can communicate directly with each other and also with the Linux application. Table 4-14 summarizes the CC3100's specifications for reference.

TI SimpleLink CC3100	
Cost	$45.00
Serial Communications Interface	UART, SPI
Wireless Communications Standard	IEEE 802.11 b/g/n WiFi
Wireless Communications Data Rate	TCP: Up to 12Mbps UDP: Up to 16Mbps
Wireless Network Security	WPA2-PSK Personal and Enterprise
Wireless Network Channels	13
Transmit Power	+18.0dBm @ 1 DSSS +14.5dBm @ 54 OFDM
Receive Sensitivity	-95.7dBm @ 1 DSSS -74.0dBm @ 54 OFDM
Supply Voltage	Wide Voltage Mode: +2.1V to +3.3V Preregulated Mode: +1.85V
Transmit Current	223mA
Receive Current	53mA
Operating Temperature	-40°C to +85°C

Table 4-14: TI SimpleLink CC3100 Specifications

Another advantage of using the CC3100 is that it is already supported by Texas Instrument's Energia IDE. TI provides the libraries and sample code required to get the CC3100 connected and communicating to either a Tiva or MSP430. This means that code for communicating between our Linux application and the CC3100 could be rapidly prototyped and deployed without having to write any low level libraries for configuring and connecting the network processor.

Implementing the CC3100 as part of the SARS wireless communication network would result in a network architecture that is quite a bit different than the Xbee implementation. To enable the CDC to communicate with both the SARS Copter and SARS Rover, additional wireless communication hardware would need to be utilized. The SARS wireless communication network would consist of 3 dedicated connections, each utilizing its own wireless communication hardware interface. The three connections would consist of a radio connection to the SARS Copter, a Wi-Fi connection to the SARS Rover, and another Wi-Fi connection to the GoPro camera. Figure 4-20 below diagrams how the SARS wireless communication network would look.

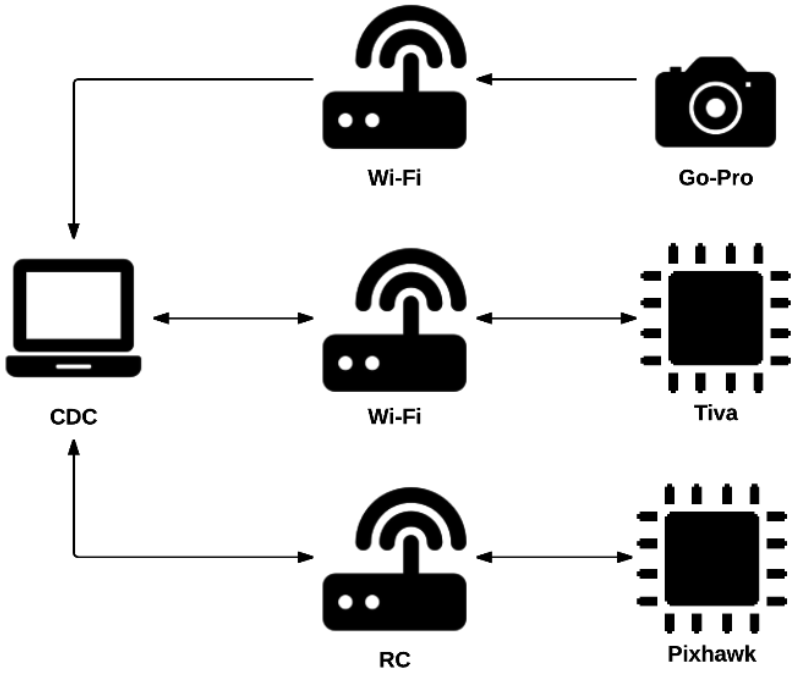

Figure 4-20: SARS Wi-Fi/Radio Communication Network

A Wi-Fi connection would need to be established between the SARS Copter camera and the CDC in this configuration. This connection utilizes an external USB Wi-Fi adapter and the internal Wi-Fi module of the GoPro to provide both live video streaming and image capturing. This secondary Wi-Fi adapter is an optional, but beneficial hardware component that was already owned by a member of the SARS team. The adapter is a TP-LINK TL-WN722N. The specifications for this adapter appear below in Table 4-15.

TP-LINK TL-WN722N	
Cost	$15.00 (already owned)
Interface	USB 2.0
Antenna Type	Detachable Omni Directional (RP-SMA) (4dBi gain)
Wireless Communications Standard	IEEE 802.11 b/g/n WiFi
Wireless Communications Data Rate	802.11n: Up to 150Mbps 802.11g: Up to 54Mbps 802.11b: Up to 11Mbps
Wireless Network Security	64/128-bit WEP, WPA-PSK/WPA2-PSK
Wireless Frequency	2.400-2.4835GHz
Modulation Technology	DBPSK, DQPSK, CCK, OFDM, 16-QAM, 64-QAM
Receive Power	-68dBm
Transmit Power	< 20dBm

Table 4-15: TP-LINK TL-WN722N Specifications

4.6. Video Streaming

In order to provide users with a real time view of SARS as it performs its search and retrieve operation, the Linux application will stream live video from the Go-Pro camera mounted to the quadcopter. The Go-Pro camera saves its live video stream using the HTTP Live Streaming (HLS) protocol, which breaks the live stream into MPEG2 transport stream (.ts) files that are indexed in a UTF-8 M3U playlist (.m3u8) file. This playlist file can be accessed from the Go-Pro's internal file system via a simple HTTP server that is running on the camera and then opened on the Linux application. In the Linux application, the integrated VLC video player natively supports M3U playlists, and will begin playing the live stream from the Go-Pro. A diagram of the HLS protocol shown in Figure 4-21 illustrates how this process works. Therefore, all that is needed for the Linux application to access the Go-Pro's live video feed is to have the integrated VLC player navigate to the Go-Pro's HTTP server and open the M3U playlist.

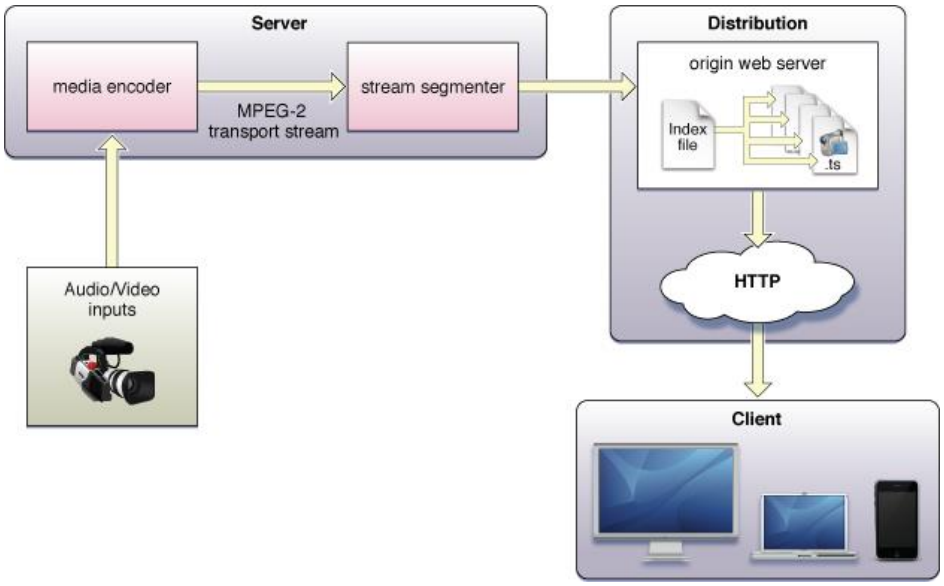

Figure 4-21: HLS Protocol
Reprinted with permission from Apple (Pending)

5. Realistic Design Constraints

Economic: The primary economic constraint of this project was the severely limited budget that the SARS Group had to work with. Boeing sponsored SARS for $1000, and SoarTech provided a $500 sponsorship. Given the scope of the project, however, a budget of $2500 probably would have been more appropriate. In fact, SARS Group members ended up having to spend nearly an extra $1000 out of pocket to buy replacement parts and things that were not originally accounted for in the budget, such as the PPM encoder or the Turnigy RC controller.

Environmental: SARS is a system designed for outdoor use. Both the SARS Copter and the SARS Rover use GPS to navigate to their respective waypoints; therefore, the system is not suitable for

use indoors. Additionally, laws and regulations prohibit the flight of unauthorized vehicles over certain areas, such as no-fly zones. While all of the testing of SARS was performed at an altitude no greater than 15 ft. a scaled up version of the system, in which the SARS Copter hovered at several hundred feet, would not be allowed to run missions in these designated areas. Furthermore, quadcopters of the size chosen for SARS are not designed to fly in inclement weather, so missions cannot be run in heavy wind or rain.

Social: No obvious social constraints apply to SARS.

Political: SARS has been designed at built at an interesting time when not many laws or regulations apply to quadcopters and other drones since this is such a cutting edge technology. Time will tell what laws are passed with regard to UAVs, but obviously relevant regulation could have effects on the use of SARS.

Health and Safety: UAVs can present significant risk to innocent bystanders. Without proper safety controls, there may be nothing preventing a quadcopter from flying into a person or out into the middle of the road. In fact, several of the SARS Group members incurred injuries during the system testing as a result of poor control of the SARS Copter. This system, in its current state, is not safe to run autonomous missions unsupervised. Because of this all testing and even the final demonstration were performed in a strictly controlled environment.

Manufacturability: SARS was not designed with manufacturability in mind. As it is, the system cost far too much too implement to mass produce; however, much of that cost was due to the initial lack of knowledge on the part of the SARS Group. It is possible with the knowledge gained since the onset of the project that future prototypes could be produced for a far lower cost.

Sustainability: SARS was not designed with sustainability in mind.

Legality: As was mentioned previously, not many laws currently exist in regard to the recreational or commercial use of drone technology. Time will tell how this will change; however, there are areas such as no-fly zones where the use of SARS might be prohibited.

Inspectability: All components of SARS are fairly easy to inspect and maintain as long as they are being used properly. Periodic checks to make sure that all nuts and bolts are tightened are necessary; furthermore, the actual performance of the system components needs to be monitored to make sure the SARS Copter flight controller or the SARS Rover microcontroller needs to be replaced. Bug fixes are to be expected for the CDC.

6. Standards

As is true with any design project, a handful of standards applied during the production of SARS. The key standards are briefly introduced in this section and include information regarding which hardware components adhere to them.

Standard	Applicable Components
802.11b/g/n WiFi	CC3100, GoPro Hero3, TL-WN722N
RoHS	MPU-6000, HMC5883L, Tiva C EK-TM4C1294XL, PING))), Ublox GPS
Green	MPU-6000
I^2C	HMC5883L, MPU-6000, Tiva C, Ublox GPS
TI BoosterPack	Tiva C, CC3100
FCC E911	Adafruit Ultimate GPS Breakout
OMA SUPL	Ublox GPS
USB	Tiva C, Pixhawk, Telemetry Radios, WiFi adapter

Table 6-1: Standards associated with SARS

7. Hardware Design

7.1. Quadcopter

7.1.1. Flight Controller

The flight controller that was included in the kit we purchased was a Pixhawk autopilot module that can be controlled from the Mission Planner flight control software. This device was mounted to the copter and was planned to be connected to both the BeagleBone Black microprocessor (which was going to sned interrupts once the object being searched for is detected) as well as the u-blox GPS (for autonomous flight). The Pixhawk is composed of an L3GD20 3-axis 16-bit gyroscope, an LSM303D 3-axis 14-bit accelerometer magnetometer, an Invensense MPU 6000 3-axis accelerometer/gyroscope, and an MEAS MS5611 barometer. Listed below are the remaining specifications of the Pixhawk.

Interfaces:

- 5x UART (serial ports), one high-power capable, 2x with HW flow control
- 2x CAN
- Spektrum DSM / DSM2 / DSM-X® Satellite compatible input up to DX8 (DX9 and above not supported)
- Futaba S.BUS® compatible input and output
- PPM sum signal
- RSSI (PWM or voltage) input
- I2C®
- SPI
- 3.3 and 6.6V ADC inputs
- External microUSB port

Power System:

- Ideal diode controller with automatic failover
- Servo rail high-power (7 V) and high-current ready
- All peripheral outputs over-current protected, all inputs ESD protected

Weight and Dimensions:

- Weight: 38g (1.31oz)
- Width: 50mm (1.96")
- Thickness: 15.5mm (.613")
- Length: 81.5mm (3.21")

The flight controller was mounted in the center of the quadcopter so as to provide and balance to the UAV while it flies. It was attached using vibration damping foam. The foam was used to limit the vibrations felt by the accelerometer and gyroscope which are highly sensitive, and whose readings can be thrown off quite easily. Gel pads also needed to be attached to the board in order to isolate it from the rest of the frame and suspend it about ½ inch high to further reduce vibrations, but after testing it was decided that they were unnecessary and that the vibration damping foam was enough to reduce the noise felt by the Pixhawk's IMU.

7.1.2. GPS

The u-blox GPS (summarized below) was easily connected to the Pixhawk device. There is a DF13 6-pin connector that can be plugged into the GPS port on the Pixhawk, and a 4-pin DF13 connector that attaches to the I^2C splitter module. The GPS module had to be mounted separately from the flight controller since it is an external piece of hardware that is interfaced to it. One extremely important detail is that the GPS had to be an acceptable distance from any interfering magnetic fields so as not to interfere with its readings. To solve this issue, we mounted the GPS on a stand several inches above the frame to avoid interference. The GPS faced the sky clearly and pointed towards the front of the copter. The device ran at 3.3 V so it was consistent with the Pixhawk microcontroller on the copter.

U-Blox GPS:

- 5 Hz update rate
- 25 x 25 x 4 mm ceramic patch antenna
- Rechargeable 3V lithium battery pack

- Low noise 3.3 V regulator
- I²C EEPROM for configuration storage
- Power indicator LEDs
- APM-compatible 6-pin DF13 connector
- Exposed RX, TX, 5V, and GND pad
- Size: 38 x 38 x 8.5 mm
- Weight: 16.8g

Because the u-blox was be connected to the I^2C splitter module, it was able to communicate simultaneously with the Pixhawk. This is helpful because no additional wiring is needed in order to connect to the GPS.

7.1.3. Power Source

As mentioned in the specifications of the Pixhawk listed above, there were several different ways to power the quadcopter. Depending on whether the copter is being tested in a closed environment or being flown outside, there are multiple combinations of power sources. The primary ways of powering the different components of the quadcopter were through USB, ESCs, and battery-eliminator circuits (BECs).

ESCs are motor controllers that are used to increase or decrease the speed of the blades on the copter that adjust its speed and orientation. The ESC received input from the flight controller (which housed the accelerometer, gyroscope, and other measurement devices used to determine how to keep the copter in flight) and used these inputs to control the speed of the motors. ESCs contain their own processors and firmware in order to correctly process the information being fed to them. While researching ESCs, there were several important factors to consider. The first was amp rating. The amp rating of the ESC needed to be higher than those of the props/motors or else it would overheat and die. The second was the refresh rate. Typically, updates are sent from the flight controller at a rate of 400 MHz or greater, so the ESC needed to be to deal with the speed of these updates. If the ESC was not designed for the specific copter being powered, it would need to be flashed with new firmware, a tedious process that could be avoided by choosing the right one before purchase. Finally, the size and energy usage were important. If the ESC was too heavy or gave off too much heat, it would need to be cooled on-board. There were several 4-in-one ESCs that were perfect for our medium-sized copter (were able to power all four propellers at once) that draw far less power.

20-Amp SimonK Electronic Speed Controller:

- 3 start modes: normal, soft, super-soft
- Programmable throttle range
- Separate Voltage regulator for on-board microprocessor
- Max supported motor speed (6 poles): 70,000 RPM
- 20 Amp continuous current and 25 Amp burst current (10 sec max)
- BEC output 5V 2A

- Battery: 2-4 LiPo
- Weight: 21g

BECs are smaller devices that eliminate the need for an additional receiver or servo battery pack. They draw from the high voltage that is used to power the blade motors and convert it to a lower-level voltage that is in turn used to power the receiver.

In addition to these, a power distribution board (PDB) was needed to properly power all of the different parts of the copter, many of which required different voltage levels. The PDB would draw power from a 14.8 V 4 Amp Lithium Polymer Battery (specifications below). The battery contained an XT60 connector and a JST-XH charging connector.

Quad Battery Pack:

- 4S 14.8 V
- 4000 mAh
- XT60 Connector and JST-XH charging connector
- Dimensions: 31 x 51 x 146 mm
- Weight: 407 g

7.1.4. Quadcopter Power Distribution

In order to correctly distribute power to all the different parts of the quadcopter, the 3DR Power Module (PM) was be used as the central power distribution board. There were several advantages to using this board, the biggest of which was that it is made to efficiently communicate with the PixHawk, GPS, and motor controllers of our specific copter. The power module provided a consistent 5.3 V and 2.25 amps to the Pixhawk and also had a built-in feature where if the power level began to reach capacity, it triggered a return-to-base safety flag that would send the copter back to its launch location or land it immediately to prevent over-powering the unit and possibly frying the circuits. Additionally, the PM allowed the firmware loaded on the quadcopter to compensate more accurately for the magnetic interference affecting the compass as a result of the board and multiple ESC's. It is important to note, though, that while the PM did provide sufficient power to a lot of the copter's components, it did not directly provide power to the servo motors; these were controlled by the ESC's.

	3DR Power Module
Max. Input Voltage	18 V
Min. Input Voltage	4.5 V
Max Current	90 Amps
Weight	38 g
Flight Battery Cable	6" 14AWG red/black cable
ESC Cables	4 female Deans connectors 1 XT60 connector

Table 7-1: 3DR power module summary

The 3DR Power Module connected to the 4S Lipo battery we had chosen to power the copter and could handle a maximum voltage input of 18V and a maximum current flow of 90 amps. However, when paired with the Pixhawk, only up to 60 amps could be measured. The setup was very straightforward when using Mission Planner's software; it allowed you to control the voltage and current levels, as well as set flags for when the power usage reaches a certain percentage of its maximum or when the battery dipped below a certain power level.

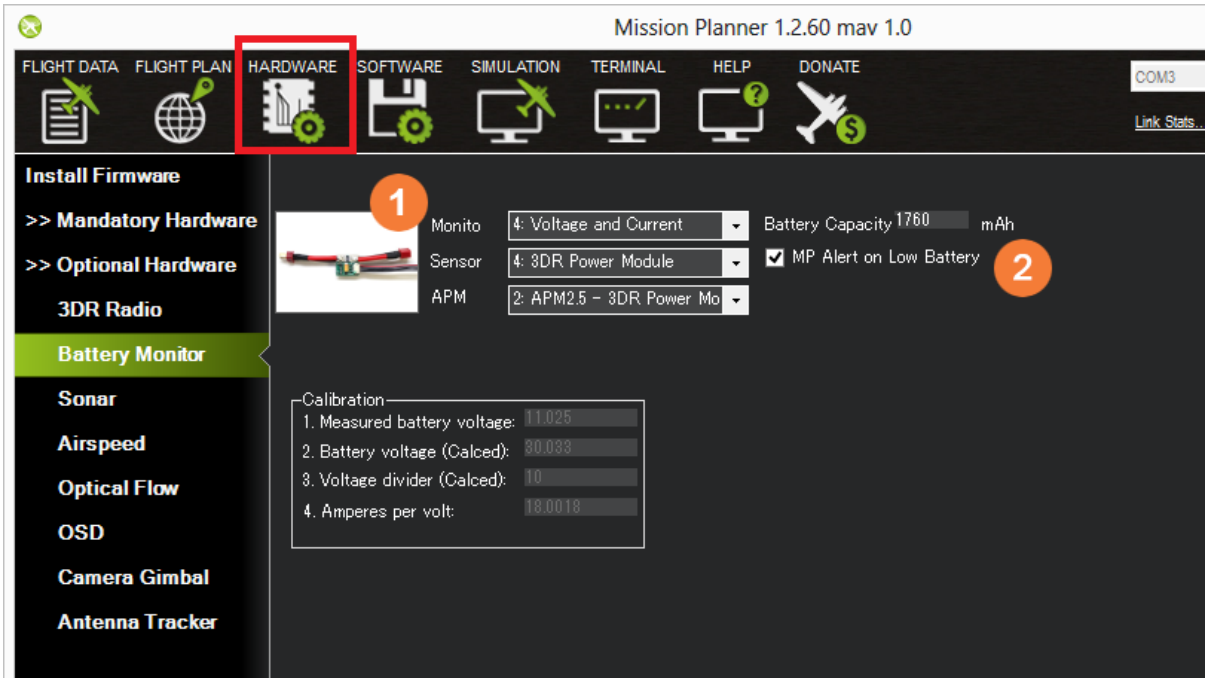

Figure 7-1: Power connection screen

When the wiring of the copter was completely finalized, the wiring diagram below reflected the final connections between each of the components needed to control the quadcopter. As can be seen, the PDB distributed power to the Pixhawk and the 4 ESC's, but not the propeller motors directly.

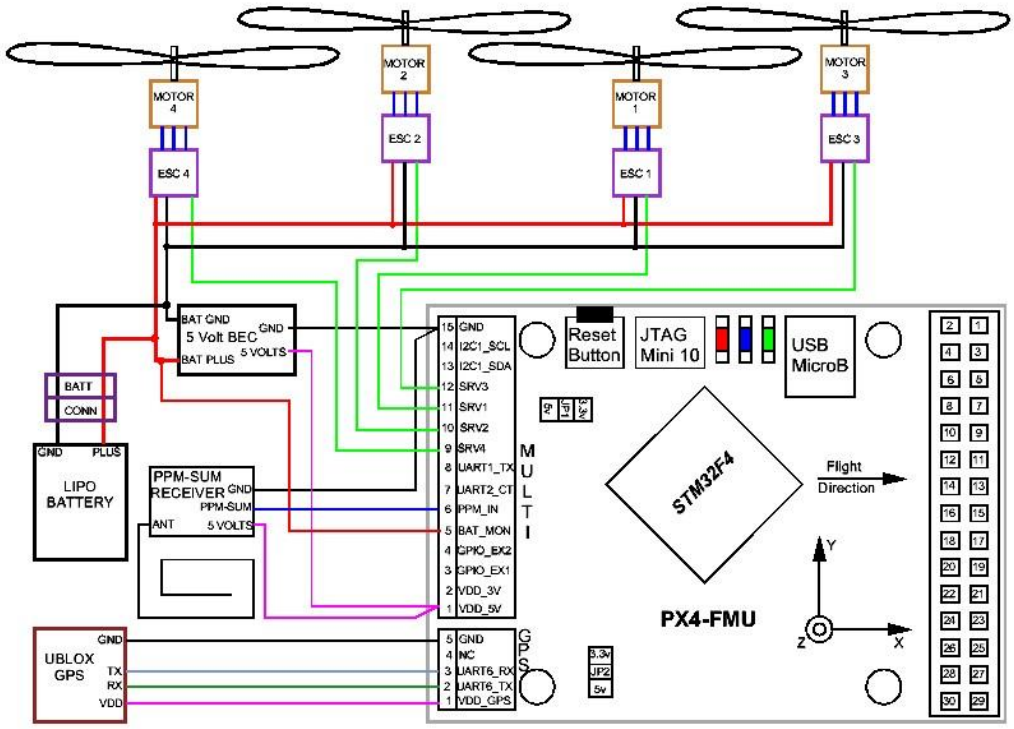

Figure 7-2: Quadcopter wiring diagram. Permission from Arducopter. See Appendix B for details.

7.1.5. Remote Controller/Receiver

Although most of the initial testing was done inside and in a controlled environment where the quadcopter was not actually flying, once the copter was tested in the field we needed a way of controlling it in case anything went wrong. The copter was supposed to fly autonomously, but there needed to be a method to regain control and return the copter to base safely so as not to damage any of the components if the copter were to crash. For that reason, we needed a remote controller/transmitter to guide it back to its launch point.

Manual flight control can be accomplished by coupling a Remote Control (RC) transmitter and receiver. The two can communicate via telemetry and the Mission Planner control station to ensure safe guidance of the copter in an emergency situation. For our project, we used both telemetry as well as controllers and receivers to ensure full control of the copter.

When setting up the controller, there were multiple channels that controlled different parts of the copter. For example, normally one channel controls throttle, one controls turning left or right, one controls rolling left or right, and one controls the pitch forward and pitch backward. When considering a controller, the amount of channels it has needs to be considered: there are 4-channel, 5-channel, 6-channel, 8-channel, and 9-channel controllers. Four channels are needed at the minimum for a quadcopter (one for pitch, one for roll, one for throttle, and one for yaw), so any of the previously mentioned controllers could work. Additional channels allow the user to control other parts of the quadcopter while in flight, such as potentiometers or flying in different modes. Below is a diagram of how a 5-channel controller can be used to fly a quadcopter.

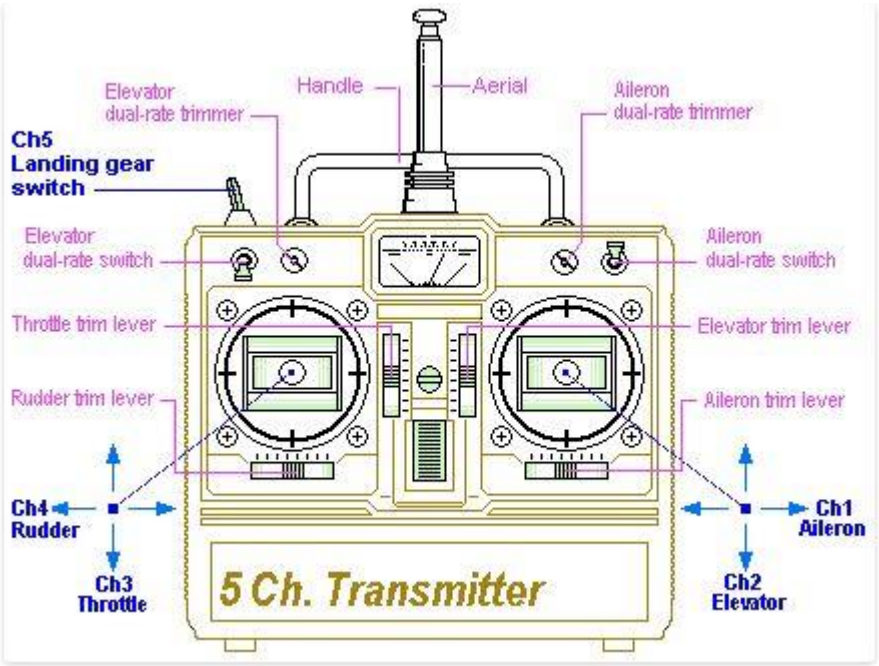

Figure 7-3: Remote controller layout example. Permission from Oscar Liange. See Appendix B for details.

When flying the copter, there are two primary modes that it can be flown in, commonly known simply as Mode One and Mode Two. Mode One has the elevator and rudder control on the left joystick and throttle and aileron control on the right joystick. Mode Two, however, is the more common setup when flying a quadcopter. In this setup, throttle and rudder are on the left joystick where elevator and aileron control are on the right joystick. The right joystick centers itself in both axes, where the left joystick only centers itself in the horizontal axis. Below is a visualization of the two different possible modes.

Figure 7-4: Mode One and Mode Two example. Permission from Oscar Liange. See Appendix B for details.

When deciding on one transmitter and receiver to consider purchasing, there were several key factors that need to be considered. The first was most obviously price, as just the transmitter can run from $20 to well over $1000, a huge price that did not fit our budget at all. The second, as mentioned before, was the number of channels. Arducopter recommended that the transmitter have at least six channels in order to efficiently control the copter. Other factors to consider were the mode it comes in (either Mode One or Mode Two), frequency it operates at, and the weight of the receiver it comes paired with (as we didn't want a receiver that would weigh down the quadcopter too much). Below is a table comparing the different transmitter/receiver combinations that were considered. After weighing the options, we decided to go with the Turnigy 6X receiver/transmitter pairing due to the lightweight receiver, cost effectiveness, and FM modulation.

In the end, Group 4 actually ended up going with the Turnigy 9x receiver/transmitter due to the limited turnaround with most of the other transmitters. The Futaba was out of stock at almost all online stores, and the 6X would have taken around two months to ship as it was backordered at most places. The Turnigy 9x turned out to be a great option as it had maximum channel flexibility, was in our price range at around $80, had a lightweight receiver, and was able to ship in less than three days.

Transmitter	Number of Channels	Mode	Modulation	Frequency (GHz)	Receiver Weight (g)	Price
Turnigy 6X	6	2	FM	2.4	11.5	$30
HK-T6A-M2	6	2	FM	2.4	15	$25
HK-6DF-M2	6	2	FHSS	2.4	14	$31
Futaba 6EX	6	2	FASST	2.4	7	$170

Table 7-2: Remote transmitter comparison table

7.1.6. PPM Encoder

After ordering the Turnigy 9X and attempting to connect it to the Pixhawk, it was discovered that there are two different types of common ESC/servo control used in the R/C field: PPM and PWM. The primary difference is where pulse-width modulation uses individual wires for each channel to monitor the amount of time a signal is set to 'high', pulse-position modulation sends the channels successively over a single wire. Where PWM measures the length of time the signal is set 'high' to determine the throttle or position, PPM references the signals relative position in a certain time division. Below Figure 5-9 illustrates the differences between the two types of signals.

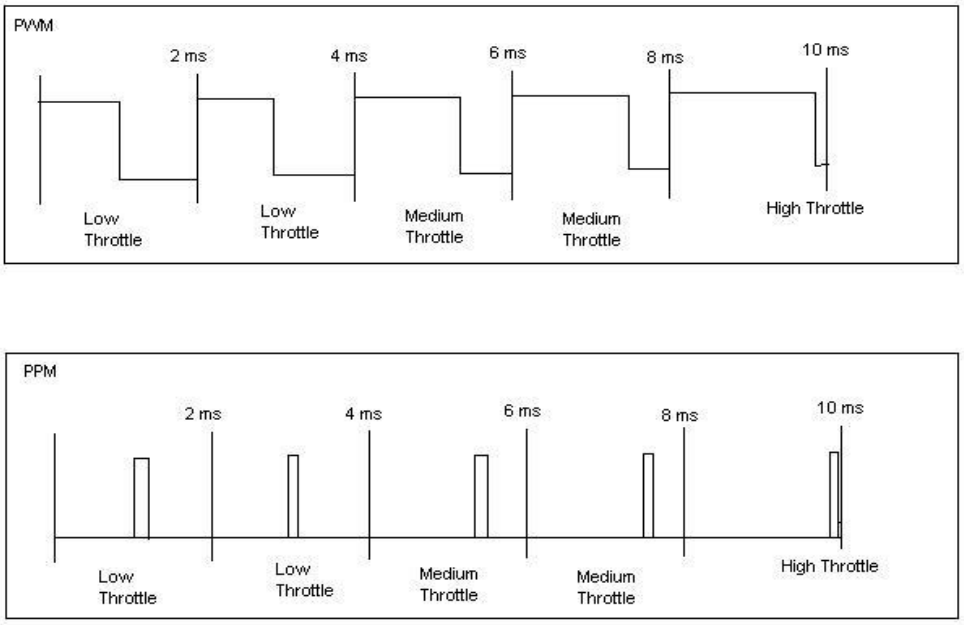

Figure 7-5: Illustration of PWM vs PPM protocol

The Turnigy 9X used the PWM protocol, where the Pixhawk only received PPM signals to controls the ESCs. As a result, a PPM encoder was necessary to convert the PWM signal of the receiver to a single-sum PPM signal the Pixhawk could use to control the motors. Since 3DR already makes a PPM encoder, it was decided to use theirs. Below is a summary of the specifications of the PPM encoder.

3DR PPM Encoder:

- ATmega 328 p microcontroller
- PPM input cable to PWM receiver DF13 10-position to servo
- 22 x 19 x 5.5 mm

7.1.7. Telemetry Modules

As far as the telemetry radios, our group chose the 3DR radio set, two telemetry radios that operate at 915 MHz (US standard) and allowed us to communicate directly with the quadcopter via the Mission Planner and while running the CDC to pull the images from the GoPro at each waypoint. By being able to communicate with the copter, we were able to fine-tune the mission in-flight, monitor data real-time (allowing us to cross-reference the data being streamed to the Android application we develop and ensure it is accurate), and even change the flight mode in case something goes wrong. Below is a layout of the specifications for the telemetry radios we used.

3DR Radio Set:

- 6-position DF3 connector
- 100 mW maximum output power
- -117 dBm receiver sensitivity
- 2-way full-duplexcommunication
- 3.3V UART interface
- Supply Voltage 4.7-6 VDC (from USB or DF13 connector)
- Transparent serial link
- MAVLink Protocol Framing
- FHSS (Frequency Hopping Spread Spectrum
- 26.7 mm x 55.5 mm x 13.3 mm

7.2. Rover

7.2.1. Chassis

The chassis selected for the SARS system is mostly prefabricated. The selected chassis includes a frame with a suspension system as well as the 6 necessary motors and wheels. Figure 5-9 and Figure 5-10 show the main dimensions of the chassis. The main frame is 120mm across by 380mm long by 89mm high. The total height of the chassis with the wheels is 135mm. Each wheel has a diameter of 126mm. The frame is made out of anodized aluminum plate with stainless steel and nickel plated brass fittings. The chassis also has a built in suspension system that allows for the

rover to traverse virtually any terrain. The suspension system is based on spring loading the individual motor housings allowing for each wheel to move independently.

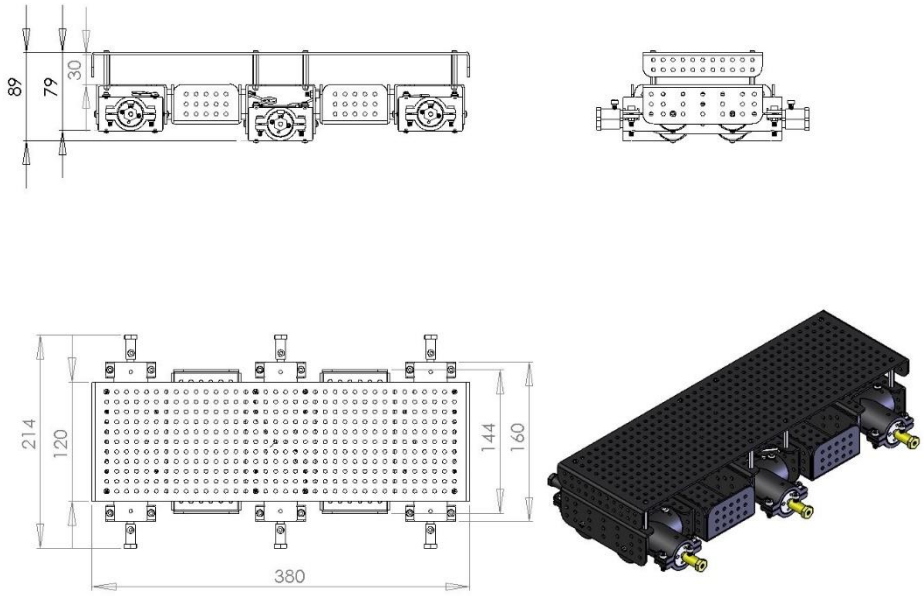

Figure 7-6: Rover chassis dimensions without wheels (Permission to reproduce pending)

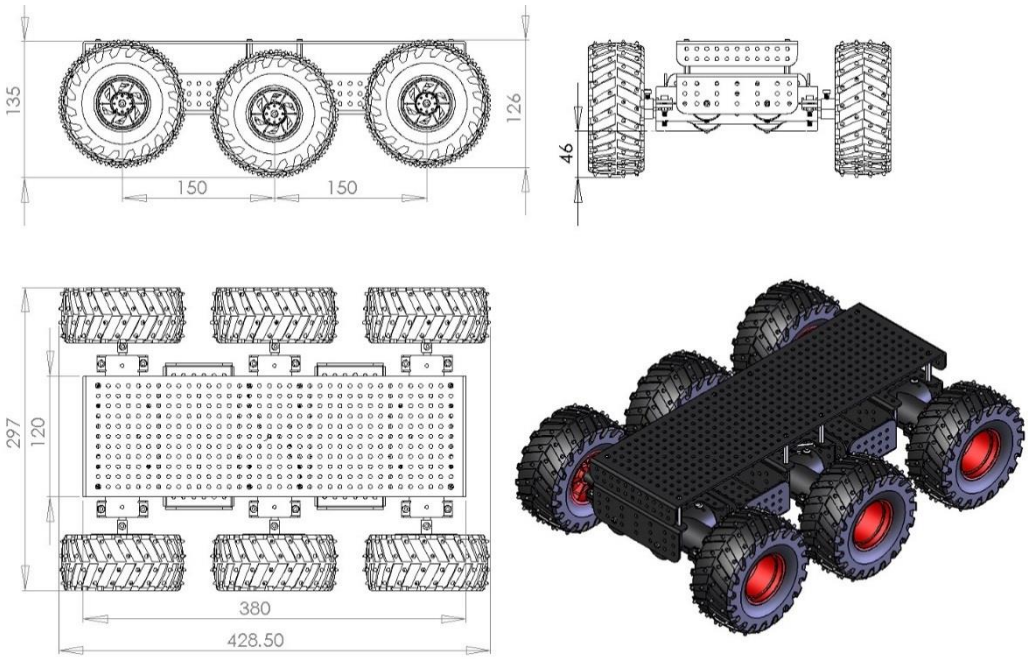

Figure 7-7: Rover chassis dimensions with wheels (Permission to reproduce pending)

7.2.2. Motors/Motor Controller

The selected chassis also includes 6 motors to be used with the system. Each of the motors is rated for 6V DC with a stall current of 5.5A. Each motor is also rated for a RPM of 10000 with a gearbox speed ratio of 75:1. The motors have an output shaft speed of 133RPM and a stall torque of 11Kg/cm. Each motor has an independent housing. The housing is designed to prevent the motors from vibrating out of place.

The motors were wired up to a motor controller for power and control. The selected motor controller is the Dimension Engineering Sabertooth 2x25 motor controller. Table 7-3: Sabertooth 2X25 Motor Controller Specifications illustrates the specifications of this motor controller. This controller is capable of handling an input voltage of 6-24V and an output current of 25A per channel continuous and 50A per channel peak. The motor controller has support for 2 separate control channels. Three motors were attached to each control channel and the motors are wired in parallel for consistent current across the different motors. Assuming the motors are identical, all 3 motors on each channel will also have roughly the same voltage across them. This motor controller is designed with heat sinks included to help dissipate the heat that will be generated while it is running. These help to keep the controller cool and prevent damage to it. Also, by mounting the controller directly to the rover chassis, which is also made of metal, the entire chassis acts as an additional heat sink for the controller, giving it maximum heat dispersion. Heat dispersion is very important in high current applications to prevent damage from all aspects of the system. The power source for all of the rover electronics was also connected to the motor controller, which distributes power to all necessary aspects of the system. More detail regarding power is discussed in section 5.3.6.

Sabertooth 2X25 V2			
Drive Channels	2	Input Modes	Analog
Continuous Amperage (Per Channel)	25A		R/C
Peak Amperage (Per Channel)	50A		Simplified Serial
Nominal Voltage	6-30V		Packetized Serial
Maximum Voltage	33.6V	Dimensions	60 x 80 x 21 mm
Weight Support	Up to 300lb	Weight	90g
Operating Modes	Independent Speed + Direction	Protection	Thermal and Overcurrent
Lithium Compatibility	Yes	Regenerative Drive	Synchronous

Table 7-3: Sabertooth 2X25 Motor Controller Specifications

7.2.3. Control Boards and Sensors

The control boards acted as the brains of the entire rover. The control boards were in charge of sending movement operations out to the motor controller, interpreting sensor data regarding wheel rotation, interpreting data from the inertial measurement unit, interpreting sensor data for object detection and avoidance, communicating with the GPS module, and communicating with the other aspects of the SARS system through the communications module.

The first, and one of the most important connections was the connection between the Tiva C control board and the Sabertooth motor controller. The Sabertooth motor controller is compatible with the RS232, so a UART communication configuration was used to send commands from the Tiva C to the Sabertooth. The next important operation was interpreting input data about wheel rotation speeds. Wheel rotations were measured using a magnet and a Hall Effect sensor. The Hall Effect sensors were connected to the Tiva C using GPIO. The sensors work by setting the data pin high when a magnetic field is detected. Based on this principle, a small magnet attached to the wheels, allowing the sensor to be made aware of how fast each of the wheels are rotating. This information was relevant in the software design.

Hand in hand with the Hall Effect sensors was the inertial measurement unit (IMU). This unit was based on a Texas Instruments Booster Pack to be used along with the Tiva C development boards. This Booster Pack communicates with the Tiva C using a combination of UART and I^2C communications across a set of 40 pins, not all of which are actually in use. These pins are illustrated in Table 7-4.

Pin	Function	Pin	Function	Pin	Function	Pin	Function
J1.1	3.3 V IN	J2.1	Ground	J3.1		J4.1	
J1.2		J2.2	Interrupt	J3.2		J4.2	
J1.3	UART RX	J2.3	Interrupt	J3.3		J4.3	RF GPIO
J1.4	UART TX	J2.4		J3.4		J4.4	UART RTS
J1.5		J2.5		J3.5	USER LED	J4.5	UART CTS
J1.6	Interrupt	J2.6	Sensor I2C	J3.6	User Button	J4.6	RF Shutdown
J1.7		J2.7	Sensor I2C	J3.7	User Button	J4.7	RF Reset
J1.8	SSI TX	J2.8	SSI RX	J3.8	RF GPIO	J4.8	RF GPIO
J1.9	RF I2C	J2.9	SSI SS	J3.9	RF GPIO	J4.9	
J1.10	RF I2C	J2.10	SSI CLK	J3.10		J4.10	

(1) Shaded cells indicate unused pins that are available for additional expansion.

Table 7-4: BOOSTXL-SENSHUB Sensor Pack pin locations (Permission to reproduce pending)

While the initial selection of IMU was to use the SENSHUB Booster Pack, interfacing with this device proved too daunting, and the device also provided large amounts of superfluous data that was unnecessary in the operation of the rover. Instead, we elected to use the HMC5883L magnetometer. The magnetometer provides data that is converted into a 0 to 359 degree compass heading so that the Tiva is aware of the rover's current heading. This module communicates with the Tiva via I^2C. It has an accuracy of 1 to 2 degrees. Required power for the module is 5V.

The Tiva C was also connected to a sensor to be used to detect objects in the travel path of the rover. The SARS Rover employed a Parallax PING))) Ultrasonic Distance Sensor. The PING)))

operates using a pulse in/pulse out pair of pins. It is powered directly from the Tiva C board. In order to maximize functionality of the ultrasonic sensor, it was mounted on the front of the rover housing, relatively close to the ground, to ensure maximum coverage in front of the rover.

The final sensor used with the rover system is the GPS. The GPS module selected for use is the Adafruit Ultimate GPS Breakout. The GPS connects to the Tiva board using UART. The Tiva C uses the GPS module to influence many of the other actions that it makes. This GPS module is accurate to less than 3 meters, which is within the specified range of operation of the PING))) sensor. This allows for seamless operation when approaching the target object. Relevant data for this module is available in Table 7-5.

Adafruit Ultimate GPS Breakout			
Tracking Satellites	22	Tracking Sensitivity	-165 dBm
Searching Satellites	66	Vin Range	3.0-5.5 VDC
Update Rate	1 to 10 Hz	Operating Current	25mA Tracking
Position Accuracy	< 3 meters		20mA Navigation
Velocity Accuracy	.1 m/s	Communication Protocol	NMEA 0183
Warm/Cold Start	34 seconds	Default Baud Rate	9600
Acquisition Sensitivity	-145 dBm	PRN Channels	Up to 210

Table 7-5: Adafruit Ultimate GPS Breakout Specifications

The last remaining aspect of the control system for the rover is the communications module. The Texas Instruments CC3100 communications module connected to the Tiva C using UART. Once the CC310 was configured, it allowed the rover to set up a small ad-hoc network to interface with the CDC. Interactions mainly included receiving start commands and transmitting telemetry data as requested.

Error! Reference source not found. shows the overall wiring design of the rover system. It llustrates the general wiring configuration used within the rover for power and data across all subsystems. The only modules that the microcontroller sent data out to were the motor controller and the CC3100, the rest of the modules were primarily for receiving data from sensors or outside sources.

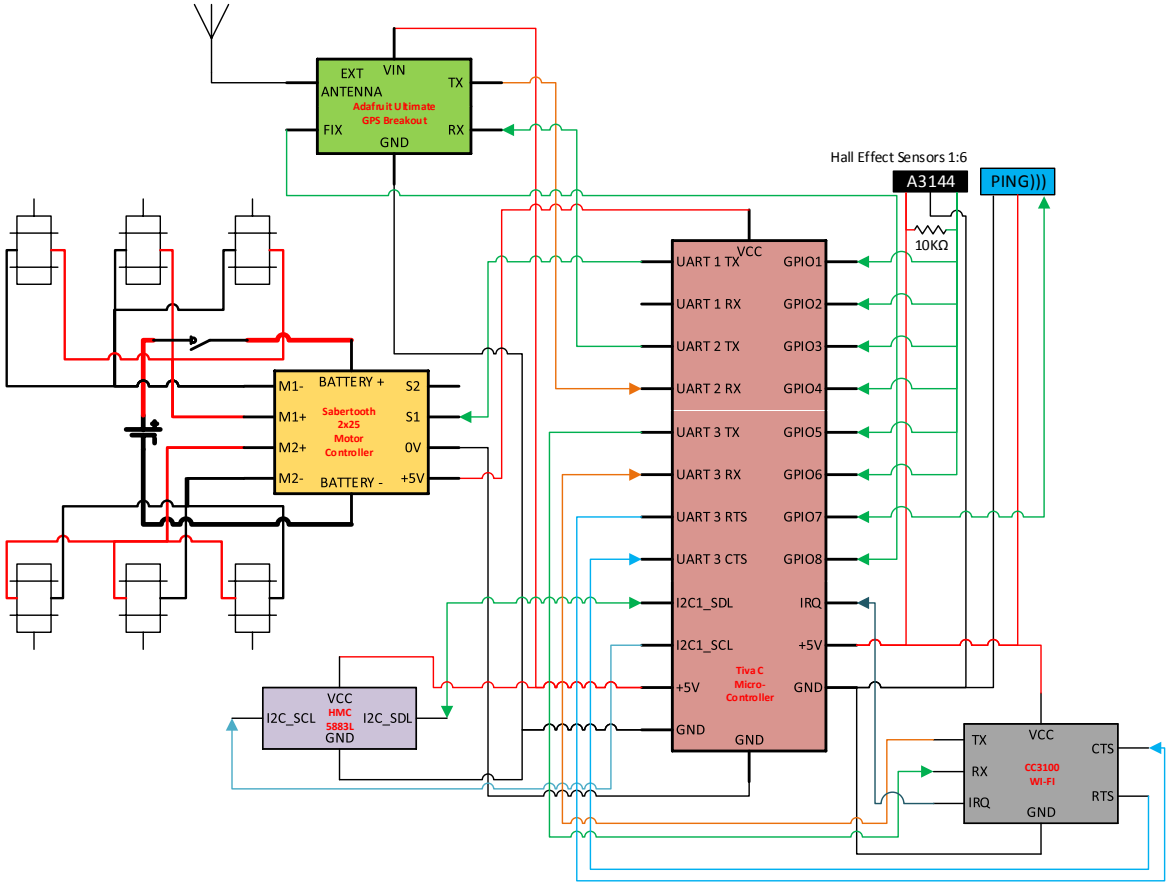

Figure 7-8: Rover Internal Communications Wiring Diagram

7.2.4. Object Retriever

At the early stages in the development of SARS, the final method of object retrieval had not yet been decided upon. Two designs were initially being considered for prototyping. Basic design information is outlined for each of these potential prototypes, as well as the final implemented design.

7.2.4.1. Universal Gripper

The Universal Gripper is a device designed by researchers at the University of Chicago and Cornell University to grab and lift any object. It involves lowering a balloon filled with coffee grounds onto an object to be lifted. Once the object is engulfed by the balloon, a vacuum is applied. The balloon tightens over the object which can then be lifted up. To implement this system, this balloon would need to be suspended at the end of a robotic arm, mounted on the front of the rover with three degrees of freedom. This setup is displayed in Figure 7-9: Robotic Arm Design below.

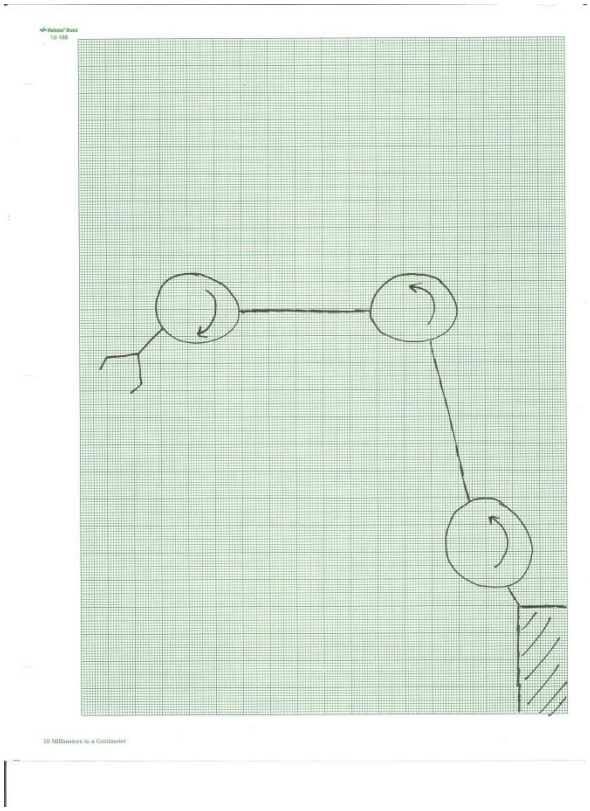

Figure 7-9: Robotic Arm Design

This design requires three degrees of freedom. The first degree of freedom, the bottom joint in the image, lowers the entire arm. The next degree of freedom, the middle joint in the image, rotates with the same orientation as the first degree of freedom to ensure that when the Universal Gripper is lowered, it travels in a straight line directly toward the ground. The final degree of freedom, the top joint in the image, rotates with the opposite orientation as the other two degrees of freedom to ensure that as the Universal Gripper is lowered, it does not tilt but faces straight down toward the object being retrieved.

Each of these degrees of freedom requires a servo motor and an encoder; however, because the robotic arm only needs to serve one function, Group 4 may not use an encoder and may instead try to hard code the rotations for the degrees of freedom.

For this system to work, the balloon needs to be secured to the end of the robotic arm. Group 4 will use a 3D printer to make a bowl-shaped casing to house the balloon. This casing will be attached to the end of the robotic arm and will have a hole in its base to allow the vacuum tubing to enter the balloon. This setup is displayed below in Figure 7-10: Universal Gripper Casing. The case would be fitted with a mount to connect it to the end of the robotic arm; however, as the materials to be used in the arm are still up in the air, this fitting is yet to be determined.

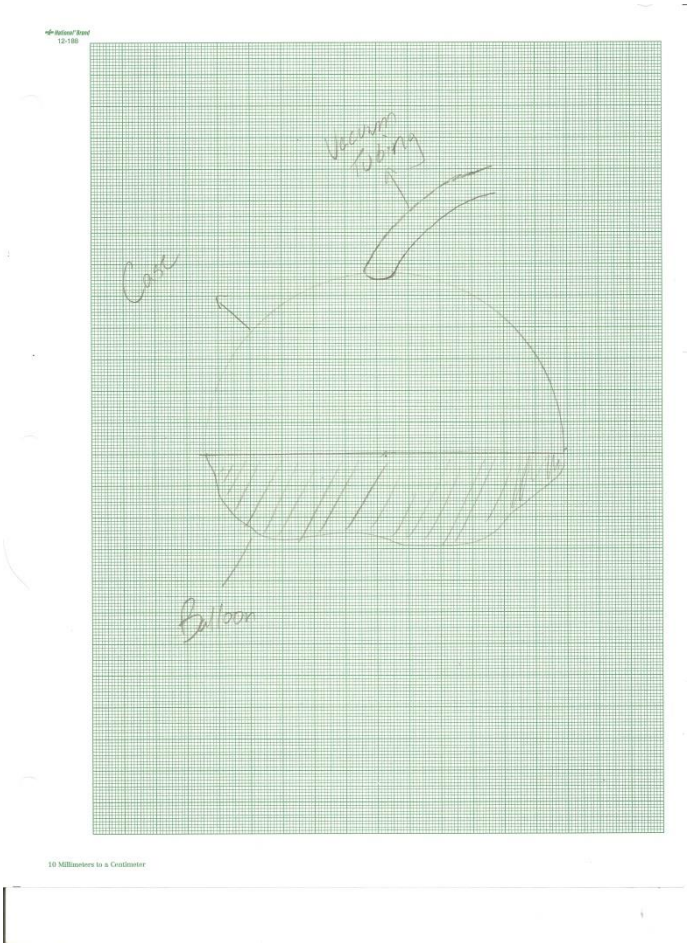

Figure 7-10: Universal Gripper Casing

The tubing will lead into a vacuum pump mounted on the SARS Rover. Group 4 has found a useful tutorial on how to convert a $10 electric air compressor into a vacuum pump. The air compressor has an air intake, an air outtake, and a tube leading to an air pressure gauge, and a piston for directing air from the intake to the outtake. The air intake needs to be sealed with epoxy. Then the line leading to the air pressure gauge needs to be cut and covered with a makeshift valve. This valve may be fashioned out of a sheet of metal and a piece of paper. The air outtake line will then be fed into the Universal Gripper Balloon. Now, when the piston fires, outtake 1 will always be open when outtake 2 is closed, and vice versa. Air will be pulled out of the balloon, but the valve on the second line will prevent air from reentering the balloon, effectively creating a vacuum. The valve on the second line can be created using a sheet of aluminum flashing, a rivet, and a sheet of paper. A diagram of this design is displayed below in Figure 7-11: Vacuum Pump Design.

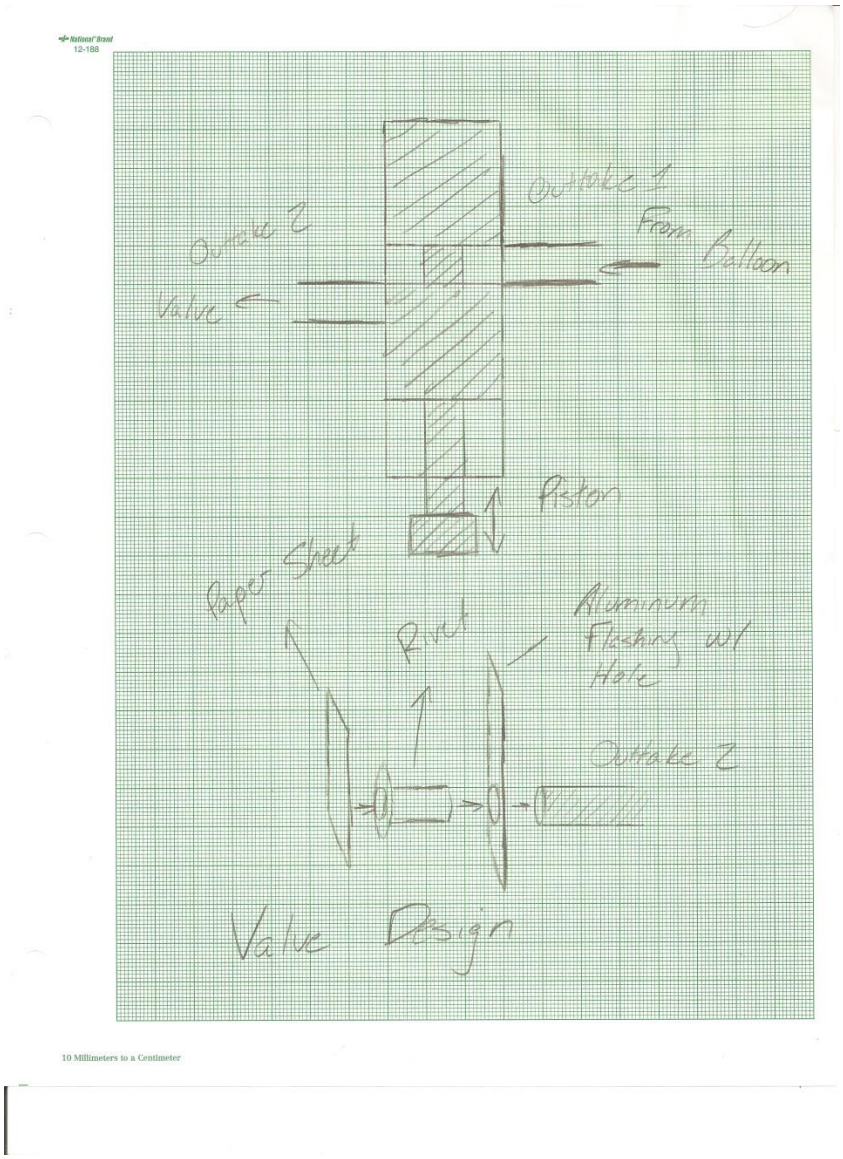

Figure 7-11: Vacuum Pump Design

7.2.4.2. Electromagnet

If Group 4 decides not to go with the Universal Gripper and instead uses an electromagnet to retrieve the target object, this electromagnet would still be mounted on the end of a robotic arm. Unlike the arm for the Universal Gripper displayed in Figure 7-9: Robotic Arm Design, it may be possible to get away with only two degrees of freedom for this robotic arm since the magnet only needs to hang over top of the target object. It does not need to be pressed directly down onto it. The robotic arm is not entirely necessary, as the electromagnet could be mounted underneath the chassis of the SARS Rover to pick up the target object when the rover drives over it.

7.2.4.3. Velcro

The final design for the retrieval arm includes an arm with a single degree of freedom. The arm itself was built out of small aluminum beams with two separate pieces connected at a 90 degree angle. The end of the arm was fitted with a sheet of Velcro, which was used to lower onto the

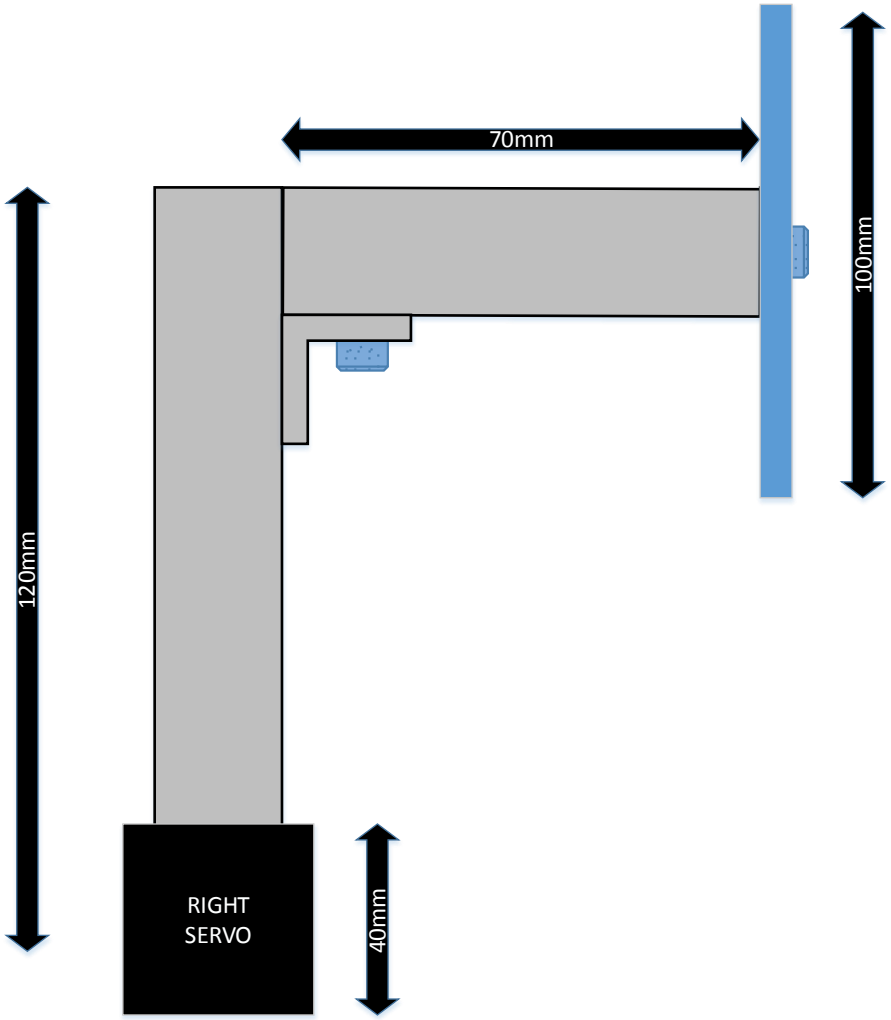

Figure 7-12: Velcro based retrieval arm design

target object and lift it up. The other major component of the arm was two servo motors, one connected to each side of the arm. By controlling the arm with two servo motors we were able to ensure that the weight of the object and the arm would not be too much to handle, and therefore limit control over the arm. Each servo is a simple analog feedback servo. They are designed to operate on 5V. The servos can rotate at a rate of .21sec/60° and each has a max torque of 6.5Kg*cm.

7.2.5. Power

The finishing aspect of the rover hardware design was the inclusion of the power system for the rover. Power to the rover was provided by a lithium polymer (LiPO) battery. The selected battery for this system has three secondary cells and one primary cell. It is designed to output 11.1V and has a capacity of 5000mAh. This battery also has is rated for 20C continuous and 30C peak output. This translates to a maximum continuous current output of 100A and peak current output of 150A. As was mentioned in the section discussing the motor controller, the motor controller has a max throughput of 64A. Based on this number, the selected battery can handle significantly higher output than the motor controller, making it an excellent candidate for usage on this system. Another important aspect in battery selection is the overall weight of the battery. This battery weighs 536g. This was a reasonable weight for the battery pack for this system. At roughly half a kilogram, the battery did not cause an excessive amount of torque on the motors it is mounted over. It was also large enough that its weight will be spread out fairly evenly across the chassis. Had a different type of battery, such as a sealed lead acid battery, been selected for use with this system, weight would be a significantly greater concern.

Now that the specifications of the battery have been discussed, mounting the battery can be covered. Due to the fact that the battery is slightly wider than the chassis, it was mounted so that the battery sat elevated within one of the storage compartments built into the chassis. Once a proper mounting space was selected, the battery was held in place using screws and rubber bands. Screws held up the battery and the rubber bands secured it to the chassis.

The final aspect of power for the rover is wiring it up to the control system. The motor controller, fortunately, handled most of the power distribution needs for the system. The Sabertooth has an integrated power stepper which allowed for the 5V and 500mA needed for the microcontroller and other electronics to operate. The battery was wired directly to the power terminals of the motor controller, and the controller took care of distribution to the motors and the control boards.

The only additional consideration in the power system was the addition of a simple power switch to turn the entire system on and off with ease. We selected a single pole single throw switch that handles 50A at 12VDC and 25A at 24VDC, which was well within the specifications of the other aspects of the rover design.

7.2.6. Printed Circuit Board

The SARS Group designed and built a printed circuit board to be mounted on the SARS Rover. This double-sided board was intended to connect the Hall Effect Sensors, the magnetometer, and the GPS using copper and to interface with the Tiva microcontroller and the other rover components using pin headers. The board dimensions are 2.167008 in x 3.729606 in, and it makes use 0.001378 in copper tracings. The board was designed using Eagle and manufactured by PCB-Pool. Most of the assembly was done by QMS; however, some of the pin headers were soldered onto the board by members of the SARS Group. The circuit board did not function as was intended. The GPS and magnetometer were completely unresponsive after they were soldered to the board. Because the board was not manufactured until late in the development process, not enough time could be devoted to troubleshooting once it became apparent that these components were not

working; therefore, the decision was made to exclude the circuit board from the final SARS prototype. The schematic for the board design can be viewed in Figure 7-13, and the board layout can be viewed in Figure 7-14 below.

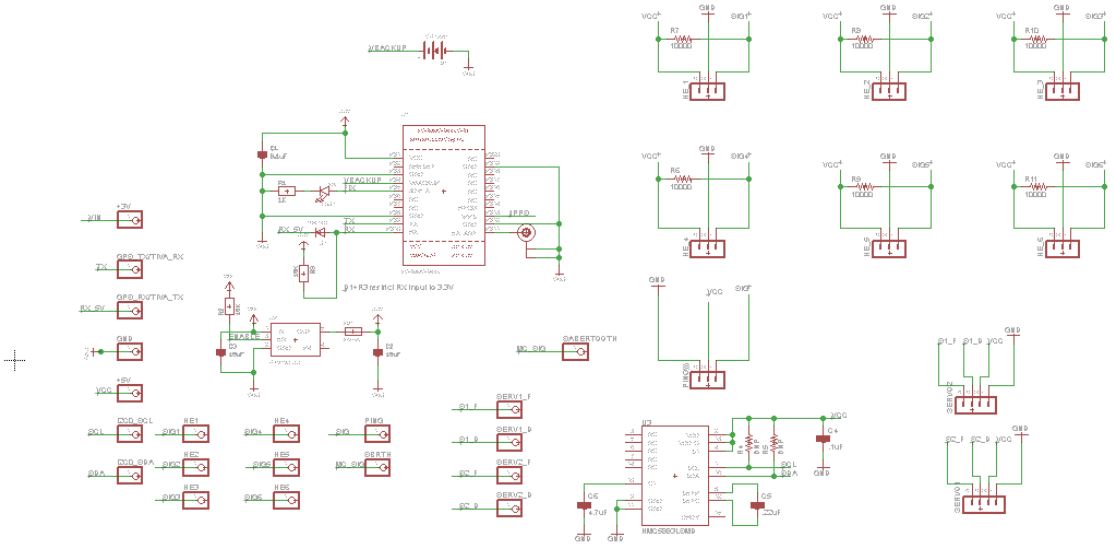

Figure 7-13: PCB Schematic

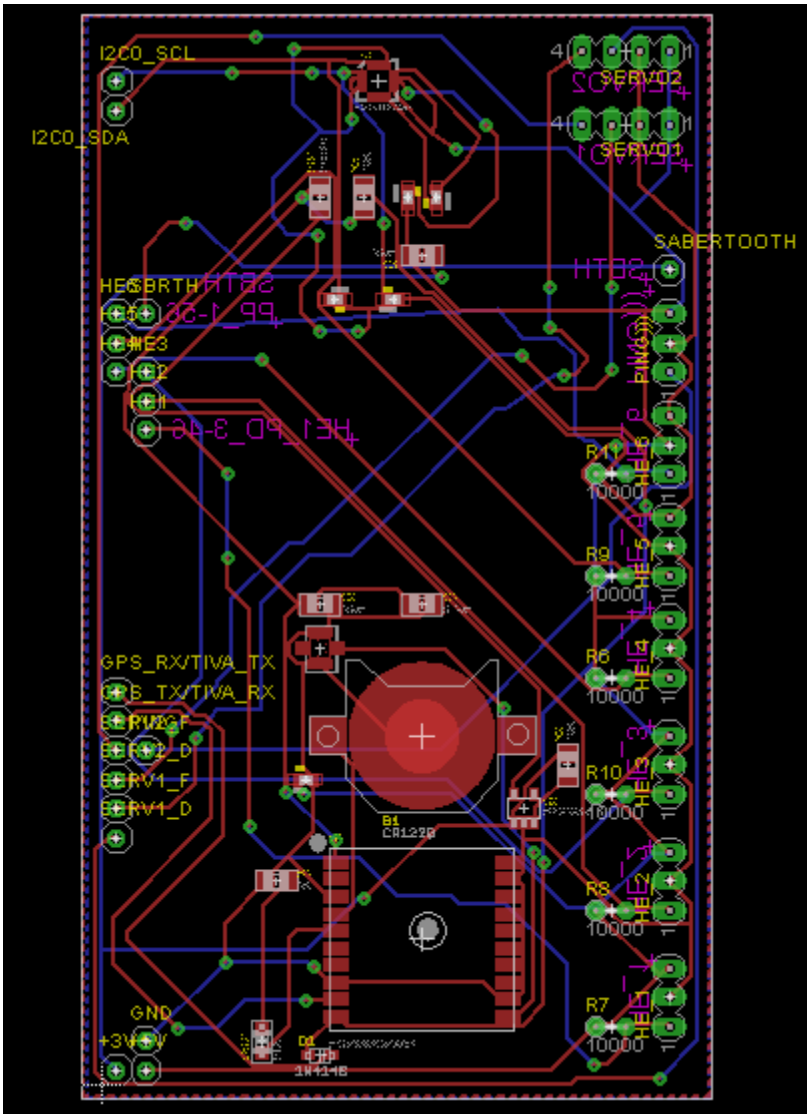

Figure 7-14: PCB Board

7.3. Wireless Communication

To enable the CDC to communicate with both the SARS Copter and SARS Rover, additional wireless communication hardware was utilized. The SARS wireless communication network consists of 3 dedicated connections, each utilizing its own wireless communication hardware interface. The three connections consist of a radio connection to the SARS Copter, a Wi-Fi connection to the SARS Rover, and another Wi-Fi connection to the GoPro camera.

7.3.1. Quadcopter

A 915MHz telemetry radio provides wireless communication between the CDC and the SARS Copter. The telemetry radio is connected via a USB port. It communicates at 57,600 baud using a

standard serial interface. In conjunction with pymavlink, this telemetry radio enables the CDC to issue commands and request parameters from the SARS Copter.

7.3.2. Rover

An 802.11n Wi-Fi connection to the SARS Rover is established using the internal Wi-Fi adapter of the laptop computer on which the CDC is running, and the CC3100 network processor connected to the SARS Rover. This connection utilizes the TCP protocol to send and receive messages between the CDC and the SARS Rover. This enables the CDC to receive status messages from the SARS Rover, and to transmit the start command which sends the GPS coordinate to the SARS Rover so that it can begin traveling to the target.

7.3.3. GoPro

Another 802.11n Wi-Fi connection is established between the GoPro camera and the CDC. This connection utilizes an external USB Wi-Fi adapter and the internal Wi-Fi module of the GoPro to provide both live video streaming and image capturing. This secondary Wi-Fi adapter is an optional, but beneficial hardware component. The GoPro camera is only capable of ad-hoc Wi-Fi connections, which means that a centralized wireless network infrastructure cannot be utilized, and both Wi-Fi connections must be handled separately. The internal Wi-Fi adapter could be used to alternate ad-hoc connections between the CC3100 and the GoPro, but this would constantly interrupt either the live video feed or the rover telemetry reporting. Utilizing the secondary Wi-Fi adapter provides a better user experience, thus the decision was made to include it.

8. Software Design

8.1. Quadcopter Software Design

8.1.1. Mission Planner

Because one of the goals for our system was to have the quadcopter fly autonomously, we needed to have a central control software that was easily integratable and allowed us to monitor the telemetry of the copter during the flight. The obvious choice was to use the Mission Planner platform, a native software/firmware integration tool that allowed us to seamlessly fly the copter while still providing a significant amount of flexibility.

Created by Michael Oborne, the software is completely open-sourced and has an extensive online library and support community that provides wide-reaching control of the copter with a sleek and user-friendly interface. Some of the features available include point-and-click waypoint entry with Google Maps integration, mission commands that can be sent during the flight, a flight simulator to create a full hardware-in-the-loop UAV simulator, and the ability to read the output from the Pixhawk's serial terminal. The last feature is especially important because this can aid in testing the flight controllers communication with the BeagleBone Black.

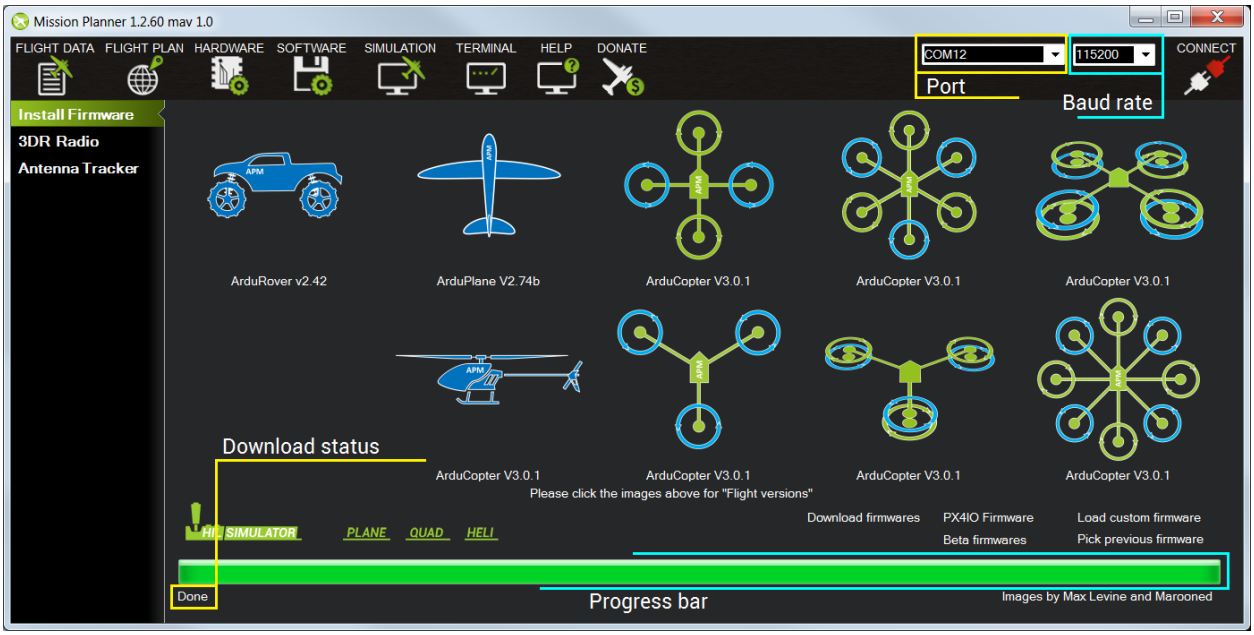

Figure 8-1: Mission Planner

Mission Planner allowed us to easily connect the different components of the copter through simple drop-down menus, as well as choose transmission rates between external modules such as the Xbee communication device. It provided firmware that can quickly be installed on the Pixhawk, as well as all the drivers necessary for the Pixhawk to correctly communicate with the control center. Finally, Mission Planner provided simple calibration for all of the on-board devices.

8.1.2. Image Processing Subsystem

The purpose of SARS's image processing subsystem is to retrieve individual frames from the GoPro camera's live video feed and analyze them for a bright white poster board. The CDC will access the video feed over a wireless ad hoc network created by the GoPro. From this point, it will use OpenCV to extract individual frames from the feed. The wireless connection with the GoPro and the frame extraction will be handled by an open source GoProController script written in Python.

The function __init__, within the GoProController script, creates the application logs and sets up and enables Wi-Fi from the CDC, and the connect function establishes the connection between the device running the CDC and the GoPro. Once the two devices are connected, the getImage function will continuously extract individual frames from the video feed. It uses cv2 to create a stream from the feed. It then uses the Image library to pull a frame from a read of the stream. The open source code contains several other function definitions, but these are the only ones needed to access the images for analysis.

The GoProController class only defines the above functions. Another Python script will need to be written to implement the object detection. This class will be called Detector. It will import the Image library, and once all the frames have been saved to an image file, Detector will analyze the image using a homemade image processing algorithm written to detect the white poster board on which the tennis ball is lying. Once the analysis is complete, Detector will return whether or not the object has been identified. If the object is found, the CDC relays the information on the waypoint from which the image containing the white poster board was pulled to the SARS Rover. Figure 8-2: Image Processing Algorithm below illustrates the algorithm being used to identify the white poster board in an image. Basically, it looks for ten consecutive, adjacent pixels whose RGB values all average out above 200.

```
for each pixel row
    for each pixel column
        if the average of the RBG values > 200
            increment white pixel count
            if white pixel count == 10
                Object Detected!
        else
            set white pixel count to zero
```

Figure 8-2: Image Processing Algorithm

8.1.3. Waypoint Navigation and Interruption

Prior to flight, using Mission Planner it is possible to create a mission consisting of a series of waypoints. Because our copter was scanning an area for an object and collecting images of the entire field, it was necessary that we planned the waypoints such that the copter scanned the full area. Mission Planner has a setting for waypoint navigation known as Auto Grid, a setting which

has the quadcopter go back and forth over a designated area in a lawnmower pattern, which was perfect for our project.

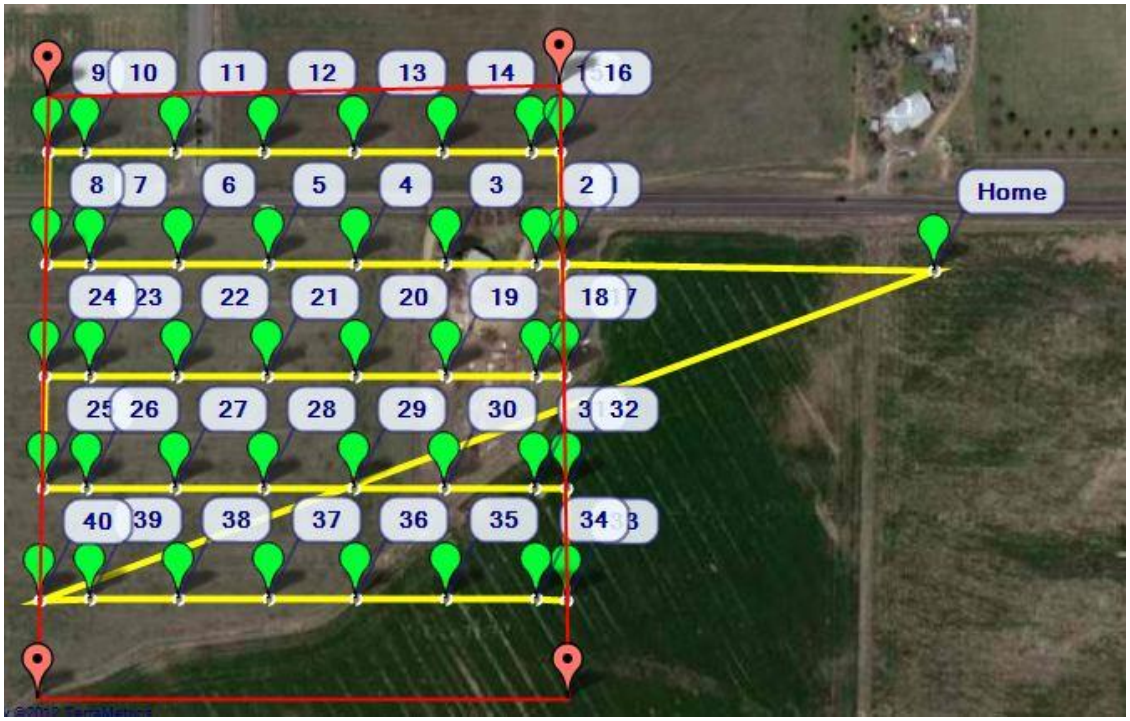

Figure 8-3: Auto Grid

The significant part of the design approaches once we have the detected the image. The copter was following a pre-determined path during its flight, but once the GoPro discovers the object we are looking for in its field of vision, the BeagleBone's image processing needs to be able to send an interrupt to the flight controller, slowly moving the quadcopter until it is hovering directly over the object. The copter will then transmit the GPS coordinates of its position to the rover, and the copter will return home.

When sent on a mission using waypoints, the quadcopter was operating in Auto mode. Once the GoPro detects the object though, the copter will need to execute a series of steps in order to complete the mission successfully. First, the copter will switch into Guided mode, a mode in which the copter receives a specific GPS coordinate as an input, and then moves toward that point and hovers above it. The BeagleBone, as it process the location of the image (how close it is to the center of the camera), will need to periodically send the next location to the flight controller. This new location can be sent using the Waypoint command, which takes in the latitude, longitude, and hold time of the next point to navigate to.

Once the object has been centered in the camera's field of view, the copter will need to switch to Loiter mode. In this mode, the copter will hover in its location for a specified amount of time. While doing this, the BeagleBone will read the coordinate from the u-blox GPS and transmit them to the rover via the Xbee communication module. The copter can be switched into loiter mode with the Loiter_Time command, which takes in the time (in seconds) in which to hold its position.

Finally, the copter can be given the Return_To_Launch command, which will override the remaining waypoints that were programmed at the beginning and cause the quadcopter to return back home as its contribution to the project will have been complete.

Because the flight controller (as well as the BeagleBone) are native to Python, a script can be sent to the copter that executes the series of commands once the image has been detected. Below is the pseudo code for what will have to be executed by the copter.

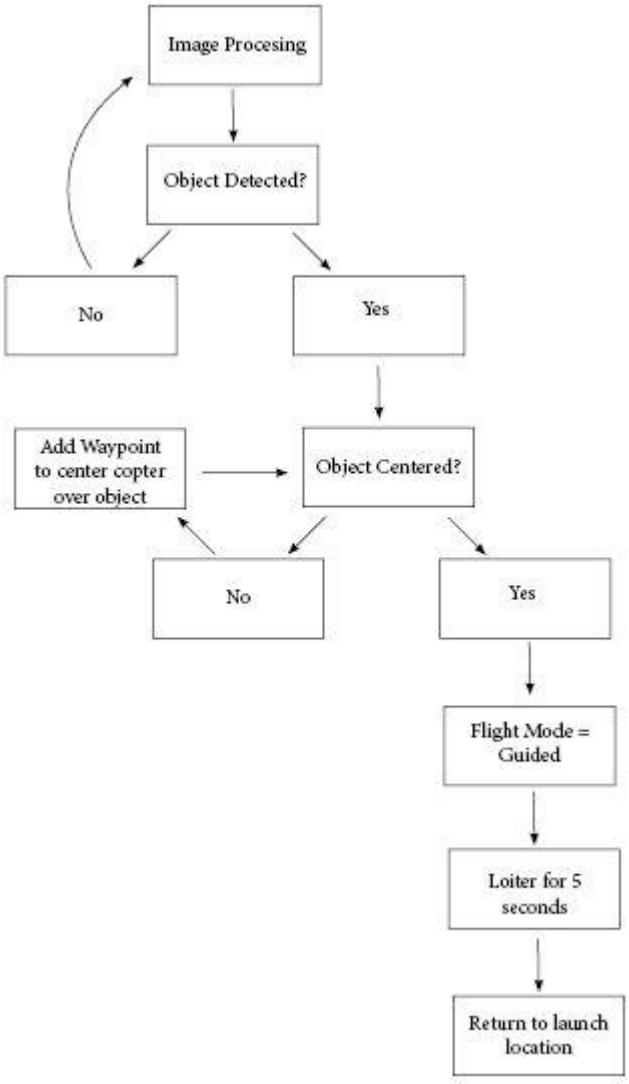

Figure 8-4: Image processing and object detection flow chart

8.1.4. Geolocation Subsystem

All of the functionality of the SARS Copter's geolocation subsystem was handled by the Pixhawk flight controller, which interfaced directly with the Ublox LEA-6H GPS module. A text file containing the GPS coordinates of the desired quadcopter waypoints along with other information such as altitude and waypoint delay can be loaded into the Pixhawk, which handles the completion of the mission involving flight to these waypoints. The navigation to these waypoints was accurate with a tolerance of 3 m; however, most of the time the Pixhawk managed to direct the SARS Copter with accuracy far better than 3 m.

The GPS information from the SARS Copter is pulled from the Pixhawk over radio communications by the CDC using the MAVLink protocol. The details of this protocol shall be discussed in depth in the CDC design section of this document.

Several issues were encountered with the functionality of writing waypoints to the quadcopter, though. The SARS Group believes that the Pixhawk flight controller was damaged at some point during the testing, and for this reason, its functionality has been compromised. Basically, the only way to get the SARS Copter to run an automated mission is to take off and land manually; furthermore, delays cannot be programmed to the Pixhawk. If a delay is written to the flight controller, the SARS Copter will hover over the specified waypoint until the battery depletes, regardless of how long it was instructed to wait. In order to get the SARS Copter to pause over a waypoint long enough for an image to be taken without pause infinitely, at least 30 waypoints without delays need to be stacked on top of each other at the exact location of the frame extraction. The time it takes for the Pixhawk to cycle through these waypoints is sufficient for the image pull.

8.2. Rover Software Design

8.2.1. Speed and Direction Control

The first and foremost aspect of rover movement is speed and direction control. It was the job of the Tiva C to send the speed commands out to the motor controller to get the rover into motion. The Tiva C was responsible for sending one or more of the fourteen possible commands to the Sabertooth controller on each iteration of the control loop. The main control loop of the software involved reading each of the sensors and then issuing the appropriate command to the motor controller.

The Sabertooth has two different control modes with a total of fourteen possible commands. One mode is the independent control mode, where each motor channel is programmed independently and commands can be issued to drive each channel either forwards or backwards. Each motor channel has three possible commands, drive forwards, drive backwards, and drive. Drive forwards and backwards both use 8 bits to define the forwards or backwards speed of the specified channel. Drive uses the same 8 bits to define whether the channel is in forward or reverse operating mode and a speed. The downside to this command is reduced control over the specific speed of the motors, but it does allow for issuing multiple direction commands with the same main command. There are also two commands that control the maximum and minimum voltages that the Sabertooth will operate on, but these commands will not be used.

The other command mode for the Sabertooth is the mixed mode command set. This command set has six possible commands and does not divide the command set based on motor channel. The controller will internally determine which direction each channel is sent in based on the command issued. This control mode has a drive forward, drive backward, and drive command with the same configuration as the independent control mode, but channel is not specified. The mixed mode command set also adds new commands for turning the rover. There are three turn control commands, turn left, turn right, and turn. Turn left and right operate as expected, and turn allows for encoding either a left or a right turn and a turn speed to be encoded in a single command. This command mode will be much easier and effective for use on the SARS rover, and will be implemented in the control software.

Now that the command set has been defined for the Sabertooth, the required code on the Tiva C for sending packets to the Sabertooth can also be defined. The Sabertooth requires a 31 bit command be sent out from the microcontroller to properly issue commands. The first byte of the packet contains the address of the Sabertooth, which is determined by physical switches on the Sabertooth. The next byte defines the command being sent. The third byte includes the data for the command being issued. The final seven bits of the command is a checksum that must be computed and included in the packet sent. Functions will be built into the Tiva C command code that will take in the desired data for each possible command being sent to the Sabertooth and will generate the packet to be sent out over the serial communication line.

After all of the functions to issue commands to the Sabertooth have been constructed, the code for deciding what commands to issue was defined. Most straight line drive commands were based on whether or not the rover is moving as expected based on checks made against object detection and GPS navigation, which will be explained later. Assuming the rover was on course and not about to collide with something, on each iteration of the control loop the microcontroller first looked to the Hall Effect sensors mentioned in section 4.3.3. The microcontroller verified that the wheels were moving as expected by analyzing the data from the Hall Effect sensors. If the Hall Effect sensors claimed that a wheel was rotating, but the software determined that the rover as a whole is not moving as expected, the software made adjustments to speed and direction commands issued because the rover is likely to be stuck on some aspect of the terrain. Corrective actions included turning as well as reversing to find a more easily traversed piece of terrain. These sensor checks were vital in keeping the rover moving forward towards its target and had the most important say in which commands are issued to the Sabertooth.

8.2.2. Object Detection

While the rover is in motion using the commands defined in the previous section, it was certainly possible that obstacles will appear in the path of the rover that needed to be avoided. Obstacle avoidance is a challenging task for any autonomous system. With functional hardware to detect obstacles, there is a large amount of software that must also go along with it. The check for obstacles must be performed on most to all iterations of the control loop, depending on the rate at which the control loop is cycled. With a more frequently operating control loop, fewer checks for obstacles must be made. If the control loop is iterating slowly, object detection checks must be more frequent. Once the frequency of checking was determined, the actual procedure for object avoidance was detailed.

At the first sign of an object being detected, the forward speed of the rover must be reduced, so a command doing such was issued. After the rover has been slowed down, the decision making process on how to avoid the object began.

The other side to object detection is target object detection. This required an entirely different set of operations from object avoidance. The GPS was relied on to get the rover within 2 meters of the target object, and from that point a coarse location of the object was made using the main object detection sensors on the rover.

8.2.3. GPS Navigation

After object detection was completed, GPS based navigation was implemented. There are two aspects to GPS based navigation, coordinate retrieval and direction determination. The first aspect is coordinate retrieval. The microcontroller had to reach out over UART to the GPS module to retrieve the rover's current GPS coordinates. Once the coordinates were retrieved, the direction heading needed to reach the target location was determined. By calculating the ΔX and ΔY values between the current location and the target location, the angle and direction the rover must rotate was determined.

$$X_2 - X_1 = \Delta X$$
$$Y_2 - Y_1 = \Delta Y$$

Table 8-1 shows how to determine the direction to the target location from the current location. Any locations to the west of the current location will require the rover to turn to the left and any

ΔX	ΔY	Direction to Target
>0	>0	Between North and East
<0	>0	Between North and West
<0	<0	Between South and West
>0	<0	Between South and East

Table 8-1: ΔX and ΔY directions by value

locations to the east of the current location will require the rover to turn right. The angle that the rover must turn can be determined using the following equation

$$\Theta = tan^{-1}(\Delta X / \Delta Y)$$

where a negative value for Θ is indicative of an angle to the left of the current position and a positive value for Θ indicates an angle to the right of the current position. These were used to approximate the command that must be sent to the Sabertooth indicating the amount of rotation that needs to take place. As the rover is rotated in place, checks were continually performed using the magnetometer to verify that the rover was rotating to the proper angle so that it could drive straight towards the target object.

During the majority of the travel time once the proper heading had been determined for reaching the target object, the rover will be traveling at its maximum capable speed. This had to change,

however, the closer the rover was to its target. Once the rover was within a few meters of the target object, as determined by the distance formula, the rover slowed down to allow for fine controlled movement, as well as to allow for other sensors to control the motion, as described previously. Once the rover is within the 2 meter range of its target, GPS was longer be a factor in determining the path moving forwards. GPS was only expected to have an accuracy up to roughly 2 meters, making it useless once the rover had covered the majority of the distance to the target.

8.2.4. Object Retrieval

The object retrieval code was a very specific portion of the overall codebase associated with the rover. The main functionality of this code was to, once the rover had reached the assigned GPS location, search for and retrieve the object.

The first section of the code within the object retrieval routine was to slowly rotate in place while constantly polling the PING))) to check if any objects were detected within the PING)))'s range. If the rover turned a full 360 degrees without locating any objects it would give up and return home, but in the case where an object was located, the rover would then move forward until the object was the exact distance away that the retrieval arm could reach. The arm would then be lowered and the object would be lifted. After lifting the object, the rover would be programmed to return back to its starting location.

8.3. Linux Application Design

8.3.1. Application Requirements

The Linux application (the CDC) serves as the interface between the human users and the robotic subsystems that form SARS. The CDC performs 3 major tasks. First, it initiates, and reports progress on a search and retrieval mission as it progresses. Second, it provides access to the live video stream of the GoPro, so that a mission can be observed in real time. Third, it provides diagnostics and telemetry data from the SARS Copter and the SARS Rover so that mission conductors can monitor each subsystem during a mission. To accomplish these tasks, the CDC employs a simple and intuitive text-based window interface that allows users to easily monitor SARS as it performs a mission. As each subsystem of SARS is physically independent of one another, the application needs to communicate with each subsystem independently. The success of a SARS mission is critically dependent on fast and accurate communication between each system, so the application must be designed to connect and communicate with each system as quickly and efficiently as possible.

Application Requirements

- Initiate SARS missions
- Monitor mission progress
- View camera live stream quickly and easily
- Communicate with other SARS subsystems wirelessly

8.3.2. Operational Features

The following are features that the SARS application must include in order to be fully operational. Without these features, the SARS application will be considered incomplete and incapable of conducting a SARS mission.

Required Features:

- Application runs on a laptop running Ubuntu 14.04 or later.
- Radio based wireless communication with the SARS Copter's Pixhawk flight controller.
- TCP/IP based wireless communication with the Rover's Tiva C microprocessor.
- HTTP/HLS based wireless communication with the Go-Pro camera's web server using 802.11 b/g/n WiFi.
- User commands for initiating missions
- Text based status report of a mission in progress
- Text based report of quadcopter diagnostics and telemetry
- Text based report of rover diagnostics and telemetry

The following are features that are not considered essential to the SARS application's operational status, but are nonetheless deemed important by members of the team. These features will be implemented if time permits, but are otherwise considered entirely optional to the operational status of the SARS application.

Optional Features:

- SARS mission live stream is viewable by multiple concurrent devices
- Aesthetically pleasing user interface with SARS logos and themes throughout the application
- Enhanced graphical display of mission status report
- Enhanced graphical display of quadcopter diagnostics and telemetry
- Enhanced graphical display of rover diagnostics and telemetry

8.3.3. Implementation

The CDC was developed with the Python scripting language. The UI was designed using Qt, which is a cross-platform application framework for designing and programming UI's so that they have the native look and feel of the operating system on which they are currently running. Qt is designed for use with C++, but Python development was made possible through the use of the PyQt framework. PyQt takes a UI form designed with Qt and binds it to Python scripts instead of C++ code. This generated script serves as the backbone on which all other Python libraries, scripts, and functions required by the CDC have been built.
To live stream the GoPro video feed, the decision was made to integrate an existing media player rather attempt to design one from scratch for the sole purpose of integrated live video streaming. LibVLC was chosen as it is free, open-source, capable of HLS streaming, and has bindings for several languages already available, including C, C++, Java, and most importantly, Python. LibVLC is also the media framework on which the popular media player VLC is based – a media player renowned for its stability and wide range of protocol and format support.

The CDC pulls images from the GoPro using an open source Python library called GoProController. This library imports OpenCV to extract individual frames from the GoPro's video feed, which it accesses via the ad-hoc network created by the GoPro. These images are stored in an array before being processed. Once all waypoint images have been stored, the CDC runs a simple blob detection algorithm to identify the large white poster on which the tennis ball will be sitting. Essentially, this algorithm checks each image for 10 consecutive pixels whose RGB values average out above a certain value. This value may be adjusted based upon the intensity of the natural light at the time of the mission.

Communication with the SARS Copter is achieved using the PyMavLink library. PyMavLink is an open source, Python implementation of the MAVLink protocol. In conjunction with the telemetry radio, PyMavLink allows the CDC to wirelessly request almost every parameter stored on the Pixhawk of the SARS Copter. These parameters are then used to display important telemetry data in the telemetry window, and to assist the mission algorithm with determining when the SARS Copter has reached a waypoint.

8.3.4. Design Issues

Reusability
The SARS application is designed for a very specific purpose and has very specific library dependencies so it is unlikely that any of the code will be reusable for future projects, with the exception of the socket programming modules.

Maintainability
SARS is a Senior Design project that was designed, developed, and tested over the course of two semesters. It is unlikely that development on the project will continue that. First, many parts of SARS are funded by sponsors, and thus belong to the University of Central Florida. Upon project completion, these parts must be returned to the university, which will make further development of the system impossible. Second, all members of the SARS team will be graduating upon completion of Senior Design, and will likely be seeking full-time employment. This will limit the available time and willingness to continue development on the project.

Testability
Testability was critical to the success of SARS as a whole. The entire system was continually and thoroughly tested throughout the development process. Thus, it was critically important that the CD be designed with testability in mind. For this purpose, several debug commands were built into the system so that users can test individual sub systems without needing the entire system operational for the application to run. For example, debug commands were included for testing the video feed, the quadcopter radio connection and communication, the rover connection and communication, the image capturing code, and the image processing code.

Performance
SARS needed to have real-time communication performance to be successful. Luckily, the data overhead was not large for the wireless communication between subsystems. The 802.11b/g/n protocol provided more than enough bandwidth and low enough latency to provide real time

communications to and from the application. However, the most troublesome issue for performance was ensuring that the wireless connections did not block other portions of the application from running so that the user interface remained responsive. This was achieved by using non-blocking socket connections to both the SARS Copter and SARS Rover.

Safety
Safety is another critical aspect of SARS. Mission conductors must be able to monitor all aspects of the SARS subsystems in real time to determine if a mission needs to be aborted at any time to prevent damage to the system or to the surroundings. The CDC was initially planned to have the ability to manually terminate a mission, but this feature was unable to be implemented due to time constraints caused by other project issues.

9. Integration Summary

The SARS integration plan was not finalized until fairly late in the development process. All of the relevant wiring diagrams have been created and are included in Section 5: Hardware Design. Because the software being written for SARS has so many dependencies, the SARS Group has decided to use Python to implement as many of the image processing and quadcopter functionalities as possible. The TP-LINK TL-WN722N dedicated Wi-Fi module shall be used to connect whatever device is running the CDC application to the GoPro's ad-hoc network. This connection is necessary for the streaming of the GoPro's video feed as well as for frame extraction. Once all of the frames have been extracted from the GoPro feed and saved as files by the CDC, the SARS image processing algorithm shall be run by the CDC to check each image for the target object.

On the SARS Copter, the Pixhawk will need to be wired to the IMU and the Ublox GPS unit as well as to the power source. Likewise, on the SARS Rover, the Tiva board used as the microcontroller will need to be wired to the CC3100 Wi-Fi module, the Adafruit Ultimate GPS breakout, the motor controllers, the compass, and the Ping)) sensor. The prefabricated rover selected for SARS makes the connection between the microcontroller and the motor controllers incredibly simple.

The Tiva board is not as high level as the Pixhawk. It and all of its functionalities are to be coded in modified C. The CC3100 Wi-Fi module will be used to receive the index of the GPS coordinates of the target object, which will be hard coded into an array, from the CDC application. The SARS Rover will use its GPS module along with the magnetometer in the sensor hub to navigate to the target object's location. Once the SARS Rover reaches the target GPS coordinates, it will search for the target object using ultrasonic sensors. Upon object retrieval, the SARS Rover will return to its starting location once again using the magnetometer and the GPS module.

The method of object retrieval selected for SARS is a pad of hook and loop fasteners mounted on an arm extending from the front of the SARS Rover chassis. The rover arm, originally designed to lower and rise, presented some issues with power distribution on the SARS Rover. The power distribution board on the rover limits the current output to the components, and with the arm drawing power in conjunction with the dedicated Wi-Fi module, there was not enough current to

distribute between the different components. As a result, the SARS Group decided to cut power from the arm and have it remain static.

The remainder of this section will be devoted to listing the parts necessary for the implementation of SARS. A complete list of the parts included in the quadcopter kit is displayed in Figure 4-10: Quad kit components; therefore, these parts shall not be included in the list below.

SARS Copter

- GoPro Hero3 White Edition
- Lithium Powered Battery
- 4 Electronic Speed Controllers
- 3DR Quad Kit
- 950 MHz Radio Receivers
- Turnigy 9x Radio Transmitter
- PPM Encoder
- GoPro Mount Supplies

SARS Rover

- Chassis
- Frame
- 6 Motors
- 6 Wheels
- Sabertooth Motor Controller
- Tiva C Series
- Adafruit Ultimate GPS Breakout
- Magnetometer
- Parallax Ping))) Ultrasonic Distance Sensor
- TI CC3100 SimpleLink
- TI CC31XX EMU Boost
- Rover Power Switch

The only hardware necessary for the implementation of the CDC application is a device running the Linux operating system; however, the TP-LINK TL-WN722N is being used to allow two separate, simultaneous Wi-Fi connections between the CDC and the GoPro and between the CDC and the SARS Rover.

10. Prototype Construction and Software Development

10.1. Quadcopter Parts Acquisition

Most of the components for the quadcopter were provided by 3D Robotics. We decided to purchase a kit that was composed of the frame, propeller motors, Pixhawk flight controller, power module, power distribution board, and ESCs. In addition, there were several more components that were

required to get the quadcopter in the air that were also purchased from 3DR in order to assure consistency and functionality, including the telemetry radios, PPM encoder, battery, and GPS.

Hobby King was used to purchase the Turnigy 9X transmitter/receiver, a decision we went with due to its cheap price compared to other transmitters similar to it, as well as the quick turnaround in shipping (approximately three days).

10.1.1. Camera

The GoPro Hero 3 White was ordered from Amazon at a price of $199.99. It came with a protective, weatherproof case, two adhesive mounts, and a USB for connecting to a computer and for charging. The two adhesive mounts will not be necessary for the final implementation of SARS as the SARS Group decided to build its own quadcopter camera mount out of neoprene from a mouse pad and hook and loop fasteners; however, they may prove useful during the testing of the image processing subsystem. The camera may be mounted on any surface and the target object placed inside its range of view. Preferably, the majority of the testing of this subsystem will be done without the use of the quadcopter. This will reduce the risk of damaging the quadcopter, the components mounted on the quadcopter, or the camera.

The GoPro arrived undamaged in the mail on Friday, October 17, 2014. The parents of one of the SARS Group members elected to pay for the GoPro as they are interested in keeping it once the project has been completed.

10.2. Quadcopter Assembly

10.2.1. Propeller Motors, Power System and ESCs

To begin assembling the quad, first the four propeller motors were mounted to each of the arms. The cables for the motor were threaded through the arm, and a mounting plate was secured at the end to assure the motor did not jostle or vibrate. Next, the arms were each attached to the base plate of the frame, and the propeller motors were wired to the ESCs. The dean connectors were then soldered onto the ends of the ESCs, and these connectors used to attach the ESCs to the power distribution board. Finally, the PDB was connected to the power module that would be hooked up to the battery. After all of these were assembled, they were mounted to the base plate of the quadcopter’s frame so that they were secure. Below is a picture of the final wiring of the power system and controls.

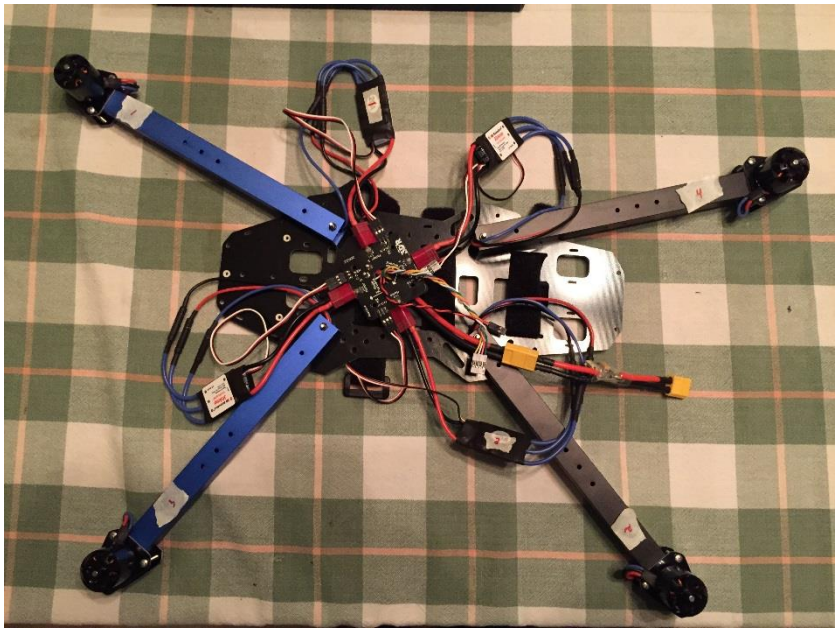

Figure 10-1: Quadcopter Power System

10.2.2. Pixhawk and Peripherals

The Pixhawk is first mounted to its own plate that sits above the top frame plate of the quadcopter. The Pixhawk is mounted on vibration damping foam to reduce noise to the internal IMU. The buzzer and mount switch are wired to the Pixhawk and secured to the frame. The GPS and external magnetometer are connected to a breakout bus that is wired directly to the Pixhawk and communicates over I^2C. After all of the components have been connected, the Pixhawk's plate is mounted to the base plate. Below is a photo of the Pixhawk's plate.

Figure 10-2: Pixhawk plate and finished wiring

After the primary peripherals are wired and the plate has been attached to the frame, the rest of the parts can be wired. The Turnigy receiver is connected to the 3DR PPM encoder, which is then connected to the RC signal pins on the back of the Pixhawk. Finally, the power module is connected to the output signal pins.

10.2.3. Frame and Propellers

After the Pixhawk plate and power system have been wired, attach the quad arms to the bottom plate, and then screw the top plate to the top of the arms and the bottom plate. Then, secure the Pixhawk plate onto the top plate and screw it into place. Finally, use thread glue to secure the propellers onto the motors to assure they don't spin off during flight. Below is a picture of the fully assembled quadcopter.

Figure 10-3: Fully assembled quadcopter

10.3. Camera

The GoPro camera will be mounted underneath the quadcopter using neoprene taken from a mousepad for image stabilization and hook and loop fasteners. For images of this setup, see Figure 10-4: GoPro Mount Bottom View and Figure 10-5: GoPro Mount Side View below.

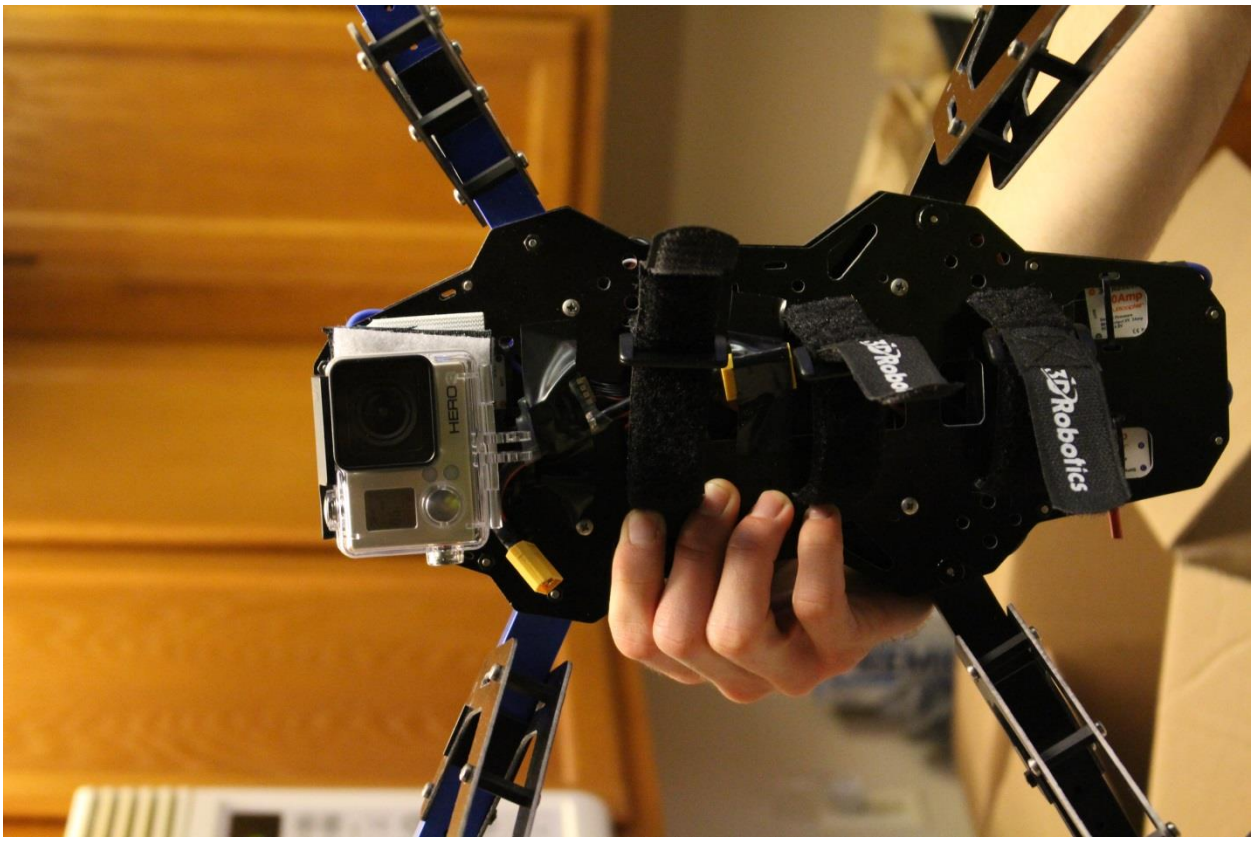

Figure 10-4: GoPro Mount Bottom View

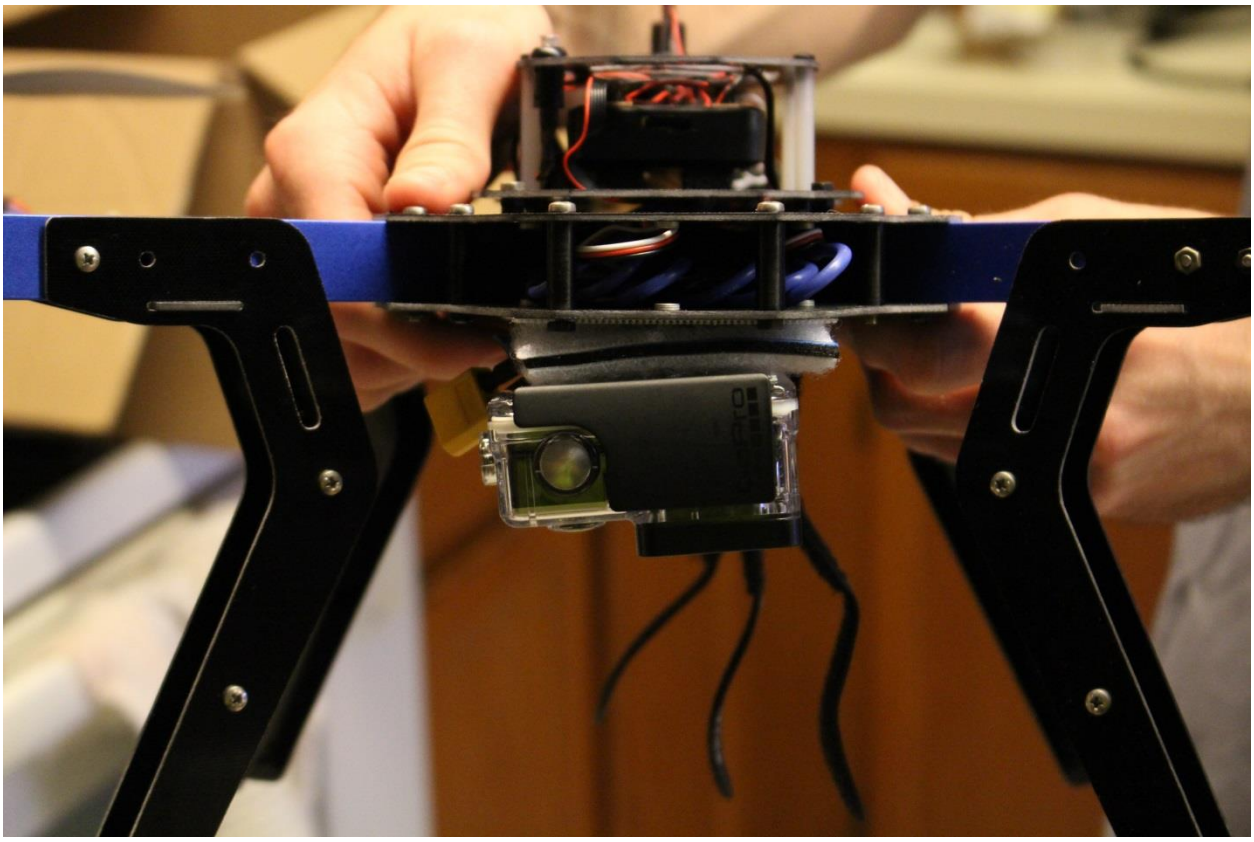

Figure 10-5: GoPro Mount Side View

10.4. Rover Parts Acquisition

10.4.1. Chassis, Motors, and Motor Controller

The chassis selected, the Dagu Wild Thumper 6 Wheel Drive is readily available from various online retailers. It is manufactured by a company called Dagu Electronics, but Dagu Electronics does not offer the product directly at a reasonable price. The chassis, however can be purchased from retailers such as Sparkfun.com, Polou.com, and RobotShop.com for a reasonable price of roughly $250. Fortunately, the chassis is a prefabricated kit, and requires minimal assembly and limited searching for compatible parts. The chassis kit, as a bonus, also includes all six motors required to drive the system. The kit includes these motors already assembled into their respective housings and with the wires already running to an easily accessible location on the chassis. The chassis also includes a preconfigured suspension system to allow for easy terrain traversal. The kit is available with two different colors, chrome and black as well as with two different motor configurations, motors with a 34:1 gear ratio and motors with a 75:1 gear ratio. The chassis with the black body and the 75:1 geared motors was selected. The 34:1 gears allow for higher speeds and a slightly lower load capacity than the 75:1 geared motors. The chassis was acquired by purchasing it through Sparkfun.com.

The other important aspect of the rover system in this category is the motor controller. The selected motor controller, the Dimension Engineering Sabertooth 2x25, is available online directly from the manufacturer, as well as through Robotshop.com and through Amazon.com. This product, however does have a short wait time before it can be shipped, and was ordered early from Amazon.com to ensure timely arrival. It is available for $125 regardless of the purchase source.

10.4.2. Microcontroller

The microcontroller selected is the Tiva C series microcontroller, which is available from Texas Instruments. There are various available development boards in the Tiva C series, but the one being used is the Tiva 1294XX. Texas Instruments distributed free development boards at a workshop that was attended by a SARS team member. This development board was used for the majority of the design and testing process. As the completed prototype deadline approached, this microcontroller was interfaced to a printed circuit board to interface with sensors that could be integrated into a PCB.

10.4.3. Sensors

The magnetometer selected, the Honeywell HMC5883L, is implemented into a variety of breakout boards for testing, and the one selected was the GY-271 board, which was available for less than $10 from Amazon.com. The HMC5883L is included in our PCB design for easy interfacing with the Tiva C.

The Hall Effect sensors and the magnets necessary for them are readily available through online retailers and will be purchased from amazon.com. Both the sensors and the magnets can be purchased for less than $10. The Parallax PING))) ultrasonic sensor is also available online and was also purchased from Amazon.com. The same is true of the Adafruit Ultimate GPS breakout chip which is priced at $45.

10.4.4. Retrieval Apparatus

The majority of the parts needed for the retrieval arm were purchased directly from Adafruit. Adafruit provided the aluminum beams, the servos, as well as most of the mounting hardware. Approximately $70 was spent on the acquisition of these parts. Unfortunately, the hardware purchased from Adafruit wasn't quite enough, and more hardware including elbow mounts and various nuts and washers were purchased from Home Depot as needed.

10.4.5. Communications Module

The communications module, the CC3100, was loaned to the SARS team from the TI Innovation Lab on the UCF main campus for use with the SARS system.

10.4.6. Power Source

The main power source for the rover was a single Lithium Polymer (LiPO) battery pack. When selecting one of these battery packs, there were a huge range of available products. The website hobbyking.com has the best selection of these batteries for the best prices, and this site was used for the majority of the research, but this site was discovered to have excessive shipping costs and poor shipping speed. A battery researched on hobbyking.com was later purchased on Amazon.com for roughly $40.

10.5. Rover Assembly

10.5.1. Chassis, Motors, and Motor Controller

Assembly of the main chassis for the rover was thought to require very little. The chassis came as a kit with the motors pre-installed along with the suspension system. The only assembly required is to mount all six of the wheels onto the motor shafts. The wheels and wheel mounts are all identical, and can be mounted on any of the motors.

Mounting the motor controller will be a slightly more difficult task. The mounting bays within the chassis are roughly .5cm too thin to house the motor controller in it's out of the box form. The walls available space within the chassis had to be slightly bent to allow the motor controller to be housed. The other task in mounting the board was drilling appropriate holes in the chassis to support it. The chassis is designed to support many mounting options because it comes pre-drilled with a full array of available mounting points. Unfortunately, these mounting points did not line up with the available holes on the motor controller for mounting. Once the appropriate holes were

drilled through the chassis, a small screw was used in conjunction with a corresponding nut to secure the motor controller in place. The mounting location is illustrated in Figure 10-6.

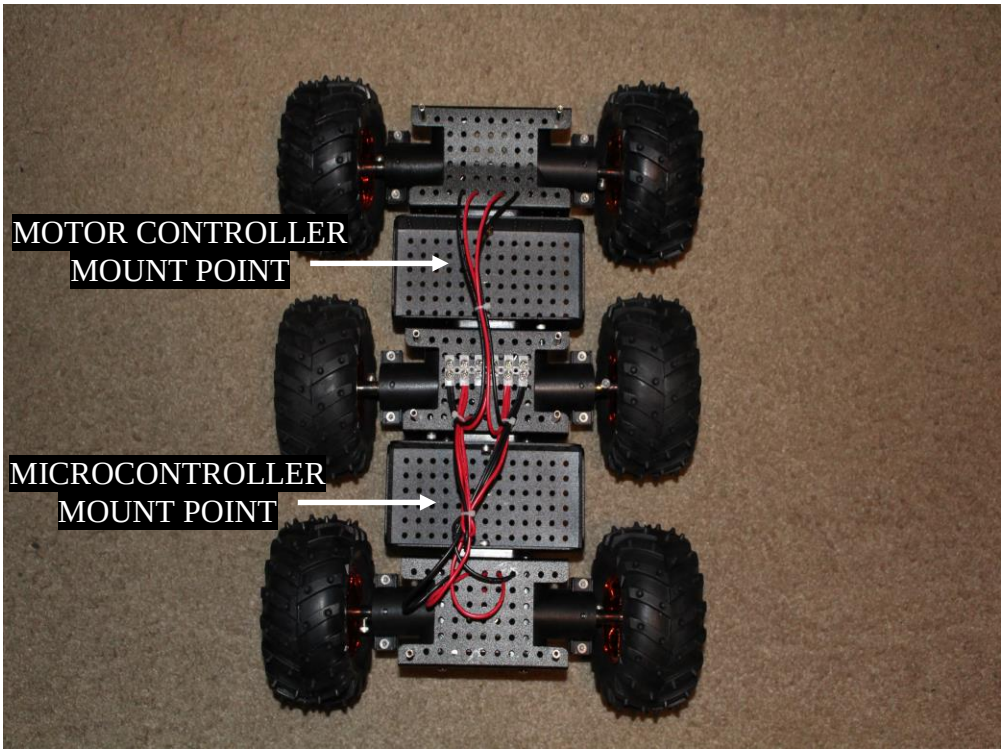

Figure 10-6: Motor Controller and Microcontroller Mount Points

After mounting was complete, the initial wiring took place. Wiring was done as specified in the design section. The motor controller was first connected to the differenct motors. This was done very simply through the central wiring hub on the chassis. This hub was part of the chassis' initial configuration and is visible in Figure 10-7. The wires seen are the wires for the each of the six motors. The motors are separated and connected to the hub based on the motors on the left side of the chassis and the ones on the right side of the chassis. The wires are also separated into positive and negative lines. Due to the way the the wires are adjoined in the hub, the motors for each side of the rover were connected in parallel, allowing for equal voltage across all of them.

10.5.2. Microcontroller, Sensors, and Communication

After the main chassis components are assembled, the electronics were mounted and installed. The first of the electronics to be installed was the microcontroller. The initially selected mounting point for the microcontroller is visible in Figure 10-. In the finished prototype, the microcontroller was interfaced with a PCB. Creating the PCB eliminated the mounting issues presented by the motor controller. With the PCB, the design intentionally left space in the necessary spots on the board so

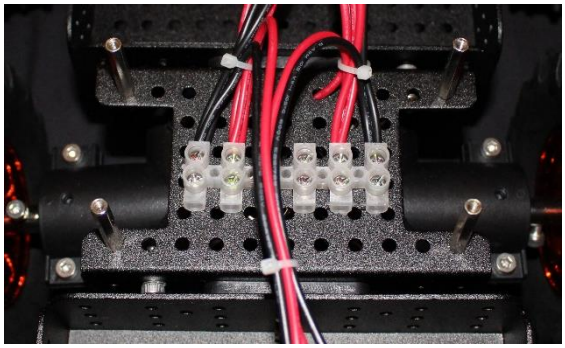

Figure 10-7: Central wiring hub for Rover

that it can be lined up and mounted using the available mounting space already on the chassis. Once the PCB was mounted, it was wired in. Power was supplied by the microcontroller power output on the motor controller. Once power was wired in, the next step was to run the data wires back to the motor controller as well. Wiring schemes are shown in the design section.

Immediately following PCB mounting, the CC3100 was mounted. It was attached directly to the PCB using the booster pack pin headers, which were configured in a manner borrowed from the reference design for the board provided by TI. This held the CC3100 in place and provided all necessary communication pins and power pins.

Now that the PCB was in place and powered, the sensors were addressed. The GPS and magnetometer were included in the PCB, so they were not further addressed during assembly. The sensor that requires the most assembly is the PING))) ultrasonic sensor. The first task was to mount the sensor onto a plate so that it can be placed wherever desired. This was achieved using a combination of aluminum, double sided tape, electrical tape, and rubber bands. Once the sensor was mounted to a frame, the frame was mounted protruding from the front of the chassis to allow the sensor maximum coverage of the area in front of the rover. The sensor was wired to the PCB for data and power.

The Hall Effect sensors were the last to be implemented. They were mounted proximally to the driveshaft for each motor using a two part epoxy. Each driveshaft was then be outfitted with a small magnet also using the two part epoxy. Each of the six sensors was wired to the GPIO pins available on the PCB.

10.5.3. Retrieval Apparatus

The building and attaching of the retrieval arm was the longest process within the assembly stage. The first step was to construct the physical arm. This involved cutting the aluminum beam into smaller pieces from the full size beam to make it into a 90 degree arm and so that it was a reasonable length to not apply too much torque to the servos. Once the beam was cut, two pieces were joined using an elbow to form the 90 degree angle.

After the main part of the arm was assembled, the next task was to attach the arm to the servos. The servos included various horns, and those were modified to attach directly to the beam used to construct the arm. Once the horns were modified, the arm was easily slid down onto them for secure mounting.

The end of the arm was then fitted with the Velcro sheet for object retrieval. This was done using aluminum and simple elbows, and was held in place with a few nuts and bolts.

The last step was mounting the servos and the arm onto the rover chassis. This was also achieved using 90 degree elbows, but these were larger than the ones used for joining other parts. The servos were mounted so that the mounting holes were parallel to the ground when the rover is upright.

10.5.4. Power Source

The final, and perhaps most crucial element of the Rover's assembly was properly attaching the battery. As discussed in the design section, the battery needed to be easy to connect and disconnect as well as install and remove. While keeping these requirements in mind, the battery must also had to be extremely secure so that movement of the Rover did not cause the battery to get displaced or jostled loose and also does not cause wires to disconnect.

Battery mounting began by first installing the necessary screws to hold the battery in place. After the screws were located properly and cushioned to not damage the battery, the battery was placed and secured using rubber bands. Despite the simplicity of this mounting solution, the battery held in place without issues and was easy to maneuver as needed.

Once the battery has been mounted, connecting it was the next priority. Wiring was run from the battery leads to the appropriate terminals on the power switch. Before running these wires, it was vital that the power switch be in the "off" positon to avoid sparking and unnecessary danger. Once the battery had been connected to the power switch, wire was run from the power switch to the main wiring hub of the rover. Once this was completed, all further power wiring was completed from the hub.

10.6. Android Application Development

SARS was initially designed to be used with an Android application component. An early prototype of the user interface was designed in Android Studio, however little work beyond the initial design was done and the Android component was ultimately abandoned in favor of a Linux

application. The image below in FIGURE# illustrates one of the initial UI prototypes for the Android application.

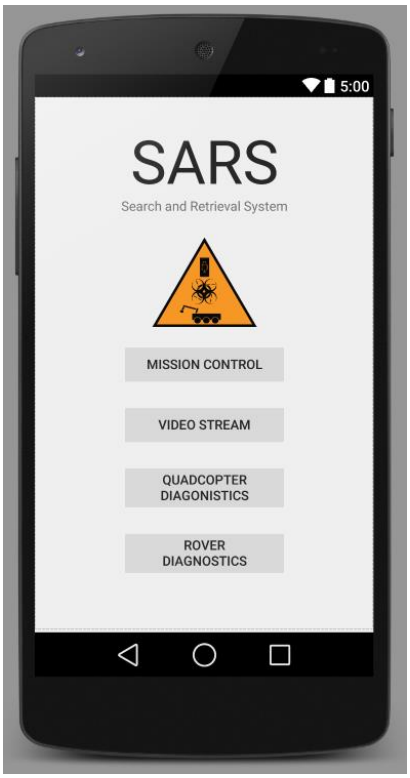

Figure 10-6: Android Application Prototype

10.7. Linux Application Development

The Command Distribution Center (CDC) is the Linux-based application which coordinates the operations of both the SARS Copter and SARS Rover. As the SARS Copter traverses a series of mission waypoints, the CDC pulls still images from the GoPro's live video feed via Wi-Fi and saves them in an indexed array. Once the SARS Copter has completed its mission and all the images have been captured, a blob detection algorithm runs on the array to check each image for the target object. If the object is detected, the index of the image is cross referenced with a lookup table containing the GPS coordinate of each waypoint in the mission. This GPS coordinate is then transmitted wirelessly via Wi-Fi to the microcontroller mounted on the SARS Rover.

In addition to handling both wireless communication and coordination of the two unmanned vehicles, the CDC acts as a user interface for the entire system. It is composed of 4 main interface components: the live video stream window, the telemetry window, the mission status window, and the command console. FIGURE# below provides a screen capture of a late prototype of the CDC for reference. This prototype is very close to the final CDC application, except for a few minor UI tweaks.

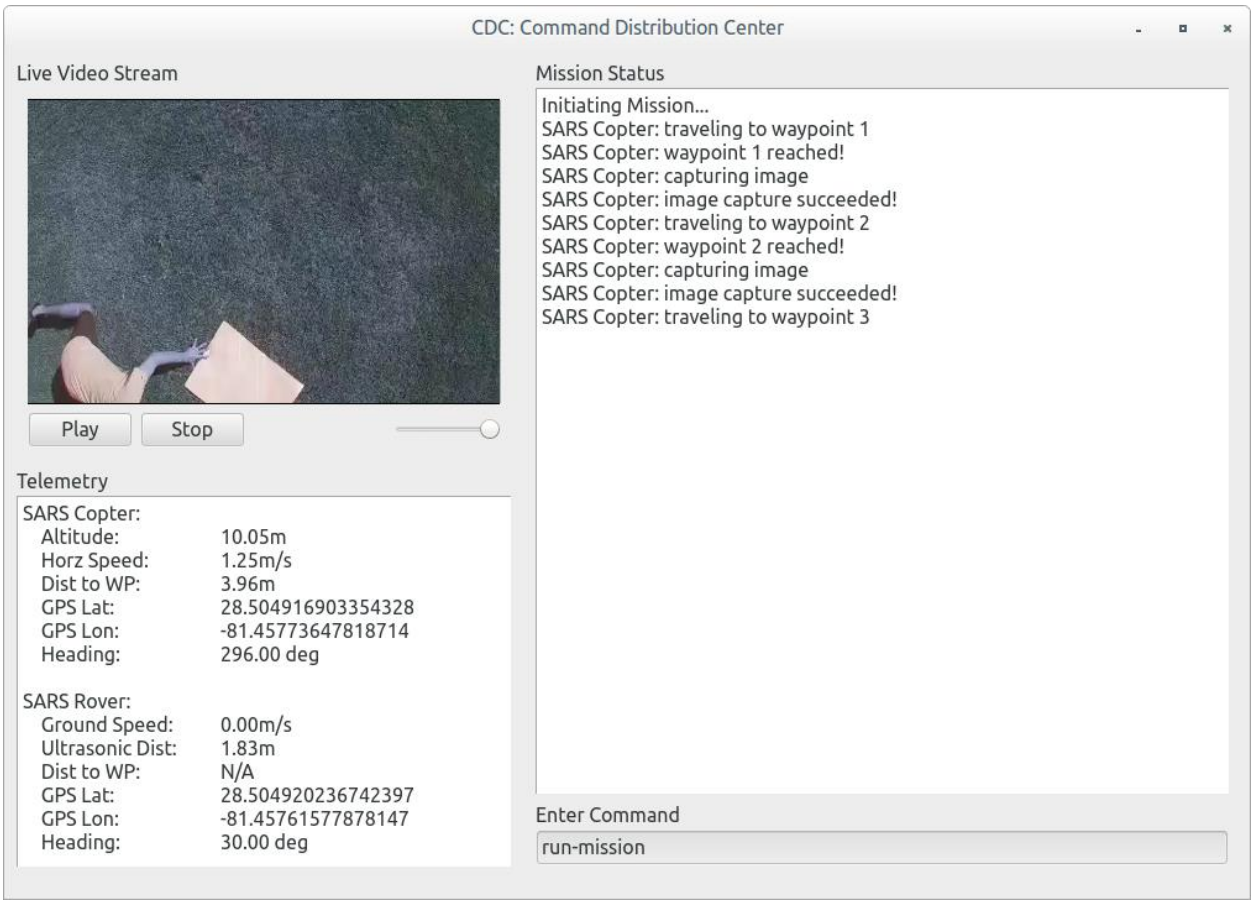

Figure 10-7:The CDC: Command Distribution Center

10.7.1. Live Video Stream

The live video stream window utilizes the LibVLC media framework to connect to the GoPro's HTTP server and stream video utilizing the HLS (HTTP Live Streaming) protocol. Play and Stop buttons are included so that the video feed can be started or stopped at any time. This live video feed provides the user with a real time, first person view of the SARS Copter as it performs its mission.

10.7.2. Telemetry

The telemetry window utilizes the RC telemetry radio that communicates with the SARS Copter to retrieve telemetry such as altitude, horizontal speed, distance to next waypoint, GPS coordinates, and current heading. It also utilizes the Wi-Fi connection to the SARS Rover to retrieve telemetry such as ground speed, distance to target object, ultrasonic sensor distance data, GPS coordinates, and heading. This information is periodically requested from the vehicles and automatically displayed in the telemetry window so that the user can monitor the status of the vehicles as the mission progresses.

10.7.3. Mission Status

The mission status window reports the progress of a mission as the mission algorithm executes. It reports information such as the SARS Copter's current mission progress, the success or failure of image captures, the index of the image containing the detected object and the corresponding GPS coordinate, and the SARS Rover's progress as it travels to the target object. The mission status window gives the user an overview of a mission so they can determine if the mission can proceed or if it must be aborted at any time.

10.7.4. Command Console

The command console is a simple one-line text box that allows the user to issue commands to the CDC. Commands for both initiating and aborting missions are recognized. Various debugging commands are also included so that specific aspects of the mission algorithm can be tested separately, such as capturing an image from the GoPro or running the object detection on captured images.

11. Prototype Testing

11.1. Hardware Test Environments

11.1.1. Quadcopter

The quadcopter will initially be tested indoors until it is capable of stable, sustained flights. Once this has been accomplished, testing will be moved outdoors. Typical outdoor test environments will include consist of sunny, clear skies with moderate temperature and climate. The quadcopter will not be tested during excessively windy conditions or other inclement weather. As this is the first SARS prototype, the quadcopter will not be weatherproofed so the team does not want to risk weather damage as the quadcopter is the single most expensive part of SARS.

11.1.2. Rover

All rover hardware testing will take place outdoors unless. Unless specified by one of the tests cases, testing will take place on a flat concrete surface, such as an empty parking lot. Incline tests will be performed in a parking garage and other rough terrain tests will be executed in any available places described in the specific test procedure. One important aspect of testing is verifying that the battery is at 50% charge for each hardware test. This will ensure that the rover is operating under ideal power.

11.2. Hardware Test Cases

11.2.1. Quadcopter

11.2.1.1. Serial Communication Test Interface

Purpose: Validate that the BeagleBone Black MCU is able to communicate to the Pixhawk flight controller over the I^2C serial communication interface.

Procedure:

1. Connect Pixhawk to laptop via micro USB cable and power on.
2. Connect BeagleBone Black to laptop via micro USB cable and power on
3. Verify correct pin connections between BeagleBone Black and I^2C bus
    a. Serial Clock Line to P9_17
    b. Serial Data Line to P9_18
4. Verify Pixhawk connected to I^2C splitter
5. On BeagleBone, run script to verify data bus can be accessed
6. Confirm connection by running iotcl() command on BeagleBone
7. Send Return_to_Launch command to Pixhawk. Confirm that the message was received by the flight controller.
8. From the Pixhawk, send the current coordinates to the BeagleBone Black over the serial interface.
9. Run the block of code that reads from the I^2C bus on the BeagleBone MCU.
10. Verify that the coordinates sent and received are identical

Expected Result: Two-way communication interface works correctly. BeagleBone Black is able to send flight mode interrupt to the Pixhawk and the flight controller is able to respond with the current location coordinates.

Conditional Requirements:

- No timeouts between two MCUs
- Data transmitted in under 1 second

11.2.1.2. Microcontroller Power Testing

Purpose: Validate that power is being distributed to the BeagleBone Black correctly and that it is able to run on batteries for the entire duration of the mission time

Supplies:

- BeagleBone Black
- Power Source
    - Voltage Regulator
    - OMRON A3DT-7111 push-button switch
    - OMRON A3DT-500GY LED
    - 6 AA Batteries
    - Battery holder

Procedure:

1. Verify power source is connected to BeagleBone Black via design specifications in IDD
2. Press the push-button switch to deliver power to the board
3. Confirm LED is beating in a heartbeat pattern to validate board is powered correctly

Expected Result: LED blinking in heartbeat pattern

11.2.1.3. Power Distribution Testing

Purpose: Verify power is distributed to correctly to the Pixhawk flight controller as well as the four ESC's that power the propellers.

Supplies:

- LiPo Battery
- 3DR Power Module
- 4 ESC's and propellers
- Pixhawk flight controller
- Associated cabling and connetors
- Laptop (with Mission Planner)

Procedure:

1. Connect LiPo battery to 3DR power module
2. Connect 3DR Power Module to Pixhawk flight controller via 6-pin DF13 conncetion cable.
3. Connect 3DR power module to ESC's to the Pixhawk by securing the power (+), ground (-) and signal (s) wires to the main output pins on the Pixhawk
4. Power up copter propellers and verify they can run at 3000 rotations-per-minute (RPM) (minimum requirement) to 10,000 RPM (maximum requirement)
5. Run a simple guided flight mission simulation on the Mission Planner and confirm it runs successfully.
6. Cross-reference with test flight data logs

Expected Result: Power correctly distributed to all components of the copter

11.2.1.4. ESC Calibration Testing

Purpose: Verify that the ESC's are calibrated correctly so that the quadcopter is able to fly with stability and accuracy and the propellers spin at the correct speeds

Supplies:

- LiPo Battery
- 3DR Power Module
- RC Transmitter Controller
- Pixhawk Flight Controller
- 4 ESC's
- Laptop (with Mission Planner)

Procedure:

1. Plug in battery to 3DR power module and secure connections from PM to all four ESC's
2. Set transmitter flight mode to "Stabilize"
3. Wait 30 seconds for GPS lock to settle. On the Pixhawk flight controller, the RGB LED light will turn green.
4. Hold the throttle down and rudder to the right for 5 seconds. Confirm that the red arming light turns solid.
5. After the light turns red, give the copter a small amount of throttle again.
6. Disarm copter

Expected Results: When giving the copter a little throttle in Step 5, the motors should all begin spinning at the same time and continue spinning at the exact same speed. If this is not achieved, re-arm the copter (Steps 2-4) and try again.

11.2.2. Rover

11.2.2.1. Stationary Power Test

Purpose: Verify that all Rover subsystems are receiving the power necessary for operation and basic load testing of motors and servos on the battery.

Procedure:

1. Verify all wiring is securely in place with no loose wires or uncapped live wires
2. Use power switch to power on the system
3. Allow 30 seconds of idle operation, if no issues, continue
4. Issue start command to microcontroller
5. Verify motor functionality
6. Issue continue command to microcontroller
7. Verify functionality of all other subsystems
8. Turn off power

Expected Results: All powered systems will function as specified in the Rover Design section.

Conditional Requirements:

- Power switch will allow current to flow from the battery to the motor controller powering it on, and subsequently powering on all other subsystems
- All subsystems power on into proper functioning mode (no failed power on attempts)
- Major powered subsystems can operate in unison without overloading circuits
  - Motors operate at full speed
  - Arm operates in specified range
  - Control boards power on and able to execute orders

11.2.2.2. Straight Line Movement Test

Purpose: Verify that the Rover can move forward in a straight line over basic terrain

Procedure:
1. Power on the Rover and issue start command
2. Verify movement along straight line
3. Power off the Rover

Expected Results: Rover moves in a straight line for specified amount of time

Conditional Requirements:
- Power source must be able to handle movement while under load of the Rover's own weight
- Motors must be functioning properly
- Motor controller must be distributing power correctly to all of the motors
- No mechanical or electrical failures

11.2.2.3. Advanced Movement Test

Purpose: Verify motor controller functionality to allow for more advanced movements including stationary turning and turning while in motion

Procedure:
1. Power on the rover and issue start command
2. Verify forward movement along a straight line
3. Verify backward movement along a straight line
4. Verify 90 degree left turn (stationary)
5. Verify 180 degree left turn (stationary)
6. Verify 90 degree right turn (stationary)
7. Verify 180 degree right turn (stationary)
8. Verify moving left turn
9. Verify moving right turn
10. Verify stop

Expected Results: All tests listed in the procedure are completed

Conditional Requirements:
- Motor controller commands operate as specified
- All motors operating as designed
- No mechanical or electrical failures

11.2.2.4. GPS Movement Test

Purpose: It must be verified that the GPS module and magnetometer for the Rover are functioning properly. This test will ensure that the Rover is capable of navigating to a specified GPS coordinate location.

Procedure:

1. Power on Rover system and start software control loop
2. Input test target GPS location
3. Issue command for Rover to start navigation to target

Expected Results: Rover successfully turns and faces the correct direction and moves forward until reaching the specified coordinates. The Rover will then stop.

Conditional Requirements:

- Magnetometer must relay correct orientation based on Earth's magnetic field (proper function)
- GPS module must receive accurate location data from satellites (proper function)

11.2.2.5. Rough Terrain Movement Test

Purpose: Verify the rover's ability to traverse difficult terrain

Procedure:

1. Place the rover near a location with loose ground (slip correction test)
2. Power on the system and issue the start command
3. Verify the rover's ability to cross this land
4. Place the rover near a location with uneven ground (suspension assisted navigation test)
5. Verify the rover's ability to cross this land
6. Bring the rover to a location with an incline of at least 30 degrees (incline test)
7. Verify the rover's ability to ascend the incline
8. Verify the rover's ability to descend the incline

Expected Results: Rover will successfully traverse all presented terrain challenges

Conditional Requirements:

- All wheels properly secured to motors
- Weight distribution does not cause tipping or a specific wheel/group of wheels to become inoperable
- Suspension functions properly under system weight

11.2.2.6. Retrieval Apparatus Test

Purpose: Verify grabber's ability to pick up an object

Procedure:

1. Place the rover within 2 meters of target object (assuming success of GPS navigation to object)
2. Verify rover positions itself in place to pick up object
3. Verify arm moves to proper position to lower onto object
4. Verify arm lowers directly onto object and grabber activates

5. Verify grabber successfully grasps the object
6. Verify arm moves to position object over storage bin
7. Verify grabber releases and arm returns to neutral position

Expected Results: Object will be successfully deposited in storage on rover

Conditional Requirements:

- Rover capable of executing fine movement adjustments
- Arm moves as expected
- Arm and servos able to withstand torque applied by object
- Grabber successfully grasps the object and is able to hold on long enough for deposit in storage bin

11.3. Software Test Environments

11.3.1. Quadcopter

11.3.1.1. Image Processing Test Environment

The image processing subsystem was tested on a laptop running the Linux operating system. The complete list of materials for this test is as follows: laptop running Linux, GoPro Hero3 White Edition. The application retrieving the frames from the GoPro camera's live feed and processing these images was written in Python, but it cannot run on a Windows machine because Windows does not have access to all of the necessary libraries. Specifically, Windows does not support the dbus library.

Initially, tests were performed indoors. Positive and negative sample images of a white poster board were supplied for testing. These first tests were not extensive. For example, the poster board simply needed to be detected while sitting alone on the floor. Once the script had been verified to detect the object indoors, the tests were moved outside. The object detection was tested with the poster board in the grass as well as on asphalt. Once these tests generate satisfactory results, the subsystem will be tested from the quadcopter as it hovers about 10 feet in the air, and, finally, once the subsystem is ready for integration, it will be tested with the rest of the SARS subsystems.

Testing for this subsystem will began in mid to late November 2014 and continued until integration in February and March of 2015.

11.3.1.2. Geolocation Subsystem Test Environment

The testing of the Geolocation Subsystem was performed using the Pixhawk flight controller. The list of parts involved in this subsystem is as follows: Pixhawk, U-blox LEA-6H GPS module, Linux device running the CDC. Tests for the Geolocation Subsystem were conducted in an outdoor setting, where the GPS module would most easily receive satellite signals. Initial testing was done on the ground, and to reduce the risk of damaging hardware, aerial testing was only conducted

once integration was ready to take place after the subsystem functions were verified and validated. The target start date for the testing of the Geolocation Subsystem was in mid-January.

11.3.2. Rover

The software testing process for the rover involves all of the necessary code sequences being prepared before each individual test begins. All software specified in the software design section should be completed so that any necessary functions can be readily accessed for testing. The key to each of the tests will be modifying the main control loop to execute the operations for the required tests.

Initial testing will involve speed and direction control. Each control program will be injected to the microcontroller and will then be executed in the environment specified in the hardware environment. Effectiveness of testing will be visually verified by the movements of the rover.

The next batch of testing will involve sensor testing, and can be completed while the rover is stationary or close to stationary and the microcontroller will be connected to a computer to keep the debug channel open while tests are performed. This will allow for verification of anticipated test data.

Once these aspects of testing is completed, the final rounds of testing will all be completed in the hardware test environment and any necessary data monitoring will be performed by transmitting the data over the wireless communication channels.

11.4. Software Test Cases

11.4.1. Quadcopter

11.4.1.1. Waypoint Navigation Testing

Purpose: Validate that when a series of waypoints are fed into the quadcopter, the copter is able to navigate to each of those points with accuracy within the tolerance range.

Supplies:

- Laptop (with Mission Planner installed)
- Fully-assembled Quadcopter (with Pixhawk and u-Blox GPS)
- Micro-USB cable

Procedure:

1. Connect the Pixhawk flight controller to the laptop with Mission Planner loaded on It via the micro-USB cable
2. On Mission Planner, click “Connect” in the upper right-hand corner and verify the connection was successful
3. Arm the motors (Test Procedure X.X)

4. In the “Flight Plan” tab, set the takeoff point to the current location
5. Add three waypoints to the flight plan, with each point exactly 15 meters from the home location (this can be done by right-clicking the point and selecting “Measure Distance”). Keep the altitudes of each waypoint consistent. Record the expected latitudes and longitudes of each point.
6. After each waypoint, include a “Loiter_Time” command, instructing the copter to hover over its position for 10000 milliseconds (10 seconds).
7. Execute the mission. At each waypoint, record the latitude and longitude displayed in the Flight Statistics tab, noting the error between expected and actual results. Also at each waypoint, place a small flag and measure the distance from the takeoff point, recording the actual distance vs. the expected difference (15 m).
8. Compare the results

Expected Result: Copter correctly navigates to all of the programmed waypoints

Conditional Requirements:

- Latitude and Longitude accurate to within ±1°
- Distance accurate to within ±1 meter

11.4.1.2. Pixhawk Compass Calibration Test

Purpose: Test to make sure that the compass offsets are within the appropriate range in order to reduce error in waypoint navigation

Supplies:

- Pixhawk Flight Controller
- Laptop with Mission Planner installed
- Micro USB Cable or Telemetry Radio

Procedure:

1. Open up Mission Planner
2. Connect to Pixhawk using Micro USB cable or telemetry radio
3. Navigate to ‘Mandatory Hardware’ under ‘Initial Setup’ tab
4. Begin ‘Compass Calibration Test’

Figure 11-1: BeagleBone Black reference LED locations

5. Rotate the Quadcopter around all axes, making sure to hit every plot point in the 3D model

Expected Result: All offsets are below 250

Conditional Requirements:
After calibrating the compass, arm the copter and takeoff manually, then switch to Alt Hold mode. If the quadcopter remains level and stabilized, it passes.

11.4.1.3. Pixhawk Firmware Test

Purpose: Verify Pixhawk firmware is loaded correctly on the Pixhawk

Procedure:

1. Confirm Mission Planner has been installed correctly on the computer.
2. Connect the Pixhawk flight controller to the laptop with a micro USB cable.
3. Verify Mission Planner drivers installed correctly.
4. In Mission Planner, select the drop-down menu in the top right-hand corner and select "PX4 FMU" with a baud rate of 115200.
5. On the "Install Firmware" tab of Mission Planner, select "Arducopter V3.0.1 Quad"
6. Go to the Flight Data screen and slowly tilt the quadcopter.

Expected Result: The firmware is correctly installed on the Pixhawk and all readings are accurate.

Conditional Requirements:

- In the bottom right, Mission Planner should display "Upload done" after the firmware is loaded
- In the Flight Data screen, the flight statistics displaying pitch, yaw, etc. should update on the Heads-Up Display as the quadcopter is tilted.

11.4.2. Rover

11.4.2.1. Speed Control Test

Purpose: Verify that the software designed decide on a speed for the rover and transmit that speed to the motor controller are functioning properly.

Procedure:

1. Power up the system and begin the testing code on the microcontroller
2. Verify 25% movement speed
3. Verify 50% movement speed
4. Verify 75% movement speed
5. Verify 100% movement speed

Expected Results: The rover will travel in a straight line at measurably different speeds

Conditional Requirements:

- Communications with the motor controller from the microcontroller will be successful
- Commands being issued from the microcontroller are encoded as expected
- Values calculated to be sent to motor controller are calculated correctly

11.4.2.2. Direction Control Test

See *Advanced Movement Test*

Additional Conditional Requirements:

- Correct commands are being issued from the microcontroller to the motor controller
- Correct values are being calculated to create directional movement
- Left and right motor channel commands are being separated appropriately

11.4.2.3. Echolocation Sensor Test

Purpose: Verify that echolocation sensor is functioning properly and results are being calculated appropriately

Procedure:

1. Power up the system and begin the testing code on the microcontroller
2. Place object at .1m from the sensor and verify detection
3. Move object to .5m from the sensor and verify detection
4. Move object to 1m from the sensor and verify detection
5. Move object to 2m from the sensor and verify detection
6. Move object to 3m from the sensor and verify detection
7. Verify servo and sensor rotation to maximum angle (120 degrees) left of straight ahead
8. Verify servo and sensor rotation to maximum angle (120 degrees) right of straight ahead

Expected Results: PING))) module will correctly locate objects at .1m, .5m, 1m, 2m, and 3m away and the servo will rotate as specified

Conditional Requirements:

- PING))) is properly connected to the microcontroller
- PING))) is functioning as specified
- Microcontroller is issuing correct signals to PING)))
- Microcontroller is correctly interpreting results returned from PING))) after it senses something

11.4.2.4. Accelerometer and Gyroscope Sensor Test

Purpose: Verify functionality of accelerometer and gyroscope sensors

Procedure:

1. Power up the system and begin the testing code on the microcontroller
2. While holding the system and without rotating it, slowly move it forwards and then backwards
3. Slowly move the system to the left and then to the right
4. Slowly move the system up and then down
5. Verify output results for accelerometer
6. While holding the system in place, slowly rotate the system forwards and then backwards
7. Slowly tilt the system with a clockwise roll and then with a counterclockwise roll

8. With the system flat, slowly rotate it clockwise and counterclockwise within the plane it is on
9. Verify output results for gyroscope

Expected Results: The microcontroller will record expected results for all six tested degrees of freedom

Conditional Requirements:
- Accelerometer and gyroscope are functioning properly
- Microcontroller is correctly connected to accelerometer and gyroscope
- Microcontroller is correctly interpreting data received from accelerometer and gyroscope

11.4.2.5. Hall Effect Sensor Test

Purpose: Verify that the Hall Effect sensors are properly connected, properly detecting magnetic fields, and data is being handled properly by the microcontroller

Procedure:
1. Power up the system and begin the testing code on the microcontroller
2. Pass a magnet by sensor number 1 and verify response on microcontroller
3. Pass a magnet back and forth past sensor number 1 and verify microcontroller response
4. Repeat steps 2 and 3 for sensors numbered 2 through 6.

Expected Results: Each Hall Effect sensor will function as it is designed to and the microcontroller will properly interpret data from each sensor

Conditional Requirements:
- All 6 Hall Effect sensors are in proper working order
- All 6 Hall Effect sensors are properly connected to the microcontroller
- Microcontroller is properly interpreting results from each of the Hall Effect sensors.

11.4.2.6. GPS and Magnetometer Test

See *GPS Movement Test*

Additional Conditional Requirements:
- Magnetometer is correctly connected to the microcontroller
- Microcontroller is properly interpreting magnetometer inputs
- GPS module is properly connected to the microcontroller
- Microcontroller is properly interpreting GPS inputs
- Angle and direction calculations are correctly performed by microcontroller
- Commands to turn the rover are functioning properly (See *Direction Control Test*)
- Stop command is issued by microcontroller when destination is reached

11.4.2.7. Rough Terrain Movement Correction Test

See *Rough Terrain Movement Test*

Additional Conditional Requirements:

- Microcontroller recognizes slippage based on information from Hall Effect sensors and properly corrects for the issue
- Microcontroller recognizes the rover is stuck based on accelerometer/gyroscope input and properly corrects
- Microcontroller handles necessary speed adjustments for ascending/descending inclines*11.2.2.5*

11.4.2.8. Retrieval Apparatus Test

See *Retrieval Apparatus Test*

Additional Conditional Requirements:

- Microcontroller successfully navigates from a short distance to the target object
- Microcontroller properly issues commands to manipulate arm servos to move the arm into place for pickup, drop off, and return to neutral
- Microcontroller issues commands for grabbing action and grasp release correctly

11.4.2.9. Object Avoidance Test

Purpose: Verify that sensor data is appropriately employed to successfully navigate around obstacles in the rover's path

Procedure:

1. Power on the rover and issue a start command with a target location separated from the rover's current position by 3 obstacles as seen in FIGURE 8-1
2. Verify successful avoidance of obstacles 1-3
3. Verify rover stops at target location

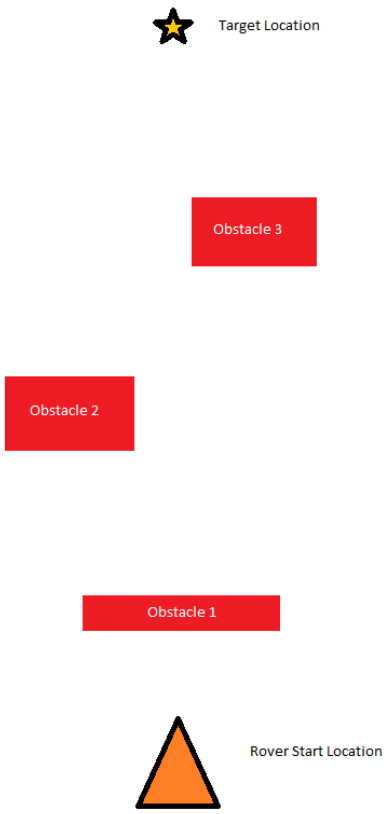

Figure 11-2: Obstacle layout for Object Detection/Avoidance Test

Expected Results: Rover will successfully navigate to specified location without colliding with any obstacles in its path

Conditional Requirements:

- Full function of the PING))) module as described in *Echolocation Sensor Test*
- Microcontroller issues necessary movement commands to move around obstacles
- Microcontroller maintains target location despite subroutines to avoid obstacles
- Microcontroller recognizes orientation of obstacles compared to rover system

11.4.3. Image Processing Subsystem Test

This section contains all the tests which were performed on the SARS image processing subsystem. It contains information on where and how these tests were performed. The section also defines the passing criteria for each test.

11.4.3.1. Testing the GoProController Python Script

Purpose: To access the GoPro video feed over the camera's ad hoc network using a Linux device and to verify that a frame may be extracted from the feed.

Materials:

- Device running Linux
- GoPro Hero3 White Edition

Test Procedure 1: The writer of the open source project has supplied a clearly defined installation and testing procedure.

To install the prerequisites on the BeagleBone Black, run the following command:

```
sudo apt-get install python-numpy python-opencv git
```

To clone the repo, run the following command:

```
git clone https://github.com/joshvillbrandt/GoProController.git
```

To test the script, run the following Python commands:

```
from GoProController import GoProController
c = GoProController()
c.test()
```

Expected Results: The result of this test should be that the script will print the status of the camera. If the device is unable to connect to the camera, an error shall be thrown. This test cannot pass until the connection is made and the status is printed successfully.

Results: The provided script resulted in a successful connection with the GoPro. The success status was printed to the device screen.

Test Procedure 2: To verify that the script may extract a frame from the GoPro video feed, run the following Python command:

```
c.getImage(<ssid>, <password>)
```

Expected Results: The result of this test should be that a .png file will be created and stored on the device running the command. Furthermore, the following message should be printed to the screen: "getImage(<ssid>) – success!". If, instead, the log contains the message, "getImage(<ssid>) – failure", then the test has failed. Before passing the subsystem on this test, open the image file and verify that it is an image from the video feed.

Results: The above command resulted in a successful image pull and printed the proper message to the screen.

11.4.3.2. Testing the Object Detection

Purpose: To verify that SARS object detection algorithm can detect a bright white poster board in the frames extracted from the GoPro video feed.

Materials:

- GoPro Hero3 White Edition
- Device running Linux
- SARS Copter

Test Procedure 1: Run the homemade object detection script on images of the poster board while it is indoors. To do this run the following series of commands:

```
from Detector import Detector
d = Detector()
print(d.detect(<path_filename>))
```

Expected Results: This test should yield no less than a 99% detection rate and no more than a 1% false positive rate.

Results: This test yielded a 99% detection rate and less than a 1% false positive rate.

Test Procedure 2: Run the homemade object detection script on images taken of the poster board while it is outside. Perform this test on both grass and asphalt. Use the same commands given for Test Procedure 1.

Expected Results: This test should yield no less than a 99% detection rate and no more than a 1% false positive rate.

Results: This test yielded a 100% detection rate and a 1% false positive rate for images taken on grass. The results for images taken on asphalt were not as favorable; therefore, it was decided to perform the final demo of the system on a grass surface.

Test Procedure 3: Run the homemade object detection script on images taken of the poster board from the GoPro mounted on the SARS Copter, hovering approximately 10 feet in the air.

Expected Results: This test should yield no less than a 99% detection rate and no more than a 1% false positive rate.

Results: This test yielded a 100% detection rate and a 0% false positive rate for images taken in broad daylight on a grass background.

11.4.3.3. Image Processing Subsystem Integration Test

Purpose: To verify the function of the subsystem once it has been integrated first with the geolocation subsystem, and finally with all of the SARS subsystems.

Materials: See the list of parts in Section 7 Integration Summary.

Test Procedure 1: This test involves the integration of the image processing subsystem with the geolocation subsystem. The SARS Copter shall be traveling through a preprogrammed mission, pausing at waypoints. It is important that the CDC pulls images while the SARS Copter is hovering in place over a waypoint. Finally, if the white poster board has been placed at one of the waypoints, the CDC should detect it and identify which waypoint it is located at.

Expected Results: The SARS Copter should travel to a series of waypoints, pause at each one, and during the pause, the CDC should extract a frame from the GoPro's video feed.

Results: The test was a success; an image was pulled successfully at each SARS Copter waypoint, and the object was detected successfully.

Test Procedure 2: This test of the subsystem involves the integration of the image processing subsystem with the entirety of SARS. The same testing procedure shall be followed as in the first image processing integration tests. This test shall be conducted after all subsystems have been thoroughly tested.

Expected Results: The result of this test should be that the SARS Rover retrieves the tennis ball after the quadcopter has relayed its position, and the SARS rover and the quadcopter should both return to their respective starting positions. A success rate of no less than 95% should be achieved.

Conditional Requirements: These tests must be conducted outside, as they require the flight of the SARS Copter and accurate receipt of GPS coordinates via satellite.

Results: As the entire system was not working properly until the night before the demonstration, integration testing was not performed extensively.

11.4.4. Geolocation Subsystem Test

This section contains all the tests which shall be performed on the SARS geolocation subsystem. It contains information on where and how these tests shall be performed. The section also defines the passing criteria for each test.

11.4.4.1. Testing the SARS Copter GPS Navigation

Purpose: To verify that the SARS Copter can navigate to a series of GPS coordinates.

Materials:

- SARS Copter
- Device running the Mission Planner application

Test Procedure: Load a series of GPS waypoints into the Pixhawk flight controller, arm the SARS Copter, take off manually, and switch the SARS Copter to Auto mode. Once the test is complete, return the SARS Copter to Stabilize mode and land it manually.

Expected Results: After the SARS Copter is switched to Auto mode, it should travel successfully to within 3 m of each waypoint.

Results: The test was a success 95% of the time. Every so often the SARS Copter would pause more than 3 m from a waypoint; however, the number of failures was negligible.

11.4.4.2. Testing the SARS Rover GPS Navigation

Purpose: To verify that the SARS Rover can navigate to a specific GPS coordinate.

Materials:

- SARS Rover
- Adafruit Ultimate GPS Breakout
- Hall Effect Sensors
- Parallax Ping)))
- Magnetometers
- Power supply

Test Procedure: Given a set of GPS coordinates as a target, the rover shall be instructed to travel to those coordinates.

Expected Results: The SARS Rover shall travel to within 3 m of the destination. If the rover stops more than 9 feet from the target, the test shall be considered a failure. This test must yield a 95% success rate before integration testing may commence.

Results: This subsystem was not fully working until the night before the SARS Group's senior design demonstration; therefore, proper testing was not performed.

11.4.4.3. Geolocation Subsystem Integration Testing

Purpose: To verify the function of the subsystem once it has been integrated first with the image processing subsystem and wireless communications, and finally after it has been integrated with all of the SARS subsystems.

Materials: See the list of parts in Section 7 Integration Summary.

Test Procedure 1: This test involves the integration of the geolocation subsystem with the image processing, and the wireless communications between the SARS Copter and the CDC and between the Tiva C and the CDC. This test shall have the same procedure as the first integration test in Section 9.4.1.3: Image Processing Subsystem Integration Test; therefore, the two tests shall be performed at the same time.

Expected Results: Once the SARS Copter has traveled to all mission waypoints and all images have been extracted from the GoPro and tested, verify that the CDC has successfully identified the waypoint at which the target object is located. Once the quadcopter has safely landed, verify that the Tiva Launchpad received the correct waypoint data from the CDC. A success rate no less than 95% must be achieved.

Results: As the entire system was not working until the night before the demonstration integration testing was not performed extensively.

Test Procedure 2: This test involves the integration of the geolocation subsystem with the entirety of SARS. The same testing procedure described in the previous integration test shall be followed. This test shall be conducted after all subsystems have been thoroughly tested.

Expected Results: The result of this test should be that the SARS rover retrieves the target object after the SARS Copter has relayed its position, and the SARS rover and the SARS Copter should both return to their respective starting positions. A success rate no less than 95% must be achieved.

Results: As the entire system did not work properly until the night before the demonstration, integration testing was not performed extensively.

11.4.5. CDC/Quadcopter Communications Test

Purpose:
To test if the quadcopter's Pixhawk flight controller and the CDC can communicate with each other using the telemetry radios.

Supplies:

- Ubuntu Linux desktop or laptop environment running version 14.04 or later
- The CDC Linux application
- Pixhawk flight controller
- 3DR 915MHz USB telemetry radio
- Micro-USB cable

Procedure:

1. Connect the quadcopter battery and ensure that the Pixhawk is on.
2. Do not arm the quadcopter. This test can be performed with the quadcopter unarmed.
3. Check the quadcopter's onboard 3DR telemetry radio connection to the Pixhawk to ensure that it is firmly connected.
4. Using a micro-usb cable, connect the other 3DR telemetry radio to a USB port on the linux desktop or laptop on which the CDC will be running.

5. On the Linux computer, open a terminal window and navigate to the directory containing the CDC source code.
6. From the terminal, type python cdc.py to launch the CDC.
7. The Telemetry window should begin populating its fields with telemetry data received from the quadcopter.
8. Wait a few seconds, and determine if the telemetry window continues to update with new values.

Expected Results:

- Telemetry data from the quadcopter successfully displays in the Telemetry window of the CDC.

Conditional Requirements:

- The telemetry messages must display on the Telemetry window of the CDC.
- The delay between receipts of messages must not exceed 1 second.

11.4.6. CDC/Rover Communications Test

Purpose:
To test if the rover's Tiva C microcontroller and the CDC can communicate with each other using Wi-Fi.

Supplies:

- Ubuntu Linux desktop or laptop environment running version 14.04 or later
- The CDC Linux application
- TI Tiva C microcontroller
- TI CC3100 SimpleLink network processor

Procedure:

1. Ensure that the CC3100 is connected and in the proper orientation on the Tiva C via the booster pack header pins.
2. Power on the rover by flipping the power switch located on the side of the chassis.
3. On the Linux computer, connect to the SSID of the SARS Rover, which should be labeled as SARSNet.
4. Make sure you make the connection with the PCI Wi-Fi adapter, not the USB Wi-Fi adapter.
5. On the Linux computer, open a terminal window and navigate to the directory containing the CDC source code.
6. From the terminal, type python cdc.py to launch the CDC.
7. The Telemetry window should begin populating its fields with telemetry data received from the rover.
8. Wait a few seconds, and determine if the telemetry window continues to update with new values.

Expected Results:

- Telemetry data from the rover successfully displays in the Telemetry window of the CDC.

Conditional Requirements:

- The telemetry messages must display on the Telemetry window of the CDC.
- The delay between receipts of messages must not exceed 1 second.

11.4.7. CDC/Go-Pro Video Streaming Test

Purpose:
To test if the CDC can successfully connect to and live stream from the Go-Pro camera.

Supplies:

- Ubuntu Linux desktop or laptop environment running version 14.04 or later
- The CDC Linux application
- TP-LINK USB Wi-Fi adapter
- GoPro Hero 3

Procedure:

1. Turn on the GoPro camera and ensure that Wi-Fi is enabled by pressing and holding the small button on the left side of the camera until the blue LED on the front starts blinking.
2. Connect the TP-LINK USB Wi-Fi adapter to an available USB port on the Linux computer.
3. On the Linux computer, connect to the SSID of the GoPro, which should be labeled as SARSGoPro.
4. Make sure you make the connection with the USB Wi-Fi adapter, not the PCI Wi-Fi adapter.
5. On the Linux computer, open a terminal window and navigate to the directory containing the CDC source code.
6. From the terminal, type python cdc.py to launch the CDC.
7. In the upper left quadrant of the CDC is the Live Video Stream window. Click the play button under the video box to initiate the GoPro live video stream.
8. Watch the video feed to determine if it is connecting and displaying the video feed properly.

Expected Results:

- The Live Video Stream window successfully plays the GoPro's live video feed when the play button is pressed.

Conditional Requirements:

- The live video must stream with acceptable video quality.
- The live video stream delay must not exceed 5 seconds.

11.4.8. CDC User Interface/Integration Test

Purpose:
To test if the CDC user interface is easy to use, intuitive, and functions according to the specifications laid out in section 6.3. This test also serves as an integration test to determine how well the CDC performs will all subsystems active.

Supplies:

- Ubuntu Linux desktop or laptop environment running version 14.04 or later
- The CDC Linux application
- Pixhawk flight controller
- 3DR 915MHz USB telemetry radio
- Micro-USB cable
- TI Tiva C microcontroller
- TI CC3100 SimpleLink network processor
- TP-LINK USB Wi-Fi adapter
- GoPro Hero 3

Procedure:

1. Connect the quadcopter battery and ensure that the Pixhawk is on.
2. Do not arm the quadcopter. This test can be performed with the quadcopter unarmed.
3. Check the quadcopter's onboard 3DR telemetry radio connection to the Pixhawk to ensure that it is firmly connected.
4. Using a micro-usb cable, connect the other 3DR telemetry radio to a USB port on the linux desktop or laptop on which the CDC will be running.
5. Ensure that the CC3100 is connected and in the proper orientation on the Tiva C via the booster pack header pins.
6. Power on the rover by flipping the power switch located on the side of the chassis.
7. On the Linux computer, connect to the SSID of the SARS Rover, which should be labeled as SARSNet.
8. Make sure you make the connection with the PCI Wi-Fi adapter, not the USB Wi-Fi adapter.
9. Turn on the GoPro camera and ensure that Wi-Fi is enabled by pressing and holding the small button on the left side of the camera until the blue LED on the front starts blinking.
10. Connect the TP-LINK USB Wi-Fi adapter to an available USB port on the Linux computer.
11. On the Linux computer, connect to the SSID of the GoPro, which should be labeled as SARSGoPro.
12. Make sure you make the connection with the USB Wi-Fi adapter, not the PCI Wi-Fi adapter.
13. On the Linux computer, open a terminal window and navigate to the directory containing the CDC source code.
14. From the terminal, type python cdc.py to launch the CDC.
15. Interact with all 4 major elements of the CDC: the Video Stream window, the Telemetry window, the Mission Status window, and the Command console.
16. Without actually initiating a mission, ensure that all 4 major UI elements are functioning and easy to use.

Expected Results:

- The user interface is easy to navigate.
- The user interface is intuitive.
- The user interface functions according specification.

12. Project Operation

12.1. Quadcopter Setup and Preparation

Before taking the quadcopter out into the field, it must be calibrated and have the waypoints loaded using Mission Planner. Connect to the quadcopter using the telemetry radio and go through the hardware setup to make sure that the correct firmware is loaded and that the compass, accelerometer, and RC transmitter have all been calibrated correctly. Then, using Mission Planner's Google Maps integration tool, set up a Mission with a set delay of ten seconds at each waypoint (in order to give the GoPro enough time to pull the frame). Then, upload the mission into the Pixhawk. Finally, set the light mode switch on the Turnigy so that AUX1 flipped up sets the copter to Stabilize, and switched down sets it to Auto mode.

To initialize the quadcopter and start the mission, there are several safety precautions and setup procedures that need to be followed. To begin, make sure the copter is in an open area with few obstacles, and that all bystanders are at least 15 feet from the takeoff point. Second, make sure the LiPo battery is firmly secured to the bottom of the frame using the Velcro straps, and that the GoPro is attached to the mount underneath the nose of the copter. Third, verify that all of the propellers have been tightened down to the motors so that they rotate in sync: the propeller should not be able to swivel independent of the motor. Finally, ensure that the setup and connection procedures for the CDC have been initialized and the application terminal is printing out telemetry data to confirm a connection to the quadcopter.

12.2. Quadcopter Operation

To prepare for takeoff, first make sure the copter is placed still on the ground (it cannot be placed on any other raised surface as it will use that as its home altitude, which will be incorrect). Plug the XT60 connector of the power module into the LiPo battery. Wait until the large round light in the center of the Pixhawk begins flashing green: this signals that the flight controller has gotten its heading and GPS location and is ready for takeoff. Check to make sure that all of the switches on the Turnigy are flipped upwards and turn on the transmitter. On the quadcopter, hold down the blinking safety button until it stops flashing. Verify that the mode switch on the Turnigy has the copter set to stabilize mode, and hold the throttle (right stick) all the way down and the left stick all the way to the right. The quadcopter will make a sound signifying that it is now armed and able to be flown.

Carefully raise the throttle, balancing the copter with the left stick. Once the quadcopter is comfortably 2-3 feet in the air, flip the mode switch to auto mode. The copter will navigate to each programmed waypoint, sitting still once it has reached the last one where the user can grab it from the bottom. After the quadcopter has switched to auto mode, make sure to pull the throttle all the way back down so when the user changes back to stabilize at the end of the mission to secure the copter, the propellers don't continue to spin.

12.3. Quadcopter Common Issues / Debugging

Quadcopter unable to arm:
This is usually caused by one of the prearm flight checks the flight controller goes through to ensure everything is ready for takeoff. One common failure is a high HDOP value, which stands for Horizontal Dilution of Precision. This means that there is too much inaccuracy with the quadcopter's GPS. This can be caused by magnetic interference, cloud coverage, or a number of other factors. If this is the case, unplug the battery and wait 10 minutes and try again.

Propellers not spinning fast enough to takeoff:
If the propellers are spinning but the quadcopter is not taking off, this is an indication that the battery is about to die. Remove the battery immediately and place it on a charger: if the LiPo battery gets drained too far, it can permanently damage one of the cells and ruin the battery.

12.4. Rover Setup and Preparation

The main setup requirement for the rover is to upload the appropriate waypoints into the control program so that the rover can easily choose the proper destination after it is initiated by the CDC. Once the potential waypoints are loaded, the program is transferred to the Tiva C to be run.

To prepare the rover for use, ensure all components are securely fastened to the rover body. Once all parts have been secured, ensure the wiring is properly connected and that the battery is charged. Ensure that the retrieval arm is in the upright, or close to the upright position and that it will not interfere with any wires. The final step in preparation is to flip the power switch into the "ON" position and to verify that all status lights indicate proper function.

12.5. Rover Operation

To begin mission initialization, place the rover on the ground and power it on. Allow 30-45 seconds until the GPS indicates that it has a lock, and then reset the Tiva using the reset button (not the power switch) so that all of the code will be processed appropriately. After this sequence is completed, the rover will go through a series of peripheral initializations and a single LED will indicate that it is waiting for a target waypoint command from the CDC. Once the rover receives a target waypoint it will begin its mission and requires no further user operation. Upon mission completion the rover remains in the state where it is waiting for a new command from the CDC indefinitely. It will continue to run missions as long as the battery supports.

12.6. Rover Common Issues / Debugging

Unable to wirelessly connect to the rover:
The peripheral startup process may have failed. Use the reset button to force the rover to go through the startup process again.
GPS, WiFi, or arm malfunctioning:

The overall power draw of the system has exceeded the current capability of the battery. The battery needs to be charged prior to further operation

Red flashing light on the Sabertooth:
Low battery voltage. Charge the battery.

12.7. CDC Setup and Preparation

Most of the prerequisite libraries needed for the CDC to function properly are included as Python libraries with the source code. However, pymavlink must be downloaded and installed separately before the CDC can be used. The easiest way to install pymavlink is by installing MAVProxy through the command line. The following procedure will download and install MAVProxy so that the pymavlink library can be referenced by the CDC source code.

1. From a Linux desktop environment, open a terminal window
2. Install the Python Package Index (PIP) by typing ‘sudo apt-get install python-pip’
3. Use PIP to install MAVProxy by tying ‘pip install MAVProxy’

With pymavlink installed, the CDC should now have everything needed to properly function. To start the CDC, open a linux terminal and navigate to the directory containing the CDC source code. See the following section for details on starting the CDC.

12.8. CDC Operation

Before initiating a mission, ensure that the quadcopter, rover, and GoPro are all powered on and operational. Connect both the USB telemetry radio and the USB Wi-Fi adapter to the Linux computer. Make sure that the PCI Wi-Fi adapter is connected to SARSNet and that the USB Wi-Fi adapter is connected to SARSGoPro. From within the source code folder, type ‘python cdc.py’ to launch the application. Once all connections have been made, the application’s user interface should appear. Test the video feed and monitor the telemetry windows to ensure that everything is operational. Type ‘run’ into the command console to begin a mission. The mission progress window will begin updating with the mission’s status. The live feed from the GoPro will play in the Video Stream window and the telemetry data from both the quadcopter and the rover will update in the Telemetry window.

12.9. CDC Common Issues / Debugging

Unable to connect to the quadcopter:
Check the connection to the telemetry radio. If the quadcopter is still unable to connect, then use Mission Planner to open a connection first, then reconnect using the CDC. There is a known issue where the quadcopter fails to communicate with pymavlink without having established a connection to Mission Planner first.

Unable to connect to the rover:

Check the physical connection from the CC3100 to the Tiva. Make sure you are connected to the SARSNet SSID with the PCI Wi-Fi Adapter.

Unable to connect to the GoPro:
Check the Wi-Fi settings on the GoPro and ensure Wi-Fi is broadcasting. Check the connection to the USB Wi-Fi adapter. Make sure you are connected to the SARSGoPro SSID with the USB Wi-Fi adapter.

13. Administrative Content

13.1. Milestones

Design documentation was handled concurrently with all project milestones. The following tables display the development phases of each project subsystem as well as the intended completion date for each phase.

Table 10-13-1: Quadcopter Milestones below displays the intended completion dates for all major development phases of the SARS Copter. Development of this project component was completed on schedule.

Component		Completion Date
Quadcopter	Specs	9/9/2014
	Research	10/10/2014
	Design	11/23/2014
	Build	1/9/2014
	Test	2/13/2015
	Integration	4/1/2015

Table 10-13-1: Quadcopter Milestones

Table 10-13-2: Image Processing Milestones below displays the originally intended completion dates for all major development phases of the SARS image processing subsystem. Development of this project component was completed on schedule.

Component		Completion Date
QC Camera/Item Detection	Specs	9/9/2014
	Research	10/10/2014
	Design	11/23/2014
	Build	1/9/2015
	Test	2/13/2015
	Integration	4/1/2015

Table 10-13-2: Image Processing Milestones

Table 10-13-3: Quadcopter Communications Milestones below displays the originally intended completion dates for all major development phases of the SARS Copter communications interface. Development of this project component was completed on schedule.

Component		Completion Date

QC Communication Interface	Specs	9/9/2014
	Research	10/10/2014
	Design	11/21/2014
	Build	12/12/2014
	Test	2/2/2015
	Integration	4/1/2015

Table 10-13-3: Quadcopter Communications Milestones

Table 10-13-4: Ground Rover Milestones below displays the originally intended completion dates for all major development phases of the SARS Rover hardware. Development of this project component was completed on schedule.

Component		**Completion Date**
Ground Rover	Specs	9/9/2014
	Research	10/10/2014
	Design	11/7/2014
	Build	1/9/2014
	Test	2/13/2015
	Integration	4/1/2015

Table 10-13-4: Ground Rover Milestones

Table 10-13-5: Rover Item Detection Milestone below displays the originally intended completion dates for all major development phases of the SARS Rover item detection subsystem. Development of this project component was completed on schedule.

Component		**Completion Date**
Rover Item Detection	Specs	9/9/2014
	Research	10/10/2014
	Design	11/23/2014
	Build	1/9/2015
	Test	2/13/2015
	Integration	4/1/2015

Table 10-13-5: Rover Item Detection Milestones

Table 10-13-6: Rover Communications Milestones below displays the originally intended completion dates for all major development phases of the SARS Rover communications interface. Development of this project component was completed on schedule.

Component		**Completion Date**
Rover Communications	Specs	9/9/2014
	Research	10/10/2014
	Design	11/21/2014
	Build	1/9/2015
	Test	2/2/2015
	Integration	4/1/2015

Table 10-13-6: Rover Communications Milestones

Table 10-13-7: Android Application Milestones below displays the originally intended completion dates for all major development phase of the SARS Android application. This project component was completed on schedule.

Component		Completion Date
Android Application	Specs	9/9/2014
	Research	10/10/2014
	Design	10/24/2014
	Build	11/23/2014
	Test	2/13/2015
	Integration	4/1/2015

Table 10-13-7: Android Application Milestones

Displayed are the significant project milestones for each SARS subsystem. These milestones correspond with the build, test, and integration phases of the related SARS subsystems.

Figure 13-1: Quadcopter Milestones below displays all major tasks necessary for the completion of the build stage of the SARS Copter development.

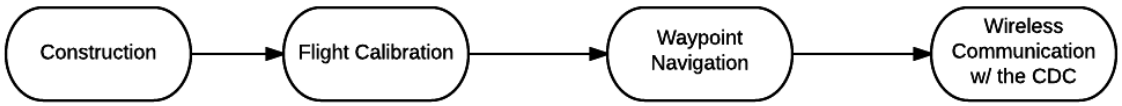

Figure 13-1: Quadcopter Milestones

Figure 13-2: Image Processing Milestones displays all major tasks necessary for the completion of the build stage of the SARS image processing subsystem development.

Figure 13-2: Image Processing Milestones

Figure 13-3: Rover Milestones below displays all major tasks necessary for the completion of the build stage of the SARS Rover development.

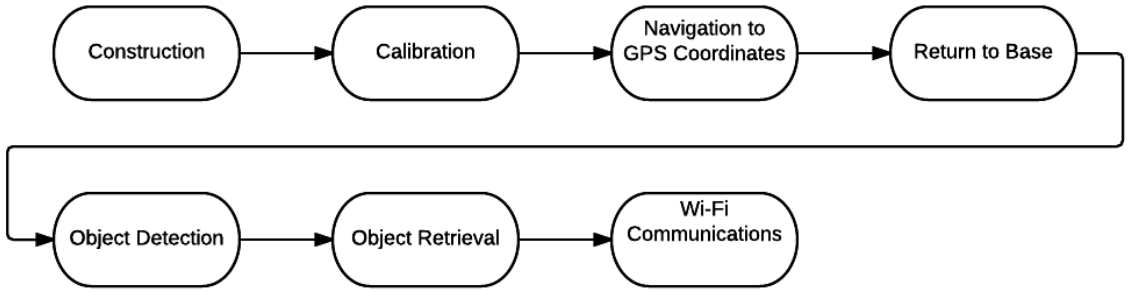

Figure 13-3: Rover Milestones

Figure 13-4: Android Application Milestones below displays all major tasks necessary for the completion of the build stage of the SARS Android application development.

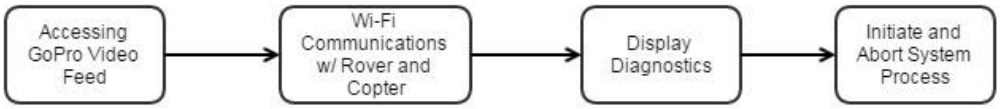

Figure 13-4: Android Application Milestones

13.2. Project Team

Matthew Bahr is a Computer Engineering student at the University of Central Florida. He graduates in Spring 2015. He is an officer in UCF's Cuong Nhu club and the front man of a local band. After his graduation, Matt will be traveling to Southeast Asia for an undetermined length of time.

Brian Crabtree is a Computer Engineering major at the University of Central Florida. He graduates in Spring 2015. He is a member of the Burnett Honors College and an officer for the Men's Ultimate Frisbee Club at UCF. After graduation, Brian will be going to work for Intel as an SOC Design Engineer

Brendan Hall is a senior computer engineer at the University of Central Florida. He is a member of Sigma Nu Fraternity and has worked as a CWEP at Lockheed Martin for over two years. After graduation, he will be joining Citi's Technology Leadership Program in Jacksonville, FL.

Erick Makris is a senior computer engineering student at the University of Central Florida. He is currently working at FLIR Systems as a Software Engineering intern, and will be begin a full-time position as a Software Developer at FLIR upon graduation

13.3. Budget and Financing

Group 4 received $500 from SoarTech and $1000 from Boeing in sponsorships for a grand total of **$1500** in funding. This amount of funding was almost enough to cover all hardware and testing needed to implement the complete search-and-retrieval system. The largest portion of project expenses came from the quadcopter. Depending on the pre-installed capabilities of the copter and whether or not it is pre-assembled, a quadcopter can realistically run up to $1000. Group 4 decided to go with a quadcopter costing $550. This cost was financed through the Boeing sponsorship. On top of just the copter, the camera device and mounting system used for the field detection can also be quite expensive as we needed a reliable camera for the item detection from the air. The camera, a GoPro Hero3 White, cost $199.99; however, the parents of one of the group members have opted to pay for it. The other main expense was the rover chassis, motors, power supply, and motor controller, along with the sensors for object detection and the object retrieval apparatus. Approximately $400 has been spent on a chassis, a motor controller, and several sensors. This was the maximum funding we expected to allocate to the rover; however, because the camera was not paid for using sponsored funding, extra funding was left open for the SARS Rover. Eventually the sponsored funding was depleted, though, and Group 4 had to pool personal funds to cover the remaining costs.

13.4. Bill of Materials

Below in Table 13-8 is the current list of all materials that will be required to implement the Search and Retrieval System. Some items have already been acquired, and the list is subject to change pending final design implementation.

Item	Supplier	Cost	Notes
3DR Quad Kit	3DR Robotics	$550.00	
950 MHz Radio Receivers	3DR Robotics	$100.00	
Turnigy 9x Radio Transmitter	Hobby King	$80.00	
PPM Encoder	3DR Robotics	$25.00	
Quad Battery Pack	3DR Robotics	$60.00	
GoPro Hero3 White	Amazon	$200.00	Donated
GoPro Mount Supplies	Home Depot	$10.00	
TI CC3100 SimpleLink	TI Innovation Lab	$-	Loaned
TI CC31XX EMU Boost	TI Innovation Lab	$-	Loaned
TP-Link TL-WN722N		$-	Previously owned
Battery Charger	Amazon	$150.00	
Rover Chassis/Motors	Sparkfun	$250.00	
Rover Motor Controller	Amazon	$125.00	
Rover Battery	Amazon	$50.00	
Rover MCU	TI Workshop	$-	Loaned
Rover IMU	TI Workshop	$-	Loaned
Rover GPS	Amazon	$60.00	
Rover Ultrasonic Sensor	Amazon	$30.00	
Rover Power Switch	Radio Shack	$5.00	
PCB		$150.00	
Total		**$1720.00**	

Table 13-8: Bill of Materials

13.5. Consultants, Subcontractors, and Suppliers

The primary consultant for SARS was one of the group sponsors, SoarTech. Joshua Haley acted as our primary contact within the company, and he provided valuable advice regarding some of the design decisions for the system. Various suppliers were used for parts acquisition. 3DR, Hobby King, and Horizon Hobby were used as suppliers for the quadcopter parts. Sparkfun, Amazon, DigiKey, Adafruit, Home Depot, and Jameco were used as suppliers for the rover parts. PCB Pool was contracted for the rover's PCB design. Quality Manufacturing Services was contracted for PCB assembly.

14. Conclusion

SARS represents the forefront of drone technology and autonomous computer systems. Using two separate autonomous systems linked with communication software, SARS proves that automated drone technology will continue to innovate new applications, both military and commercial. Each of the SARS subsystems presented a significant design challenge to the engineers, and the project offered valuable integration experience. Over the past eight months, each member of the SARS Group has learned much and been allowed the opportunity to improve a number of engineering skills.

A. References

1. “Autonomously Stabilized Quadcopter” Web
   < http://www.thomasteisberg.com/quadcopter/ >

2. “Welcome to the Next Open Source Generation Quadcopter” Web.
   < http://ng.uavp.ch/FrontPage >

3. “Quadcopter Dynamics, Simulation, and Control” Web
   <http://andrew.gibiansky.com/downloads/pdf/Quadcopter%20Dynamics,%20Simulation,%20and%20Control.pdf >

4. “Build a Quadcopter from Scratch – A Hardware Overview” Web.
   < http://blog.oscarliang.net/build-a-quadcopter-beginners-tutorial-1/ >

5. “Best Flight Controller for Quadcopter and Multicopter” Web.
   < http://robot-kingdom.com/best-flight-controller-for-quadcopter-and-multicopter/ >

6. “Beagle-Fly” Web.
   < https://code.google.com/p/beagle-fly/ >

7. “Snoopy Copter” Web.
   < http://beagleboard.org/project/Snoopy/ >

8. “BeagleDrone” Web.
   < http://andicelabs.com/beagledrone/>

9. “AeroQuad Cyclone Frame” Web.
   < http://aeroquad.com/showwiki.php?title=AeroQuad+Cyclone+Frame>

10. “Tradeoffs when considering SPI or I2C?” Web.
    < http://electronics.stackexchange.com/questions/29037/tradeoffs-when-considering-spi-or-i2c>

11. “USART, UART, RS232, USB, SPI, I2C, TTL, etc. what are all of these and how do they relate to each other?” Web.
    < http://electronics.stackexchange.com/questions/37814/usart-uart-rs232-usb-spi-i2c-ttl-etc-what-are-all-of-these-and-how-do-th>

12. “How to build your own Quadcopter Autopilot/Flight Controller” Web.
    < https://ghowen.me/build-your-own-quadcopter-autopilot/>

13. “Using I2C to communicate between Pixhawk and other boards” Web.
    < https://groups.google.com/forum/#!topic/px4users/hsPxLHKhQkc>

14. “APM Copter” Web.
< http://copter.ardupilot.com/>

15. “APM adaptive flying/video analysis waypoint help” Web.
< http://diydrones.com/forum/topics/apm-adaptive-flying-video-analysis-waypoint-help>

16. “BeagleBone: an I2C tutorial” Web.
< http://derekmolloy.ie/beaglebone/beaglebone-an-i2c-tutorial-interfacing-to-a-bma180-accelerometer/>

17. “BeagleBone Black I2C References” Web.
< http://datko.net/2013/11/03/bbb_i2c/>

18. “Quadcopter ESC’s” Web.
< http://oddcopter.com/2012/02/21/quadcopter-escs-electronic-speed-controllers/>

19. “BeagleBone – Unleash Your BeagleBone: Battery Powered” Web.
< http://inspire.logicsupply.com/2014/08/beaglebone-unleash-your-beaglebone.html>

20. “How To Choose RC Transmitter for Quadcopter” Web.
< http://blog.oscarliang.net/choose-rc-transmitter-quadcopter/>

21. GoPro Hero3 White
http://shop.gopro.com/cameras/hero3-white-edition/CHDHE-302-master.html

22. Raspberry Pi w/ PiCam
http://www.raspberrypi.org/products/camera-module/

23. Open Source GoProController Python Code
https://github.com/joshvillbrandt/GoProController

24. Homemade camera mount
http://flitetest.com/articles/_Vibration_Free_Camera_Mount

25. MMAL API
http://www.jvcref.com/files/PI/documentation/html/

26. V4L API
http://linuxtv.org/downloads/v4l-dvb-apis/index.html

27. Image Processing
http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html

28. Image Processing
Robust Real-Time Object Detection Paper

29. Adafruit Ultimate GPS Breakout
http://www.adafruit.com/products/746

30. Sparkfun Copernicus II GPS Module
https://www.sparkfun.com/products/10922

31. RoboGoby Project UART + GPS
http://robogoby.blogspot.com/2014/04/beaglebone-black-uart-gps.html

32. 3D Robotics Inc.
http://store.3drobotics.com/products/3dr-gps-ublox-with-compass

33. LNA and SAW Filter
http://www.digikey.com/Web%20Export/Supplier%20Content/Melexis_413/PDF/Melexis_AppNote_MLX71120_21_SAW.pdf?redirected=1

34. GPS Serial Communication
http://www.boondog.com/tutorials/gps/gps.html

35. Robot Arm
http://www.societyofrobots.com/robot_arm_tutorial.shtml

36. Vacuum Pump
https://www.youtube.com/watch?v=oGb-UwAXicI

37. HTC RE Camera
http://www.htc.com/us/re/re-camera/

38. A Positive Pressure Universal Gripper Based on the Jamming of Granular Material
John R. Amend, Jr., Student Member, IEEE, Eric Brown, Nicholas Rodenberg, Heinrich M. Jaeger, and Hod Lipson, Member, IEEE. IEEE TRANSACTIONS ON ROBOTICS, VOL. 28, NO. 2, APRIL 2012.

39. Sabertooth dual 2X25A Motor Driver
https://www.dimensionengineering.com/products/sabertooth2x25

40. Turnigy 5000mAh 4S1P 14.8V 20C Hardcase Pack
https://www.hobbyking.com/hobbyking/store/uh_viewItem.asp?idproduct=15521.html

41. Advantages & Limitations of Lithium Ion Batteries
http://batteryuniversity.com/learn/article/is_lithium_ion_the_ideal_battery

42. What's the Best Battery?
http://batteryuniversity.com/learn/article/whats_the_best_battery

43. AWG Wire Sizes

http://www.powerstream.com/Wire_Size.htm

44. Wheels vs Continuous Tracks: Advantages and Disadvantages
http://www.intorobotics.com/wheels-vs-continuous-tracks-advantages-disadvantages/

45. Texas Instruments Tiva C Product Listings
http://www.ti.com/product/TM4C129XNCZAD/compare

46. Texas Instruments Concerto Product Listings
http://www.ti.com/product/F28M35H52C/compare

47. ZigBee
http://en.wikipedia.org/wiki/ZigBee

48. IEEE 802.15.4
http://en.wikipedia.org/wiki/IEEE_802.15.4

49. Xbee Family Features Comparison
http://www.digi.com/pdf/chart_xbee_rf_features.pdf

50. TI Wireless Connectivity Guide
http://www.ti.com/lit/sg/slab056d/slab056d.pdf

51. Digi XCTU
http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/xctu

52. Android Studio
https://developer.android.com/sdk/installing/studio.html

53. Intellij IDEA
http://en.wikipedia.org/wiki/IntelliJ_IDEA

54. MPEG Transport Stream
http://en.wikipedia.org/wiki/MPEG_transport_stream

55. HTTP Live Streaming
http://en.wikipedia.org/wiki/HTTP_Live_Streaming

56. Parallax Ping))) Ultrasonic Distance Sensor
http://www.parallax.com/product/28015

57. Infrared vs. Ultrasonic – What You Should Know
http://www.societyofrobots.com/member_tutorials/node/71

58. SHARP GP2Y0A21YK Data Sheet

http://www.sharpsma.com/webfm_send/1208

59. A Sensitive DIY Ultrasonic Range Sensor
    http://www.kerrywong.com/2011/01/22/a-sensitive-diy-ultrasonic-range-sensor/

60. Sensors – Robot Sonar
    http://www.societyofrobots.com/sensors_sonar.shtml

61. Sensors – Sharp IR Range Finder
    http://www.societyofrobots.com/sensors_sharpirrange.shtml

62. BeagleBone: Serial Ports and Xbees
    http://www.jerome-bernard.com/blog/2012/06/04/beaglebone-serial-ports-and-xbees/

63. Nexus 5 Tech Specs
    https://support.google.com/nexus/answer/3467463?hl=en&ref_topic=3415523

64. Xbee Wi-Fi
    http://www.digi.com/products/wireless-wired-embedded-solutions/zigbee-rf-modules/point-multipoint-rfmodules/xbee-wi-fi

B. Copyright Permissions

Figure 4-1: "Schematic of reaction torques on each motor of a quadrotor aircraft, due to spinning rotor" by Gabriel Hoffma is licensed under CC by 2.5

Figure 4-2: "Cascade Classifier" by Paul Viola and Michael Jones

Figure 4-3: "Classifier Features" by Paul Viola and Michael Jones

Figure 4-7: "Black Hardware Details" by jkridner under CCA by 3.0

Figure 4-8: "Typical SPI bus: master and three independent slaves" by en:User:CBurnett under CCA BY-SA 3.0

Figure 4-9:

Privacy Policy of EngineersGarage

Dear Visitor

Please read the privacy policy before using this website.

EngineersGarage strives to maintain the highest standards of decency, fairness and integrity in all its operations. Any user who finds material posted by another user objectionable is encouraged to contact us via e-mail. EngineersGarage is authorized to remove or modify any data, submitted by any user to its forums, for any reason it feels constitutes a violation of our policies, whether stated, implied or not.

Content

The content provided on this website is only for the purpose of reference, education and personal use. If you are using our content for reference, you should acknowledge us by putting a link of our website www.engineersgarage.com under suitable heading. The content should not be used in any form for any commercial purpose without prior permission from us. Although the content posted in EG Labs section is well researched and experimented before posting, however EngineersGarage bears no responsibility in any form for the failures that may arise. EngineersGarage does not bear any responsibility in any form for the content posted by its users.

Figure 4-12: "Triangulation with Infrared Sensors" Licensed under CC-BY-SA-2.5

Figure 4-13: "Sharp GP2Y0A21YK V-L Relationship"

Senior Design Project

Erick Makris <nox357@gmail.com> Wed, Dec 3, 2014 at 12:06 PM
To: karamy@xposureunlimited.com

Dear Karamy,

My name is Erick Makris and I'm a Senior Computer Engineering student at the University of Central Florida. I am currently working on a Senior Design project, and we may be incorporating some Sharp IR sensors into our automated rover to aid with object detection and avoidance. As part of our design documentation, we would like to request permission to use some of the figures and graphs from Sharp's datasheets, specifically the data sheet for the Sharp GP2Y0A21YK infrared sensor. I'm not sure if you are the right person to contact regarding this matter. If you aren't, would you be able to

direct me to the proper channels to get this permission? Your assistance in this matter is greatly appreciated!

Sincerely,
Erick Makris

Figure 14-1: "Ultrasonic Ranging" Dual licensed under GFDL and CC-BY-SA-3.0

Figure 4-20: "HLS Streaming Protocol"

Erick Makris
3379 S. Kirkman Rd., Apt. 1038
Orlando, FL 32811
12/02/14

Apple Inc.,
Attention: Rights and Permissions,
1 Infinite Loop MS 169-3IPL,
Cupertino,
CA 95014.

To Whom It May Concern,

My name is Erick Makris and I'm a Senior Computer Engineering student at the University of Central Florida. I am currently working on a Senior Design project, and will be performing some HTTP Live Streaming from a Go-Pro camera to an Android device as part of the project. As part of our design documentation, we would like to request permission to use the HTTP Live Streaming diagram illustrated in Apple's HTTP Live Streaming Guide located at

https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/Introduction/Introduction.html

The design document will be hosted on the University of Central Florida's Computer Engineering Senior Design website, but the design document itself would not be used for any commercial purposes, only educational. Please respond in kind at your earliest convenience. Your assistance in this matter is greatly appreciated!

Sincerely,
Erick Makris

Figure 5-1: "Cape Headers I2C" by jkridner under CCA by 3.0

Figure 5-1: "Quadcopter Wiring Diagram" by Arducopter under CC by 3.0

Figure 5-7 & 5-8: Photos by Oscar Liange under CC by 2.5

Figure 5-9 & Figure 5-10: "Product Details Dimensions" 2005-2014 Welcome to DAGU Hi-Tech Electronic Robotics online Shop! Copyright, All Rights Reserved.

Robust Real-time Object Detection

Permission to Use Figures From Robust Real-time Object Detection

Matthew Bahr <mattbahr1992@gmail.com> Wed, Dec 3, 2014 at 9:43 PM
To: viola@merl.com, mjones@crl.dec.com

Hello,

I am a Senior Computer Engineering student at the University of Central Florida. I am currently working on a design project that involves the training of classifiers to detect objects using a camera mounted on a quadcopter. As part of our design documentation, we would like to request permission to use some of the figures and graphs from your paper, Robust Real-time Object Detection. Your assistance in this matter is greatly appreciated!

Sincerely,
Matthew Bahr

Equation 1: "Integral Image Equations" by Paul Viola and Michael Jones

OpenCV Copyright Permissions

Permission to Use Copyrighted Material

Matthew Bahr <mattbahr1992@gmail.com> Wed, Dec 3, 2014 at 10:08 PM
To: admin@opencv.org

Hello,

I am a Senior Computer Engineering student at the University of Central Florida. I am currently working on a design project that involves the training of classifiers to detect objects using a camera mounted on a quadcopter. As part of our design documentation, we would like to request permission to use some of the figures from the OpenCV website. Your assistance in this matter is greatly appreciated!

Sincerely,
Matthew Bahr

Figure 6-4: "Face Detection Output" Copyrighted 2011-2014 by opencv dev team

Diagram Copyright Permission Request

crabbybrian

Wed 12/3/2014 1:14 PM

Sent Items

To:sale@dagurobot.com <sale@dagurobot.com>;

Hello,
I am a student at the University of Central Florida and I am currently working on a senior design project involving the
Dagu Wild Thumper 6WD Robot Chassis. I would like to request permission to use the product dimension diagrams available on your website on the Wild Thumper product page (http://www.dagurobot.com/goods.php?id=154) in our documentation of this project. Documentation will be posted online and visible to the general public, and Dagu will be properly identified as the creator of the diagrams. Please let me know your decision regarding your permission to use the copyrighted material.

Thank You,
Brian Crabtree

GEN, Email Technical Support, www.ti.com, EKTM4C1294XL/BOOSTXL-SENSHUB

crabbybrian@knights.ucf.edu
Wed 12/3/2014 7:23 PM
Inbox
Cc:crabbybrian@knights.ucf.edu <crabbybrian@knights.ucf.edu>;

[This Email Sent From: Email Technical Support
http://www.ti.com/general/docs/contact.tsp]
[wfsegen]
[DATE / TIME (UTC): Thu, 04 Dec 2014 00:23:49 GMT]
[CUSTOMER'S REGIONAL LOCAL TIME: 12/3/2014, 7:23:49 PM]
[Name: Brian Crabtree]
[Prefix: Mr.]
[First Name: Brian]
[Last Name: Crabtree]
[Job Title:]
[Company: University of Central Florida Student]
[Email: crabbybrian@knights.ucf.edu]
[Phone: 4079252953]
[FAX:]
[Country: USA]
[Address1: 560 Serenity Place]
[Address2:]
[City: Lake Mary]
[State: FL]
[Postal Code: 32746]
[Part# or Description: EK-TM4C1294XL/BOOSTXL-SENSHUB]
[Category: Access and Licensing]
[Application: Other]
[Design Stage: New design]
[Estimated Annual Production: 1 units]
[Production Date: 4/1/2015]
[Problem:
Hello, I am a student at the University of Central Florida and I am currently working on a senior design project involving the
TM4C1294 Connected LaunchPad and the Sensor Hub BoosterPack. I would like to request permission to reproduce diagrams and tables contained within the user manuals for both of these products within our design documentation. Once completed, our design documentation will be posted online and visible to the general public, and Texas Instruments will be properly identified as the creator of the diagrams and tables. Please let me know your decision regarding your permission to use the copyrighted material. Thank You, Brian Crabtree]

Table 5-5: “BoosterPack XL Connector” Copyright © 2013, Texas Instruments Incorporated