SARS: Search and Retrieval System

Senior Design I Fall 2014

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. The third subsystem is an Android application which shall be used to initiate 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 are as follows:

- Image processing to identify the target object from the air
- Interrupting the SARS Copter software to alter its route once the object has been found
- Wireless communications
- Automated GPS navigation for the SARS Rover
- Object avoidance and retrieval for the SARS Rover
- Power distribution in both the SARS Copter and the SARS Rover
- Android development

At the current point in project development, the SARS Group has made significant progress in the design of all major subsystem. The only aspect of the system that is still unready for the build stage of development is the object retrieval mechanism. The team came up with the concept for SARS with scalability in mind. Depending on the relative ease of implementing the other SARS subsystems, a more versatile and challenging retrieval method may be built; otherwise, a simple method shall be implemented as a proof of concept. There is no doubt that unforeseen difficulties shall be encountered during the upcoming build, testing, and implementation, but the team is optimistic and confident that the work done thus far has placed it in good standing to complete the project.

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 an android application. All three aspects of the system work simultaneously 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 quadcopter, is responsible for locating the object we will be searching for. Mounted with a high quality camera, the quadcopter will be fed a series of waypoints from the android application, dictating the area of which it will need to search for the object. The copter will then scan the area using object detection, fly directly above that object's locations and relay the GPS coordinates of the target to the rover on the ground via Bluetooth communication. The quadcopter is composed of six primary components: the copter itself (which we will purchase prefabricated), an Internal Measurement Unit that will analyze real-time telemetry that allows the quadcopter to stay in flight, a camera to provide live video streaming and that will be used to detect the target object, a Bluetooth device to communicate with the rover and android interface, a GPS chip that relays the quadcopter's location to the user via the android application, and finally a microcontroller.

The ground rover is the aspect of the project which will physically retrieve the object from its location. Once the object's GPS coordinates have been transmitted from the quadcopter, the rover will move to that object location and pick it up using an attached arm, similar to that of a golf ball retriever pole. After retrieving the item, the rover will return to its original location. Throughout the entire mission, the rover will constantly be relaying information about its position and speed to the user via the android application. The main components of the rover are the rover itself (which we'll also order prefabricated), the retrieval arm, the microcontroller, and the Bluetooth device.

The final piece of our project is the android application. This will display real-time diagnostics to the user about the two vehicles, as well as provide a live aerial video stream from the quadcopter. The only user inputs necessary are the initial waypoints for the quadcopter to search, but after that the system acts completely autonomously.

Although an ambitious project, we believe that this allows us to gain experience in many different arenas, from android development to wireless communication to object detection to hardware development, system integration and artificial intelligence. Our four team members are up to the challenge, and with the help of Boeing, we know we can accomplish our goals.

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 is being implemented in a classroom setting rather than a market setting, its main objective shall be to meet all specifications and perform in a manner that will secure each member of Group 4 a grade of A in Senior Design.

- To pass Senior Design with a grade of A.
- To attract potential employers.
- To implement an effective, professional quality 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 an Android interface. Successful communications between these three subsystems will be vital to the project implementation.

3.2.1 Quadcopter

Group 4 decided upon the following specifications for the SARS Copter subsystem. These specifications are intended to allow the SARS Copter to complete the tasks of identifying an object on the ground and relaying its GPS location to the SARS Rover subsystem.

- Capable of interfacing wirelessly with a rover on the ground and with an Android device.
- 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 5 ft.
- Capable of hovering at a constant height between 10 ft and 50 ft with a variation in height no greater than 3 in.
- Has battery life up to 10 minutes.
- Basic weatherproofing

3.2.2 Ground Rover

Group 4 decided upon the following specifications for the SARS Rover subsystem. These specifications are intended to allow the SARS Rover to complete the tasks of receiving GPS coordinates from the SARS Copter subsystem, traveling to those coordinates while avoiding obstacles, and retrieving a target object located at the received coordinates.

- Capable of interfacing wirelessly with a quadcopter in the air and with an Android device.
- Capable of travelling up to 1000ft. 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 GPS coordinates.
- Able to return to within 5ft. of its starting location with the retrieved target object.
- Able to carry a load of 5lbs.
- Basic Weatherproofing

3.2.3 Android Application

Group 4 decided upon the following specifications for the SARS Android application. These specifications are intended to allow a user to interface with the SARS subsystems. The application should provide a live video stream from the SARS Copter and display diagnostic information for the SARS Rover and the SARS Copter. Finally, it should allow the user to initiate and abort the functioning of SARS.

- Provide stop and start commands to rover and quadcopter
- View live video stream from quadcopter camera
- View GPS and other sensor data from quadcopter and rover

3.3 System Overview

SARS consists of 3 major systems. The quadcopter, the rover, and the Android application. Each system is diagrammed below, with a color coded legend indicating which member of the SARS team will be responsible for a given subsystem. A primary and secondary engineer is designated for each subsystem. The primary engineer is responsible for subsystem development, and may choose to delegate some tasks relating to that subsystem to the secondary engineer.

Figure 3-1: Legend below displays the color codes associated with each member of Group 4. In the subsequent figures, these colors will be used to indicate which group members are primarily and secondarily responsible for the development of each SARS subsystem.

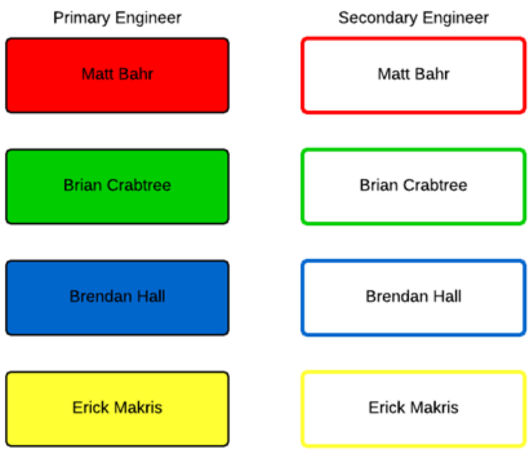


Figure 3-1: Legend

Figure 3-2: System below displays the block diagram for the entirety of SARS. Responsibilities are not yet specified for the quadcopter, Android interface, and rover subsystems; however, Erick Makris will be primarily responsible for the development of all inter-subsystem wireless communications, and he will have support from Matt Bahr.

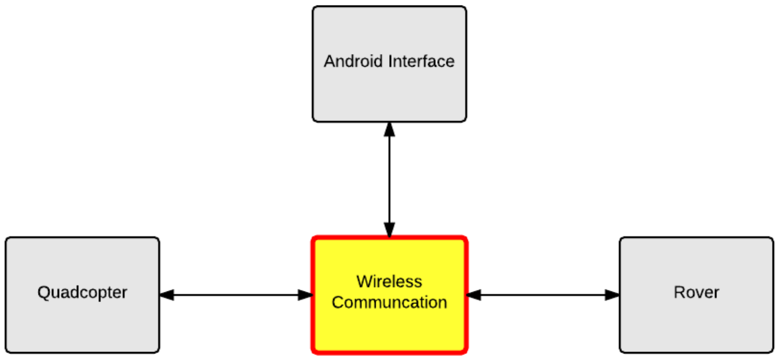


Figure 3-2: System

Figure 3-3: Quadcopter below displays the block diagram for the SARS Copter subsystems. As displayed in the figure, Brendan Hall shall be primarily responsible for the SARS Copter device control, and he shall have support from Matt Bahr. Matt Bahr shall be primarily responsible for the geolocation, image processing, and camera subsystems, and he shall have support from Brendan Hall on the camera subsystem and from Brian Crabtree on the image processing and geolocation subsystems. Furthermore, Erick Makris shall be primarily responsible for the communications with the SARS Rover and with the Android application, and he shall have support from Brendan Hall.

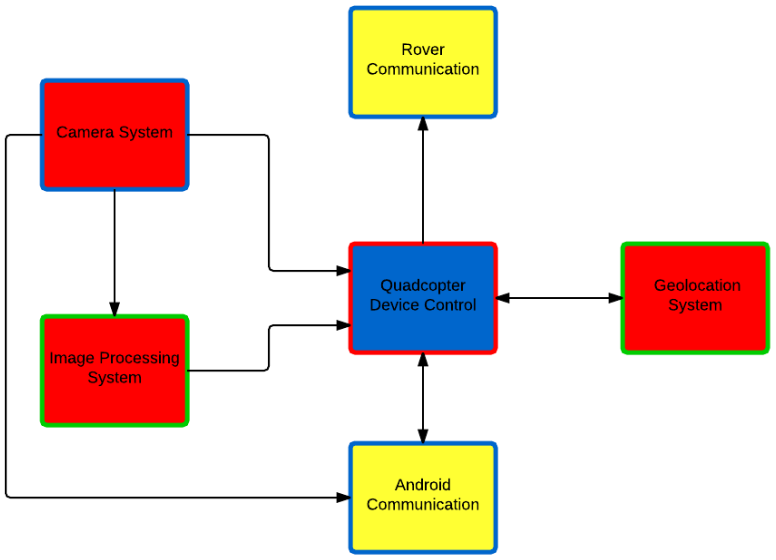


Figure 3-3: Quadcopter

Figure 3-4: Rover below displays the block diagram for the SARS Rover subsystems. As displayed in the diagram, Brian Crabtree shall be primarily responsible for the device control unit, motor controller, wheel motors, item retriever, and GPS unit, and he shall have support from Erick Makris on the device control unit, from Brendan Hall on the motor controller, the wheel motors, and the item retriever, and from Matt Bahr on the GPS unit. Brendan Hall shall be primarily responsible for the target object detector and obstacle sensor, and he shall have support from Brian Crabtree. Finally, Erick Makris shall be primarily responsible for the wireless communication interface, and he shall have support from Brendan Hall.

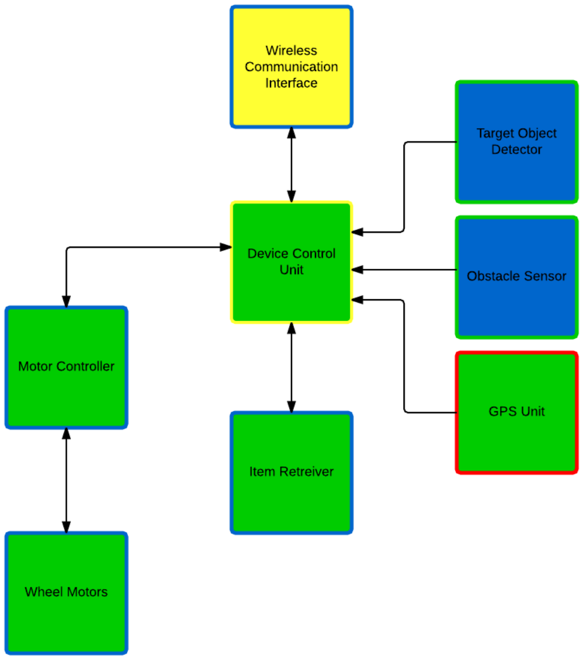


Figure 3-4: Rover

Figure 3-5: Android Application below displays the block diagram for the SARS Android application. As displayed in the figure, Erick Makris shall be primarily responsible for the Graphical User Interface, the Quadcopter Communication/Diagnostics System, and the Rover Commnication/Diagnostics System, and he shall have support from Brendan Hall. Brendan Hall shall be primarily responsible for the Quadcopter Video Stream System, and he shall have support from Erick Makris.

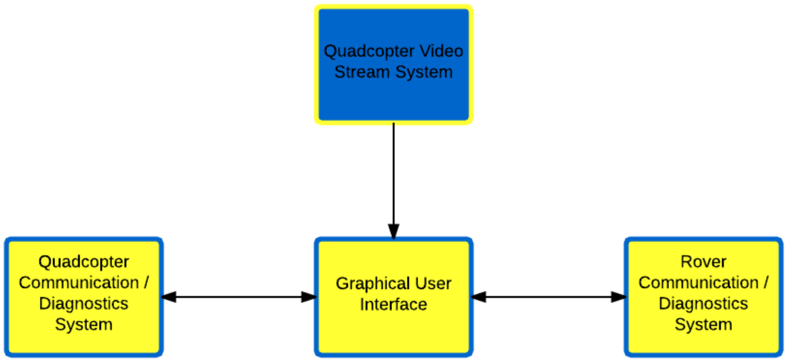

Quadcopter Video Stream System
Quadcopter Communication / Diagnostics System
Graphical User Interface
Rover Communication / Diagnostics System

Figure 3-5: Android Application

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).

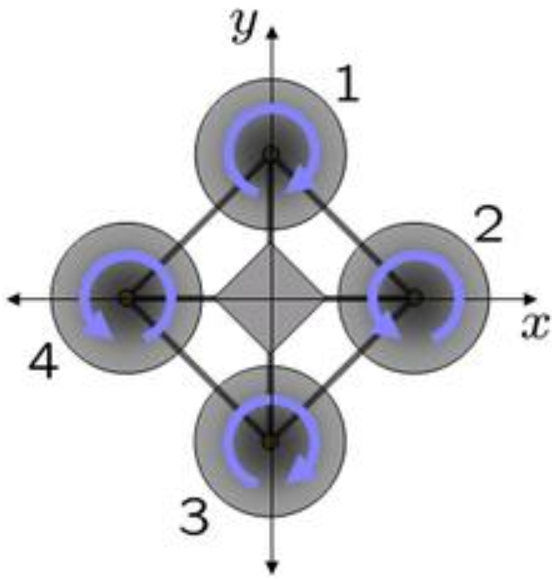


Figure 4-1: Rotational direction of quadcopter blades. Permission from Gabriel Hoffma. See Appendix B for details

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 purchases 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 will be considered are cost, customization, and the amount of time it will take to get the copter off the ground.

4.1.2 Camera

SARS will use a camera mounted below a quadcopter to identify the target object on the ground. The camera will be angled so that the field of vision is directed straight down at the ground. The camera must be capable of interfacing with a microcontroller so that image processing algorithms may be run on the individual frames from the video feed. The camera must produce high enough quality images that a reasonably large, brightly colored object may be clearly identified from an approximate height of 10 ft. The camera must have stabilization software or a stabilized mount so that the video feed is clear. SARS may include video streaming from the camera to an Android application. The final decision as to whether or not this functionality will be included will be based upon the relative ease of interfacing the camera with the Android device as well as upon the extra power consumed by the video stream.

The three cameras that will be the most suitable for SARS are 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 is 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 of whether or not to use the GoPro to implement SARS will be based upon but not limited to these factors.

Pros:

- Wi-Fi Communications; may interface with a BeagleBone
- Weatherproof
- 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 960p video resolution will be used because of its ultra wide field of view. Having a wide field of view will facilitate the detection of the target object. The determination of the number of frames per second will be made based on speed testing of the OpenCV object detection application.

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 GoPro engineering testing.

	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 will allow SARS to interface between the GoPro camera and a BeagleBone via Wi-Fi communications. This project can open the camera's live stream and save a single frame using OpenCV. Item detection algorithms can then be run on the images saved to the BeagleBone, enabling SARS to identify the target object.

4.1.2.2 Raspberry Pi Camera Module

The Raspberry Pi Camera Module is 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 of whether or not to use the Raspberry Pi to implement SARS will be 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

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

header 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. If the Raspberry Pi and the Raspberry Pi camera module are selected as the final board and camera for SARS, more research will have to be done to determine which API will best serve the purpose of accessing still photos or individual frames from a video stream to be processed for item detection.

While no prefabricated quadcopters include mounts for a Raspberry Pi camera module, building an inexpensive, stabilized mount is still a possibility. According to the article, "'Vibration Free' Camera Mount" on Flite Test, such a mount can be built using only the following materials:

- 3mm plywood
- 1.5mm fiberglass sheet
- 6mm silicone fuel tubing
- M3 thread 25mm long
- 30 minute slow cure epoxy
- Strong self-adhesive Velcro
- Foam rubber material

While having to assemble this mount or a variation of this mount would not be ideal, if the GoPro Hero3 does not end up as the final selection for SARS, the lack of prefabricated mounts should not prevent the use of the Raspberry Pi camera module. Considering this obstacle, however, the Raspberry Pi camera module does not seem to be the ideal candidate.

4.1.2.3 HTC RE Camera

The HTC RE Camera is the third camera under consideration for the SARS image processing subsystem. Listed below are the pros and cons of using this camera. The decision of whether or 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. While 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 team felt that it merited at least some research.

4.1.2.4 Final Camera Decision

After some discussion, the SARS Group has 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 is confident that this setup will yield high quality images for object detection. The compatibility of the GoPro with the BeagleBone Black is also a major convenience as the BeagleBone can handle all of the necessary image processing.

4.1.3 Image Processing

SARS will use OpenCV to detect the object once the BeagleBone has access to the GoPro images. OpenCV is an open source framework with C++, C, Python and Java interfaces. It supports Linux; therefore, it is compatible with the BeagleBone, which is our top choice for our quadcopter microcontroller. The code for the image processing will be written in Python to keep it consistent with the open source code found which will allow the BeagleBone to access the GoPro via Wi-Fi; however, because no one on the team has significant experience with Python, if learning the language proves problematic, the SARS Group will revert to Java, with which all group members have experience. Matt Bahr will be primarily responsible for this SARS subsystem, and Brian Crabtree will assist him in its implementation.

OpenCV allows for two different methods of object detection: Cascade Classification and Latent SVM. The team has 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.

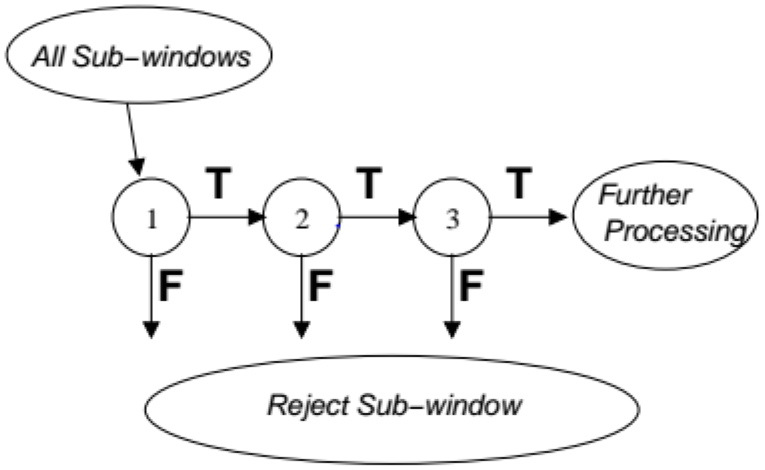


Figure 4-2: Cascade Classifier

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.

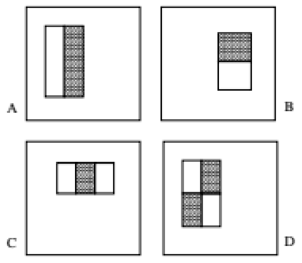


Figure 4-3: Classifier Features

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.

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

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

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

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.

4.1.4 GPS Module

A GPS module must be mounted on both the quadcopter and on the ground rover. The quadcopter must be capable of relaying its GPS coordinates to the rover once it finds the target object, and the rover must be capable of travelling to those coordinates. The two subsystems have slightly different requirements of their respective GPS modules. The quadcopter module must be small and lightweight, and it must consume little power. It only needs to retrieve GPS data once it is hovering directly over top of the target object, and it must be capable of interfacing either with a BeagleBone Black or with a Pixhawk. Because the quadcopter module only needs to run when the object has been found, it does not need to start until the GPS coordinate needs to be sent. This will prevent wasted power; however, it requires that the unit have a low startup time. The rover module does not have the size or power limitations as the quadcopter module. The main concern with the GPS for the rover subsystem is that it be capable of tracking an object in motion; however, as the rover will not be required to move at high speeds, this should not be an issue. Adafruit and Sparkfun both produce GPS units which meet these requirements. The other option being considered for SARS is the Ublox LEA-6H module, which comes included with the ArduCopter microcontroller.

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, which will be more than sufficient to track the movements of the land rover. Finally, the built-in microcontroller supports datalogging; it stores GPS information in its FLASH memory. This functionality will be useful for debugging purposes. 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 is that it comes included with one of the quadcopter kits being considered for SARS. It has an APM compatible 6-pin DF13 connector which will allow 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 will be 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 facts that it comes with the Pixhawk used to run ArduCopter was the critical factor in the decision to use this module.

The Adafruit Ultimate GPS Breakout shall 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 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. The Ultimate GPS Breakout should 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, a 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.

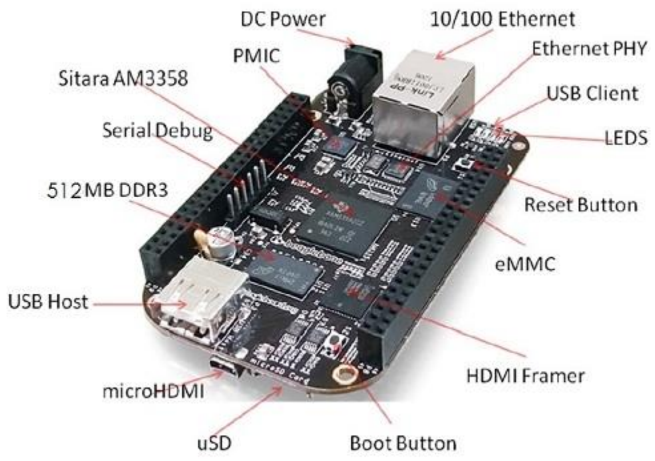


Figure 4-7: BeagleBone Black. Permission from jkridne. See Appendix B for details.

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^{2}C, 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.

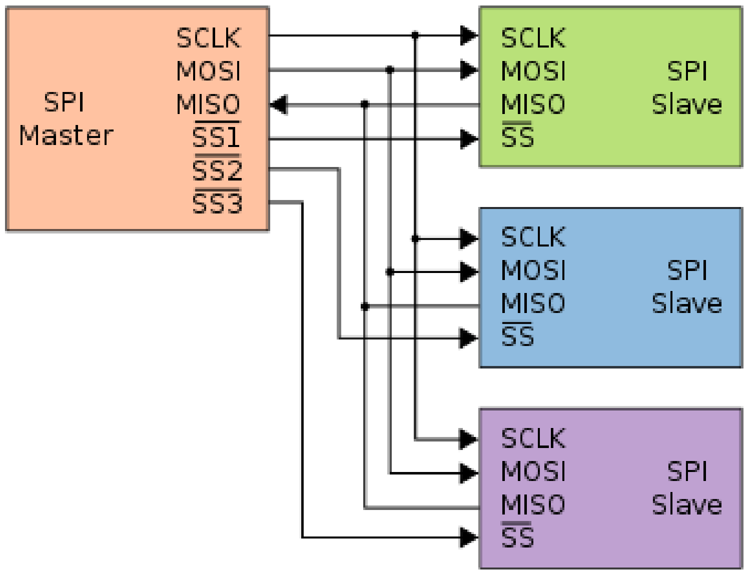


Figure 4-8: Example of Typical SPI Bus. Permission from en:User:CBurnett. See Appendix B for details

I^{2}C (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.

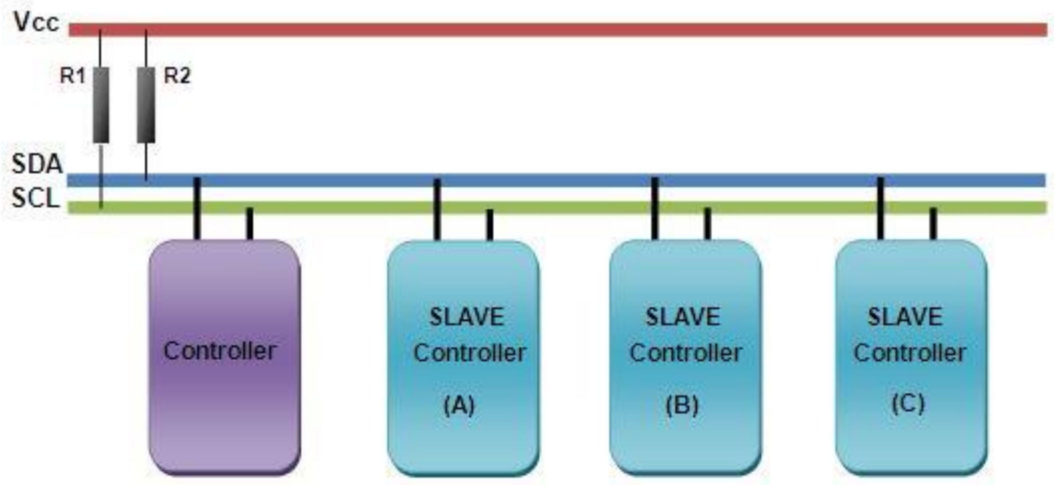


Figure 4-9: I2C communication protocol. Permission from www.engineersgarage.com. See Appendix B for details.

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:

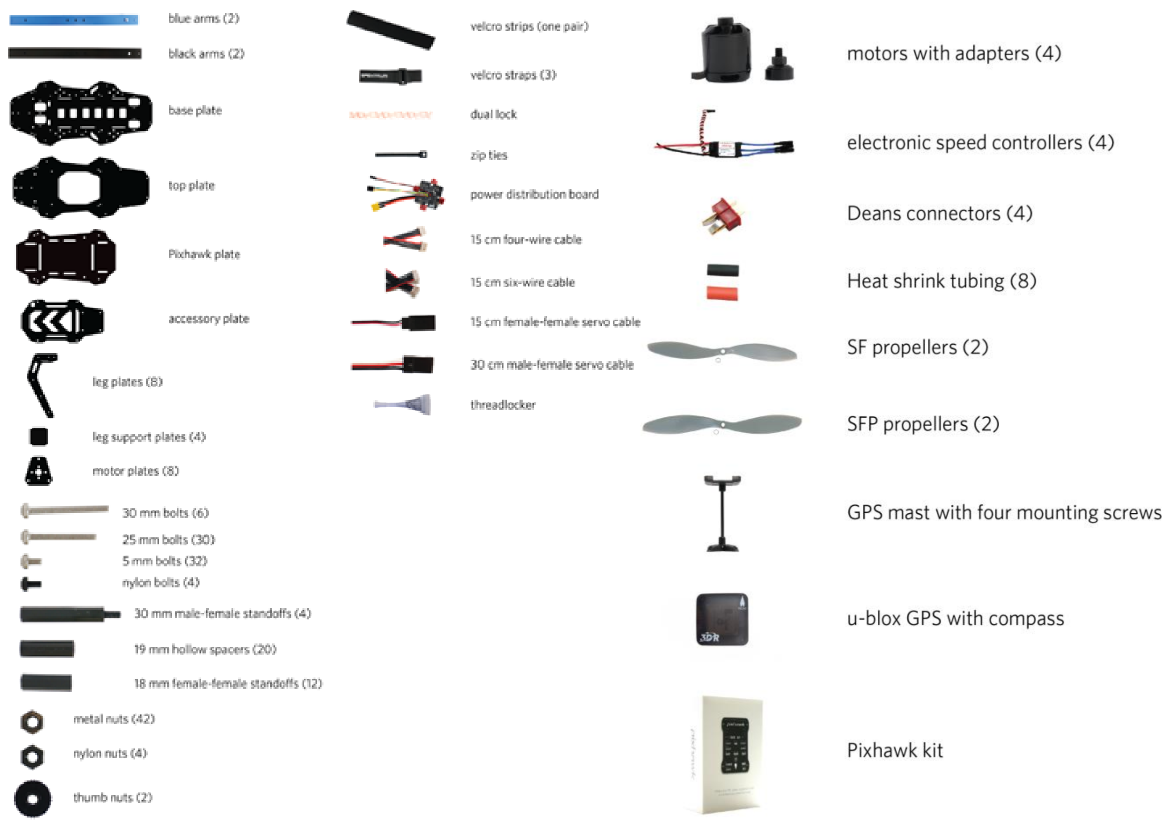


Figure 4-10: Quad kit components

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 will allow us to determine the proper travel speed of the rover. It might also be 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 of their own. 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 will be slightly cheaper and uses less battery power. For the SARS rover, a 4 wheel drive system will 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 will be possible and motion will be much smother. This will also be 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 is 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 (Android app or quadcopter), and transmit data to the Android 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 are many different approaches to picking up objects with a robotic grasping apparatus. 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 will be 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 minimal effects on the rover. Most of the required hardware can be stored directly on the chassis. The heaviest component of the system is the vacuum pump which will be housed on the chassis. The actual retrieval apparatus will consist 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 will need servos to control its motion as well. The arm must have roughly 90 degrees of up and down motion where it connects to the rover and at the end of the arm where the gripper is, the hand, must have 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 in on the rover. This will be further discussed in the rover design.

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.

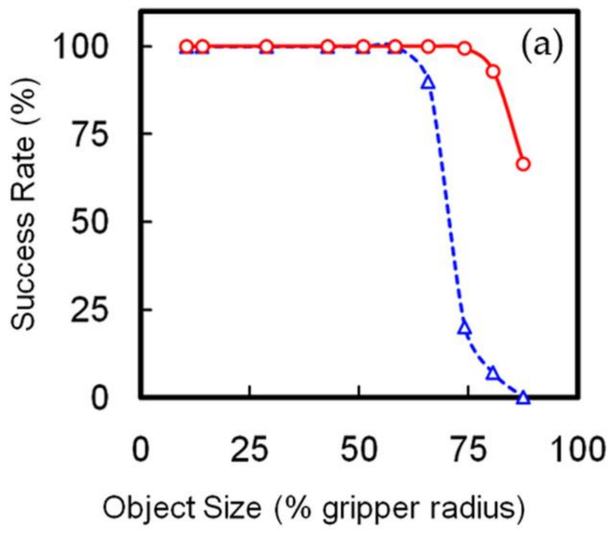


Figure 4-11: Grabber gripping success

Another option, should the Universal Gripper prove too difficult to implement, would be to use an electromagnet to lift some metal object. This magnet could be mounted on the end of a robotic arm and lowered overtop of the target object. If this ends up being the final retrieval method implemented for SARS, the target object would need to be modified since a tennis ball cannot be lifted with a magnet. In this case, a large, brightly colored sheet would be placed on the ground so as to facilitate aerial detection. On top of this sheet a small, three-dimensional metal object would be placed. This is the object for which the rover will be searching. Upon detection, the arm would lower, the electromagnet would be turned on, and the object would be 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. The SARS Group has not yet decided specifically what 12-V servo motor shall be used for each degree of freedom. This decision shall be made once Group 4 has a better idea which object retrieval prototype shall be implemented for SARS.

More than likely, if a magnet is 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.

Weight and moment calculations, among other kinematic measurements, will need to be performed for each degree of freedom of the robotic arm. If the team decides to use an object retrieval method that requires a robotic arm, these calculations shall be included in the design segment of this document.

Since the robotic arm will only be performing one function, it does not need extra sensors for finding the target object besides those already mounted on the rover. Essentially, the arm only needs 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 may end up being a waste of time and resources.

If the robotic arm is too involved for Group 4 to build in addition to all of the other SARS subsystems, a simpler design may be implemented. An electromagnet may be mounted underneath the chassis of the SARS Rover and turned on as the rover travels over top of the target object.

The decision of which retrieval method is best for SARS has yet to be made. The scope of this subsystem is entirely dependent upon the difficulty in implementing the other SARS subsystems, so the decision will more than likely be postponed until January 2015 when the SARS Group has had more of an opportunity to work on the image processing, geolocation, flight dynamics, and rover object avoidance and detection elements of the project development. Limited design materials shall be included in this document for each of the proposed methods outlined above.

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.

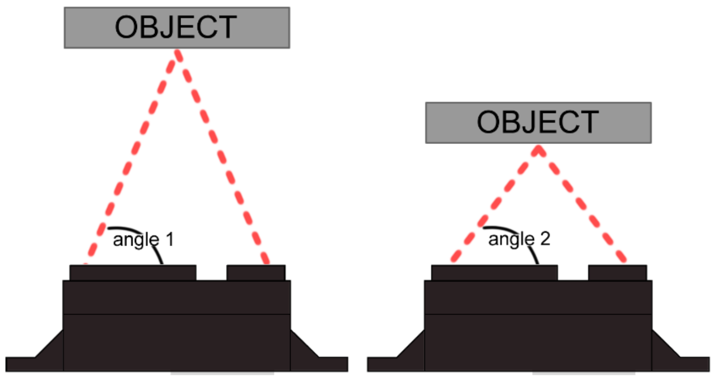


Figure 4-12: Triangulation with Infrared Sensors
Image Courtesy of ROBOTC

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.

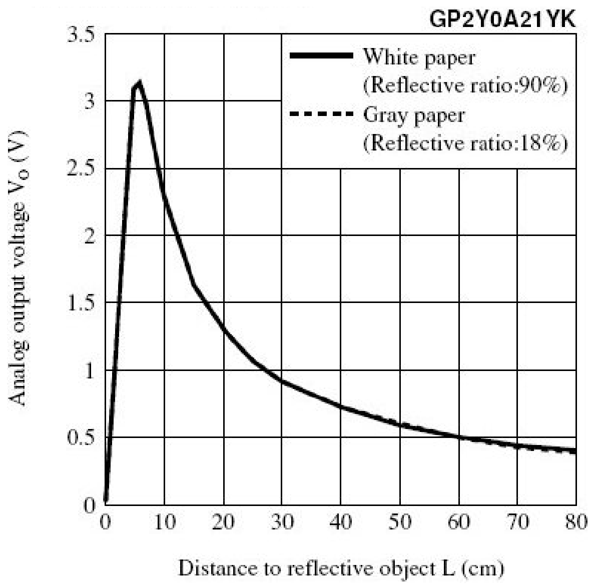


Figure 4-13: Sharp GP2Y0A21YK V-L Relationship
Reprinted with permission from Sharp (Pending)

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.

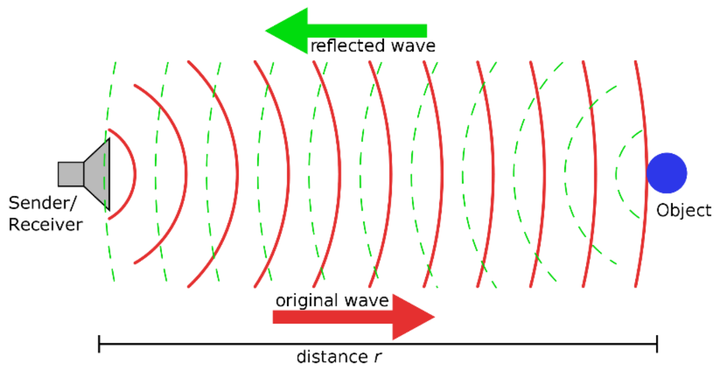


Figure 4-14: Ultrasonic Ranging
Image Courtesy of Wikipedia

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.

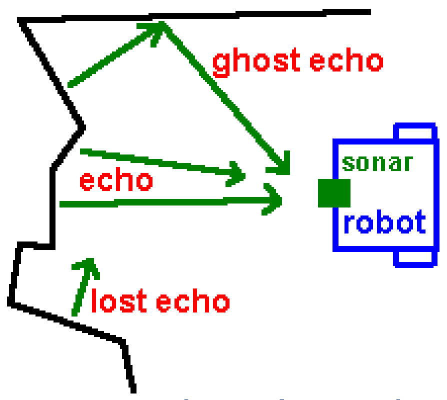


Figure 4-15: "Ghost" and "Lost" Echoes

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	100 to 200	200 to 300	<100	150 to 250	200 to 300	200 to 2000
(includes peripheral circuits) in mΩ	6V pack	6V pack	12V pack	7.2V pack	7.2V pack	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	20C	5C	5C	>2C	>2C	0.5C
- best result	1C	0.5C or lower	0.2C	1C or lower	1C or lower	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

The Android application is a critical component of SARS. It will serve as the communications hub through which the quadcopter and rover will report their current status and system diagnostics. The application will also be responsible for sending the commands to both begin a search and terminate a search early. It will also provide a live video stream of the camera mounted to the quadcopter so that users have a real time view of the SARS system as it performs its search and retrieve operation.

4.3.1 Hardware

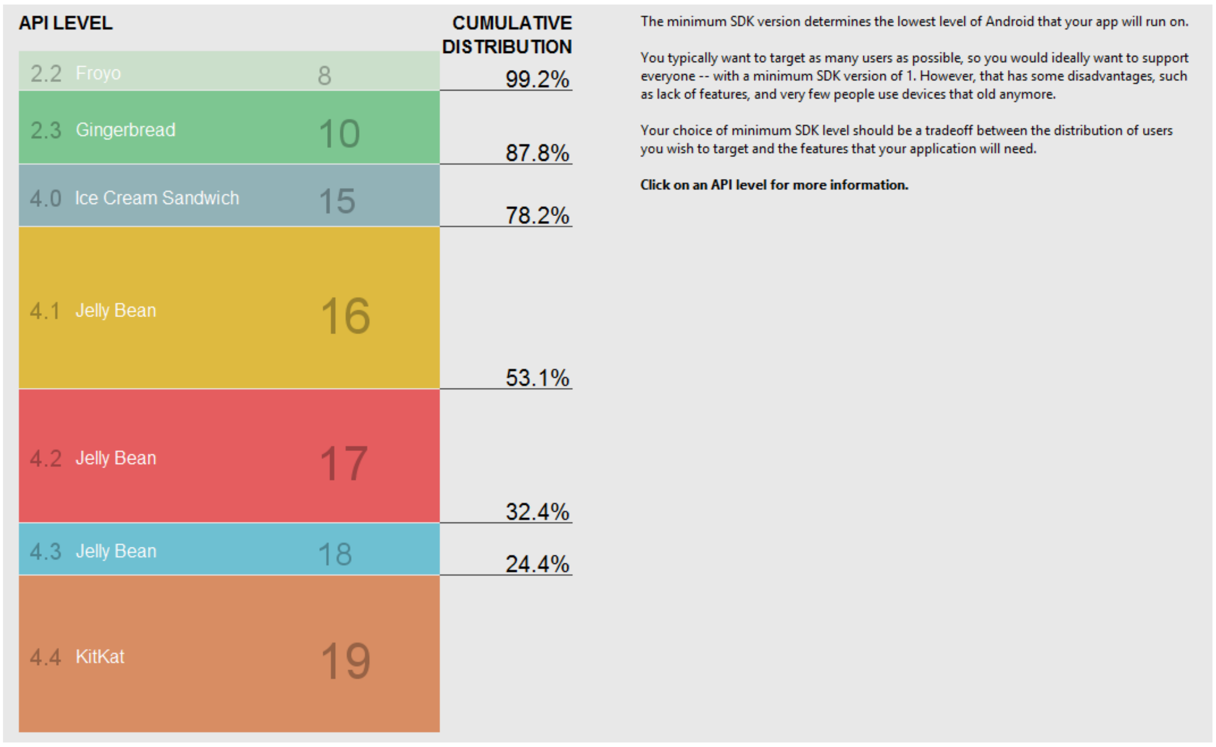
The Android application will be initially designed for use with a Nexus 5 smart phone which is part of Google's Nexus family of Android devices. The Nexus devices are designed, developed, marketed, and supported by Google but manufactured by original equipment manufacturers (OEMS) such as Samsung and 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 proves successful on the Nexus 5, and ample time is allotted after project completion, the team may decide to also support SARS on tablets. In this case, a Nexus 10 tablet will be 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 will be 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 is a prototype application with very specific purposes, the team has decided to develop 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 will be targeted only for these Android versions. As with device support, if software development with API 19 proves successful, and ample time is allotted after project completion, the team may decide to lower the target API down to version 14, which will support all devices running Android version 4.0 or higher.



The minimum SDK version determines the lowest level of Android that your app will run on.

You typically want to target as many users as possible, so you would ideally want to support everyone -- with a minimum SDK version of 1. However, that has some disadvantages, such as lack of features, and very few people use devices that old anymore.

Your choice of minimum SDK level should be a tradeoff between the distribution of users you wish to target and the features that your application will need.

Click on an API level for more information.

Figure 4-16: Android SDK API Level Distribution

4.3.3 Development Environment

The Android application will be 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 is 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 is important to the SARS team as the application is 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 Wireless Communication

4.4.1 Xbee

One of the wireless communication technologies that has 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 4-17. 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.

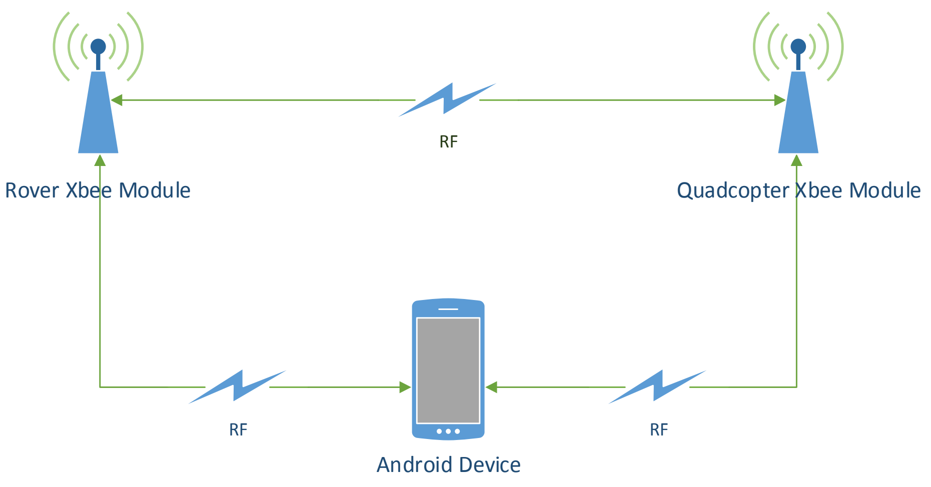


Figure 4-17: Ad-Hoc Topology

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 4-18.

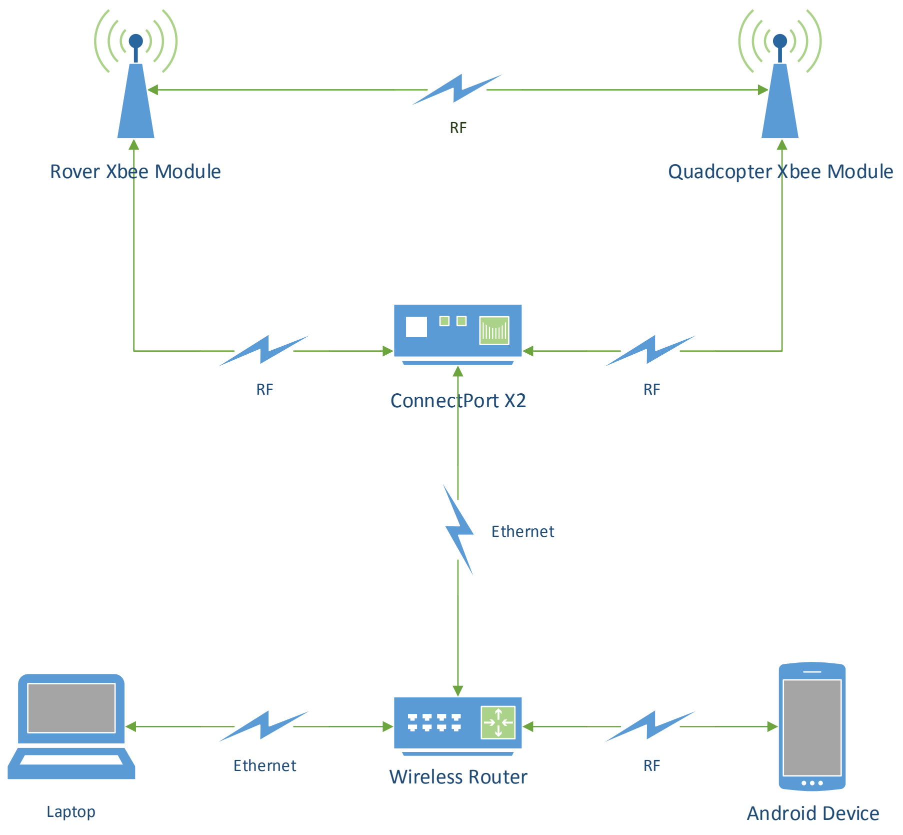


Figure 4-18: Gateway Topology

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.

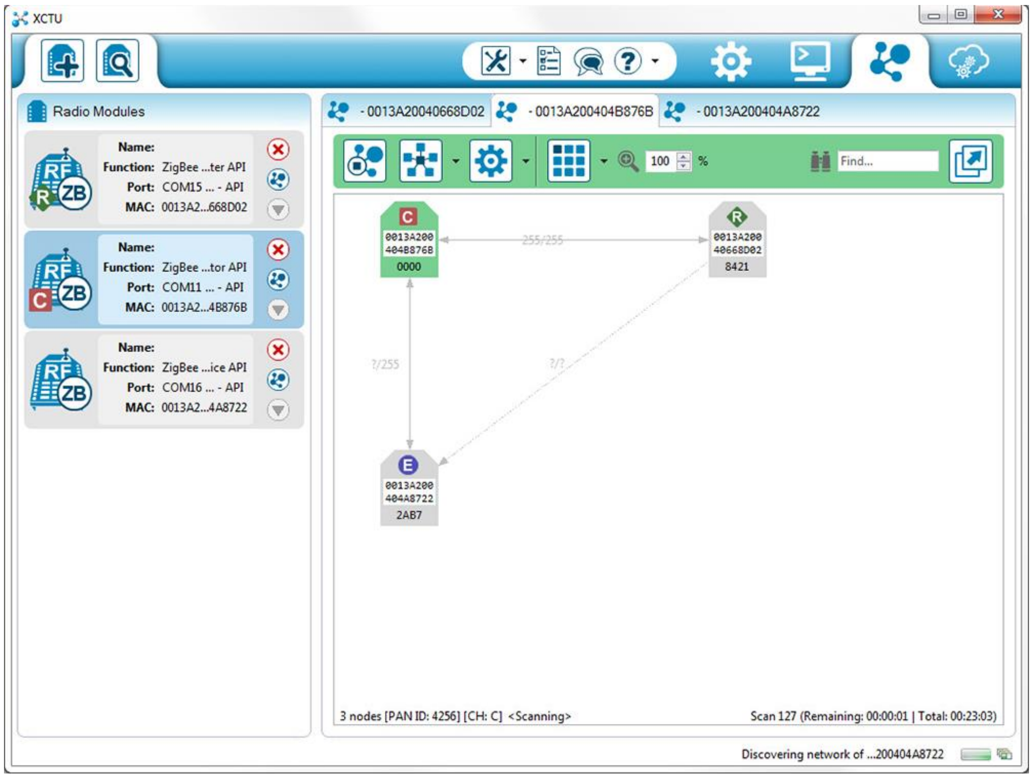
Digi ConnectPort X2	
Cost	$135.00
Device Management Interface	HTTP/HTTPS Web Interface Device Cloud Management Interface
Protocols	TCP, UDP, DHCP, SNMPv1
Wireless Interfaces	802.15.4, ZigBee, DigiMesh 2.4, 900HP, XSC, 868
Ethernet Interfaces	ZigBee, 802.15.4, DigiMesh (XBee-PRO 900HP)
Supply Voltage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

Table 4-13: Digi ConnectPort X2 Specifications

Group 4 is currently favoring the first scenario. While the extensibility of the second scenario would be beneficial, the simple and inexpensive setup of the first scenario is more ideal. The goal is to keep the wireless communication as simple as necessary but as effective as possible.

4.4.2 X-CTU

Another advantage to using the Xbees is that they are easy to set up using Digi's X-CTU configuration software. X-CTU is a free, multiplatform application that allows users to quickly and easily configure Xbee modules for deployment in an Xbee network. X-CTU includes several useful features for developers, such as the ability to visualize the Xbee network with a graphical network view that shows the topology of the Xbee network as well as the signal strength of each Xbee's connection. Using X-CTU will drastically simplify the setup, configuration, and management of wireless serial communication for both the quadcopter and the rover. This is favorable to the team as it will allow for less time to be spent on *how* the quadcopter, rover and Android device will communicate, and more time on *what* will be communicated between the various subsystems. A screenshot of the X-CTU application displaying the aforementioned graphical network view appears below in Figure 4-19.



Name:
Function: ZigBee ...ter API
Port: COM15 ... - API
MAC: 0013A2...668D02

Name:
Function: ZigBee ...tor API
Port: COM11 ... - API
MAC: 0013A2...4B876B

Name:
Function: ZigBee ...ice API
Port: COM16 ... - API
MAC: 0013A2...4A8722

C 0013A200 404B876B 0000

R 0013A200 40668D02 8421

E 0013A200 404A8722 2AB7

255/255

?/255

?/?

3 nodes [PAN ID: 4256] [CH: C] <Scanning>

Scan 127 (Remaining: 00:00:01 | Total: 00:23:03)

Discovering network of ...200404A8722

Figure 4-19: X-CTU Configuration Platform

4.4.3 SimpleLink

Another wireless communication technology that has been considered is Texas Instrument's SimpleLink WiFi solutions. The TI SimpleLink CC3100 BoosterPack 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 are 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 Android device. Table 4-14 summarizes the CC3100's specifications for reference.

TI SimpleLink CC3100	
Cost	$45.00
Serial Communications Interface	UART, SPI
Wireless Communications Standard	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

4.5 Video Streaming

In order to provide users with a real time view of SARS as it performs its search and retrieve operation, the Android 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 Android device. Once opened on the Android device, the Android 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 4-20 illustrates how this process works. Therefore, all that is needed for the Android application to access the Go-Pro’s live video feed is to have the application navigate to the Go-Pro’s HTTP server and open the M3U playlist with the native video player application.

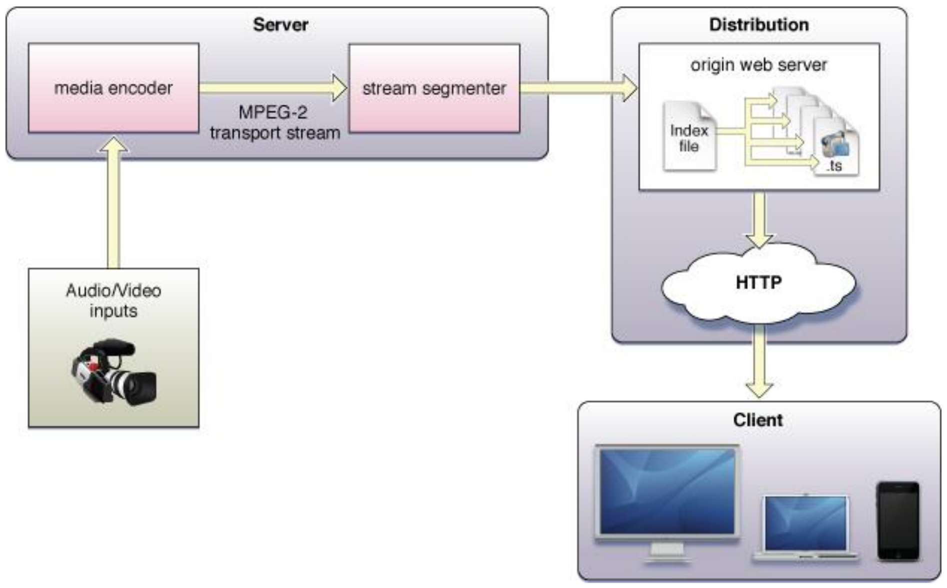


Figure 4-20: HLS Protocol
Reprinted with permission from Apple (Pending)

5 Hardware Design

5.1 Quadcopter

5.1.1 Flight Controller

The flight controller that is included in the kit we purchased is a Pixhawk autopilot module that can be controlled from the Mission Planner flight control software. This device will be mounted to the copter and connected to both the BeagleBone Black microprocessor (which will send 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:

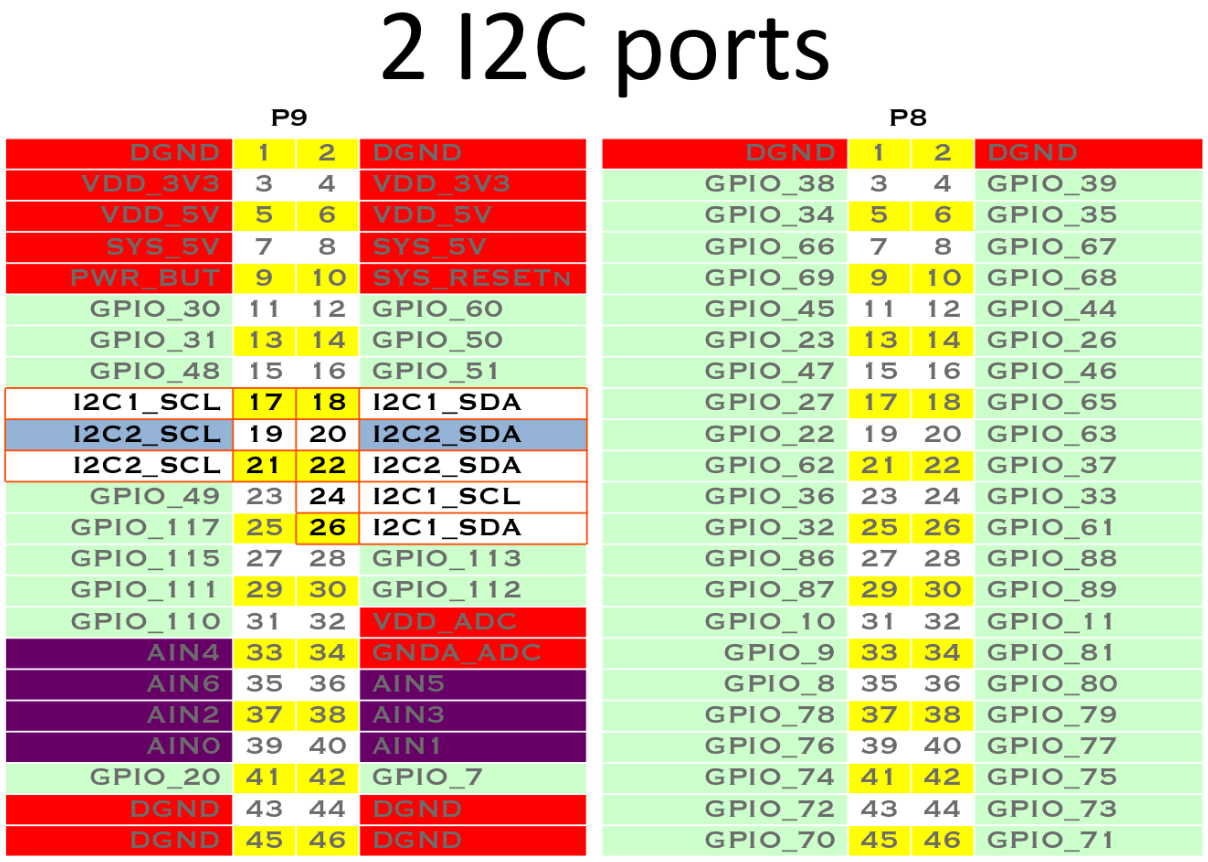
- Weight: 38g (1.31oz)
- Width: 50mm (1.96")
- Thickness: 15.5mm (.613")
- Length: 81.5mm (3.21")

The flight controller is mounted in the center of the quadcopter so as to provide and balance to the UAV while it flies. It is attached using vibration dampening foam. The foam is used to limit the vibrations felt by the accelerometer and gyroscope which are highly sensitive, and whose readings can be thrown off quite easily. Dampening foam isn't all that needs to be done though: gel pads also need 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.

5.1.2 Microcontroller Serial Communication

One of the major components of our quadcopter system is the communication of the BeagleBone black with the flight controller. After being fed the series of waypoints from Mission Planner, the BeagleBone will then begin processing all of the images fed to it by the GoPro. Once, the image is detected, the BeagleBone MCU will need to signal to the Pixhawk that the waypoints can be ignored from then on, and the copter needs to instead navigate over to the object and then return to base.

The easiest way to connect the two devices using the I^2C serial interface protocol. The BeagleBone has three I^2C buses: i2c0, i2c1, and i2c2. i2c0 is not exposed in the expansion headers, so the bus i2c1 will be used for our project. The BeagleBone has 2x46 pin headers (Figure 5-1), and the location of the pins used for the i2c1 bus are pins 17 and 18 on the P9 header. Pin 17 is I2C1_SCL (the serial clock for the I^2C bus) and pin 18 is I2c1_SDA (the serial data line). Below is a layout of the pins for the P8 and P9 headers.


2 I2C ports

P9				P8			
DGND	1	2	DGND	DGND	1	2	DGND
VDD_3V3	3	4	VDD_3V3	GPIO_38	3	4	GPIO_39
VDD_5V	5	6	VDD_5V	GPIO_34	5	6	GPIO_35
SYS_5V	7	8	SYS_5V	GPIO_66	7	8	GPIO_67
PWR_BUT	9	10	SYS_RESETN	GPIO_69	9	10	GPIO_68
GPIO_30	11	12	GPIO_60	GPIO_45	11	12	GPIO_44
GPIO_31	13	14	GPIO_50	GPIO_23	13	14	GPIO_26
GPIO_48	15	16	GPIO_51	GPIO_47	15	16	GPIO_46
I2C1_SCL	17	18	I2C1_SDA	GPIO_27	17	18	GPIO_65
I2C2_SCL	19	20	I2C2_SDA	GPIO_22	19	20	GPIO_63
I2C2_SCL	21	22	I2C2_SDA	GPIO_62	21	22	GPIO_37
GPIO_49	23	24	I2C1_SCL	GPIO_36	23	24	GPIO_33
GPIO_117	25	26	I2C1_SDA	GPIO_32	25	26	GPIO_61
GPIO_115	27	28	GPIO_113	GPIO_86	27	28	GPIO_88
GPIO_111	29	30	GPIO_112	GPIO_87	29	30	GPIO_89
GPIO_110	31	32	VDD_ADC	GPIO_10	31	32	GPIO_11
AIN4	33	34	GNDA_ADC	GPIO_9	33	34	GPIO_81
AIN6	35	36	AIN5	GPIO_8	35	36	GPIO_80
AIN2	37	38	AIN3	GPIO_78	37	38	GPIO_79
AIN0	39	40	AIN1	GPIO_76	39	40	GPIO_77
GPIO_20	41	42	GPIO_7	GPIO_74	41	42	GPIO_75
DGND	43	44	DGND	GPIO_72	43	44	GPIO_73
DGND	45	46	DGND	GPIO_70	45	46	GPIO_71

Figure 5-1: BeagleBone I2C ports. Permission from jkridner. See Appendix B for details.

When connecting to the device though, sometimes the actual name of the device will not be mapped correctly to its physical name so it is also important to know the memory address of each of the buses. The memory address of i2c0 is 0x44E0B000, i2c1 is 0x4802A000, and i2c2 is 0x4819C000. The BeagleBone for this project is set up using an embedded Ubuntu environment, so it is possible to detect the names of the I^2C buses using the ‘ls’ Linux command.

Both the BeagleBone Black and the Pixhawk operate at 3.3V, so there is no need for power conversion between the two or the addition of any level-shifting devices. Below is a schematic diagram demonstrating the wiring between the BeagleBone and the Pixhawk.

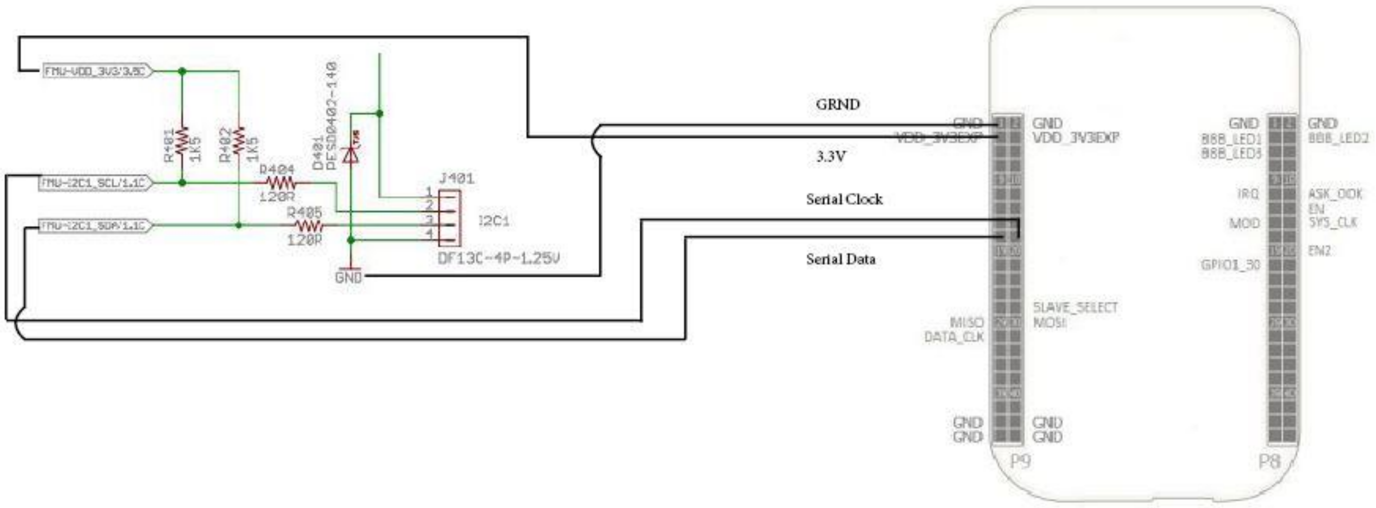


Figure 5-2: Wiring diagram between BeagleBone and flight controller

5.1.3 GPS

The u-blox GPS (summarized below) can be 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^{2}C splitter module. The GPS module has 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 must be an acceptable distance from any interfering magnetic fields so as not to interfere with its readings. The GPS should face the sky clearly and point towards the front of the copter. The device runs at 3.3 V so it is consistent with the other microcontrollers on the copter.

<u>U-Blox GPS:</u>

- 5 Hz update rate
- 25 x 25 x 4 mm ceramic patch antenna
- Rechargeable 3V lithium battery pack
- Low noise 3.3 V regulator
- I^{2}C 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 will be connected to the I^2C splitter module, it will be enable to communicate simultaneously with both the BeagleBone and the Pixhawk. This is helpful because no additional wiring is needed in order to connect the BeagleBone to the GPS.

5.1.4 Power Source

As mentioned in the specifications of the Pixhawk listed above, there are 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 are 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 receives input from the flight controller (which houses the accelerometer, gyroscope, and other measurement devices used to determine how to keep the copter in flight) and uses 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 it. When researching ESCS, there are several important factors to consider. The first is amp rating. The amp rating of the ESC needs to be higher than those of the props/motors or else it will overheat and die. The second is the refresh rate. Typically, updates are sent from the flight controller at a rate of 400 MHz or greater, so the ESC needs to be to deal with the speed of these updates. If the ESC is not designed for the specific copter being powered, it will need to be flashed with new firmware, a tedious process that can be voided by choosing the right one before purchase. Finally, the size and energy usage are important. If the ESC is too heavy or gives off too much heat, it may need to be cooled on-board. There are several 4-in-one ESCs that are perfect for our medium-sized copter (are able to power all four propellers at once) that draw far less power.

<u>20-Amp SimonK Electronic Speed Controller:</u>

- 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) is needed to properly power all of the different parts of the copter, many of which require different voltage levels. The PDB will draw power from a 14.8 V 4 Amp Lithium Polymer Battery (specifications below). The battery contains 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

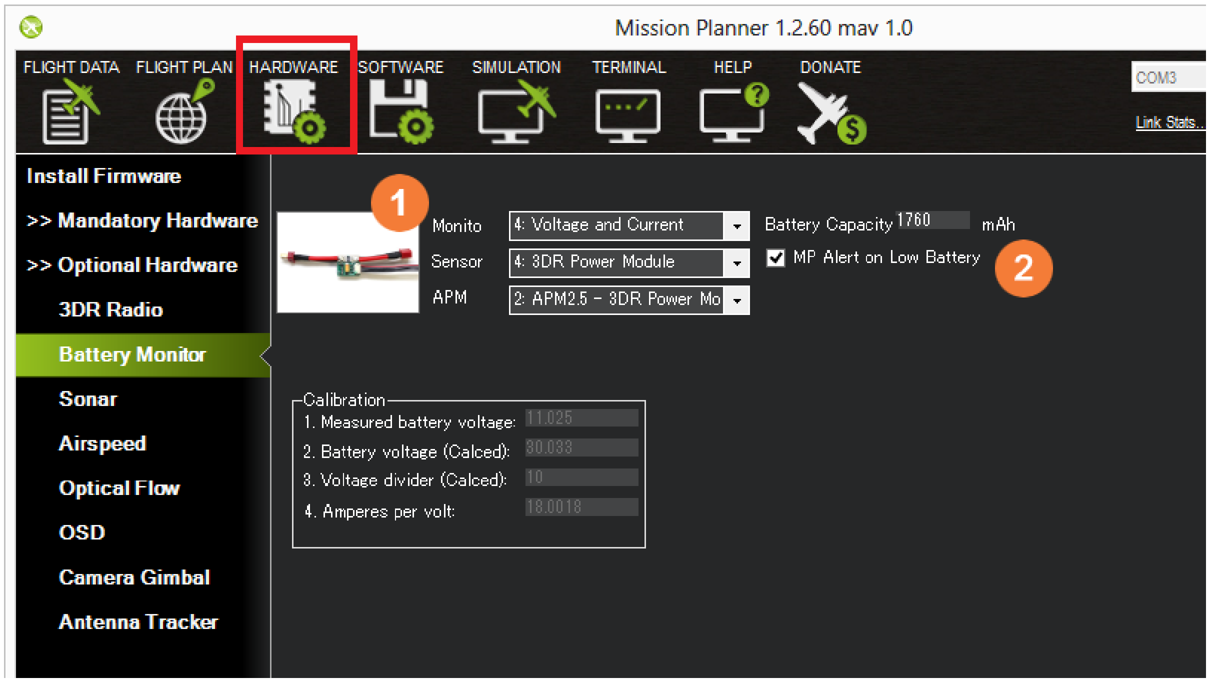
5.1.5 Quadcopter Power Distribution

In order to correctly distribute power to all the different parts of the quadcopter, the 3DR Power Module (PM) will be used as the central power distribution board. There are several advantages to using this board, the biggest of which is that it is made to efficiently communicate with the PixHawk, GPS, and motor controllers of our specific copter. The power module provides a consistent 5.3 V and 2.25 amps to the Pixhawk and also has a built-in feature where if the power level begins to reach capacity, it triggers a return-to-base safety flag that will send the copter back to its launch location to prevent over-powering the unit and possible frying the circuits. Additionally, the PM allows the firmware loaded on the quadcopter to compensate more accurately for the magnetic interference affecting the compass as a result of board and multiple ESC's. It is important to note, though, that while the PM does provide sufficient power to a lot of the copter's components, it does not directly provide power to the servo motors; these are 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 5-1: 3DR power module summary

The 3DR Power Module connects to the 4S Lipo battery we have chosen to power the copter and can 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 can be measured. The setup is very straightforward when using Mission Planner's software; it allows 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 dips below a certain power level.



Monito: 4: Voltage and Current
Sensor: 4: 3DR Power Module
APM: 2: APM2.5 - 3DR Power Mo
Battery Capacity 1760 mAh
[✓] MP Alert on Low Battery

Calibration
1. Measured battery voltage: 11.025
2. Battery voltage (Calced): 30.033
3. Voltage divider (Calced): 10
4. Amperes per volt: 18.0018

Figure 5-3: Power connection screen

When the wiring of the copter is completely finalized, the wiring diagram below will reflect the final connections between each of the components needed to control the quadcopter. As can be seen, the PDB distributes power to the Pixhawk and the 4 ESC’s, but no the propellers directly.

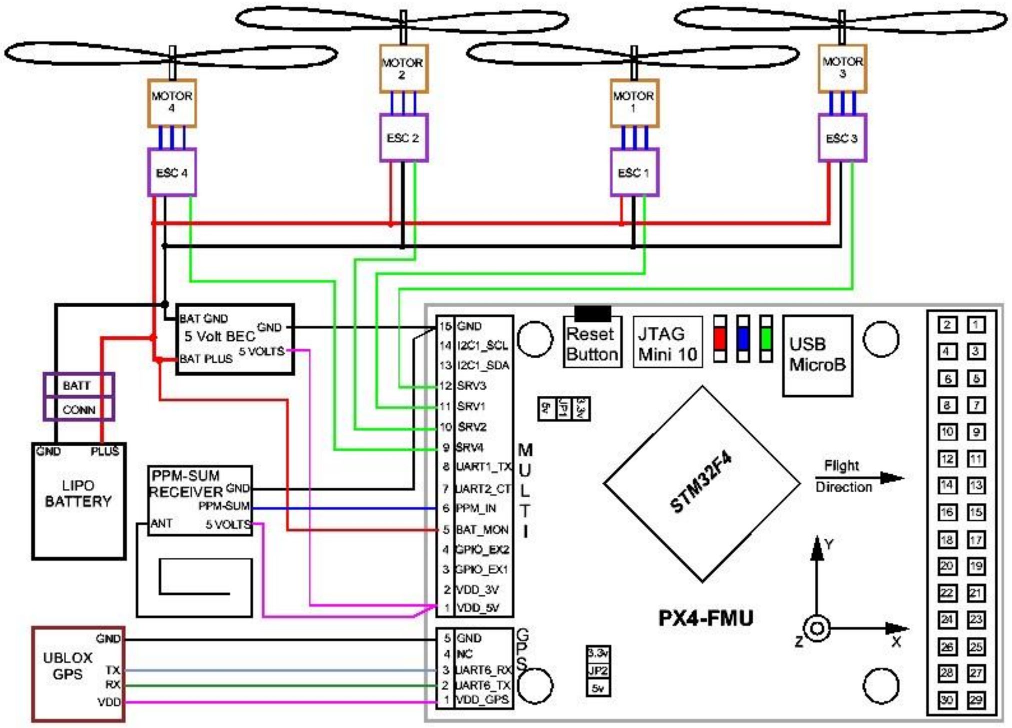


Figure 5-4: Quadcopter wiring diagram. Permission from Arducopter. See Appendix B for details.

5.1.6 Microcontroller Power Distribution

Although the primary components of the quadcopter can be powered by the PDB, the BeagleBone Black is a separate piece of hardware that is not native to the copter, and thus will have to have its own power source. There are two primary ways to power the BeagleBone: through the 5V barrel connector or through a USB input. These two methods are fed to the on-board regulators. Powering through the USB port can provide specific limitations as to the amount of power distributed, and there needs to be some runtime management to continuously check that the current running through the USB input does not exceed its threshold, and thus accurately powering the board through the USB can be a bit more tedious. As far as simplicity and ease in providing power, the 5V barrel connector is the preferred method. However, because we are powering the BeagleBone from a remote location and will be battery powered, the USB option is easier for our purposes.

In order to maximize battery life, a voltage regulator with tremendous efficiency will need to be used. A switching regulator that provides at least 5V at 1 amp is preferred, and though it may be possible to connect the board to the LiPo battery powering the rest of the quadcopter, it is easier to simply use “AA” batteries. To be able to monitor whether the BeagleBone is powered efficiently during our testing, a simple LED and push-button connection will be used to power-on the board. Thus, while the BeagleBone is powering up, the LED will flash dictating that the board is currently booting up. Once the MCU is running, the LED will be held in a solid-state. Finally, when the board is powered down, the light will turn off. One of the benefits of this system is that as we implement certain tasks for the BeagleBone black to perform during testing, we can include additional indicators to the LED to let us know it is running correctly. For example, when the BeagleBone is doing image processing, the LED can blink at a specific rate, or when the board sends the interrupt once the item has been detected, the LED could blink three times. This helps us tremendously during debugging when we are doing field testing because we will not be able to peek into the board during flight time.

The following list of materials will be needed in order to correctly power the BeagleBone:

- BeagleBone expandable case
- (1) Push-button
- (1) LED
- (1) Resistor 330 ohms
- A soldering iron and wires
- 5V switching regulator
- USB DC power adapter
- Six AA Cells with suitable holder

There are several voltage regulators that could be used. The first, the PT78ST105, is a wide-input range 3-terminal regulator that has a maximum output current of 1.5 amps and an output voltage that is tailored to several industry standard voltages: 3.3V, 5V, and 12V. It has an efficiency greater than 85%, self-contained inductor, internal short-circuit protection, and over-temperature protection. The next possible voltage regulator, the PTS84250, has a 2.5 amp DC/DC convertor, a wide input range from 7 V to 50 V, and a variable output that can be adjusted from 2.5 V to 15 V. On top of that, it has a 96% efficiency, switching frequencies between 300 kHz and 1 MHz, overcurrent and temperature protection (temperature range between -40^{o} C and 85^{o} C), and is 9mm x 11mm x 2.8 mm. The third and final voltage regulator taken into consideration, the MuRata OKI-78SR-5. The MuRata has an input range of 7V to 36V and fixed outputs of 3.3V or 5V up to 1.5 amps. In addition, it has an efficiency of 90.5%, short circuit protection, a DC/DC power converter, and is 10.4 mm x 16.5 mm. Below is a summary of the specifications of all the voltage regulators being taken into consideration. After considering all the specifications, the PT78HT205 is the best option due to its wide output voltage rage, small size, and high efficiency.

	PT78ST105	PTS84250	MuRata OKI-78SR-5
Input Voltage Range	9V - 38V	7V - 50V	7V - 36V
Output Voltage Range	3.3V, 5V, 12V	2.5V - 15V	3.3V or 5 V
Max Current	1.5 A	2.5 A	1.5 A
Efficiency	80%, 85%, 90%	96%	90.50%
Short-Circuit Protection?	Yes	Yes	Yes
Temperature Protection?	Yes	Yes	No

Size	23.9mm x 22.9mm x 7.9mm	9mm x 11mm x 2.8 mm	10.4 mm x 16.5 mm
Frequency Range	600 kHz - 700 kHz	300 kHz and 1 MHz	500 kHz
Cost	$20.00	$17.00	$4.30

Table 5-2: Voltage regulator comparison table

To power the BeagleBone, first connect the positive side of the battery to the voltage regulator input and the negative side to the ground. Then, connect the output of the voltage regulator to the USB +5V and the regulator ground to the BeagleBone ground. Below is an image of the USB converter being used to attach the regulator to the USB cord of the BeagleBone.

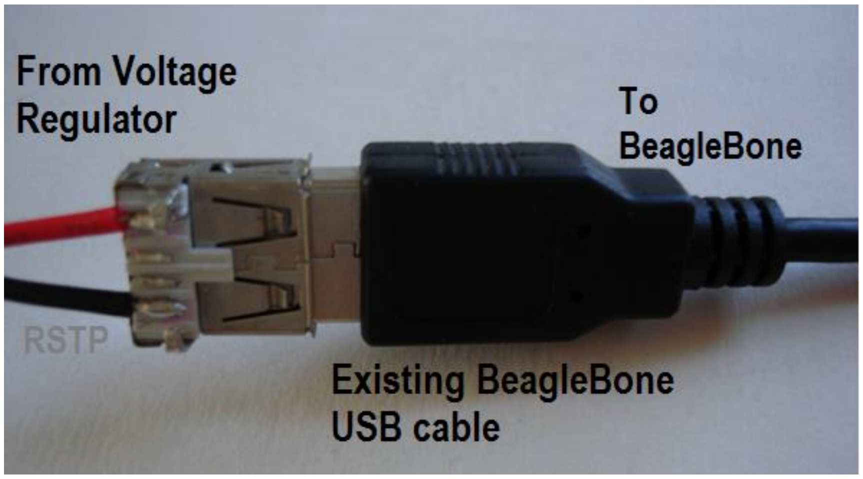


Figure 5-5: Voltage regulator to USB connection

After the connection is secure, the circuit will need to be verified before attaching the LED. The polarity should be checked and there needs to be 5V of power being provided to the board. Once the circuit is checked, the push button and LED switch can be attached to the BeagleBone. For our purposes, we will be using an OMRON A3DT-7111 push-button switch with an OMRON A3DT-500GY LED. Pin 1 of the push-button switch will be wired to P8 pin 26 and Pin 2 of the push-button switch will be connected to the 330 ohm resistor then to P8 pin 45 (ground) on the BeagleBone. For the LED, the anode is connected to P9 pin 14 and the cathode is connected to another resistor of 330 ohms and P9 pin 1 (ground). Below is a wiring diagram for the push-button switch and LED to the BeagleBone.

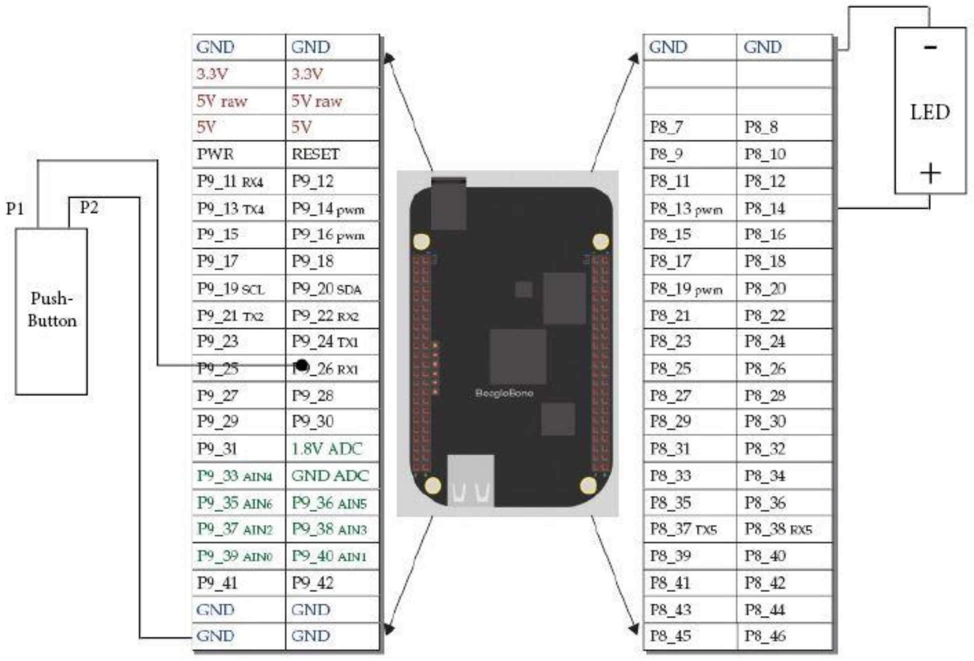


Figure 5-6: BeagleBone Black power source wiring diagram

5.1.7 Remote Controller/Receiver

Although most of the initial testing will be done inside and in a controlled environment where the quadcopter will not actually be flying, once the copter is tested in the field we will need a way of controlling it in case anything goes wrong. The copter is supposed to fly autonomously, but there needs 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 will need a remote controller 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 will be using both telemetry as well as controllers and receivers to ensure full control of the copter.

When setting up the controller, there are multiple channels that control 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 controller could work. Additional channels allow the user to control other parts of the quadcopter while I 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.

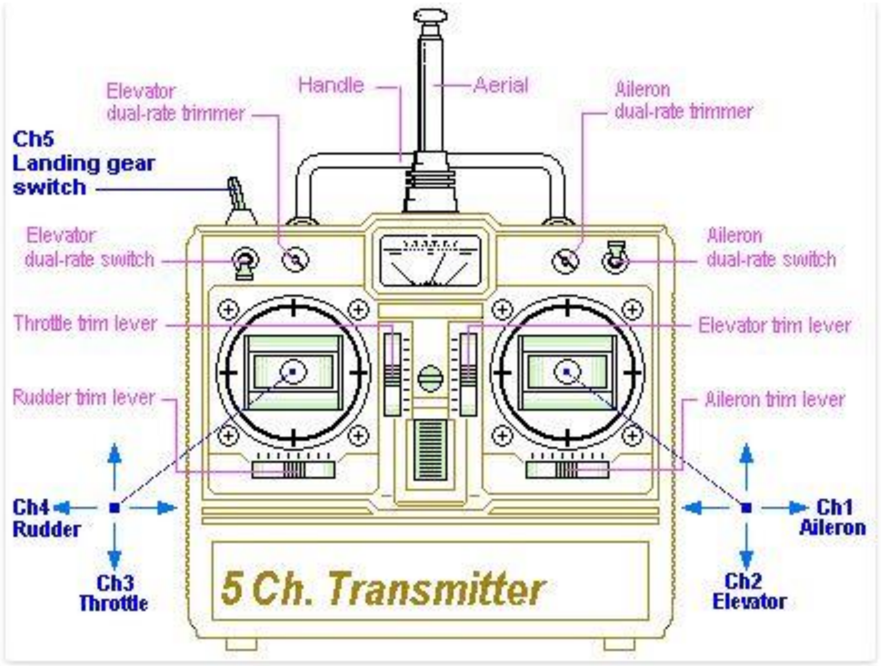


Figure 5-7: Remote controller layout example. Permission from Oscar Liange. See Appendix B for details.

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.

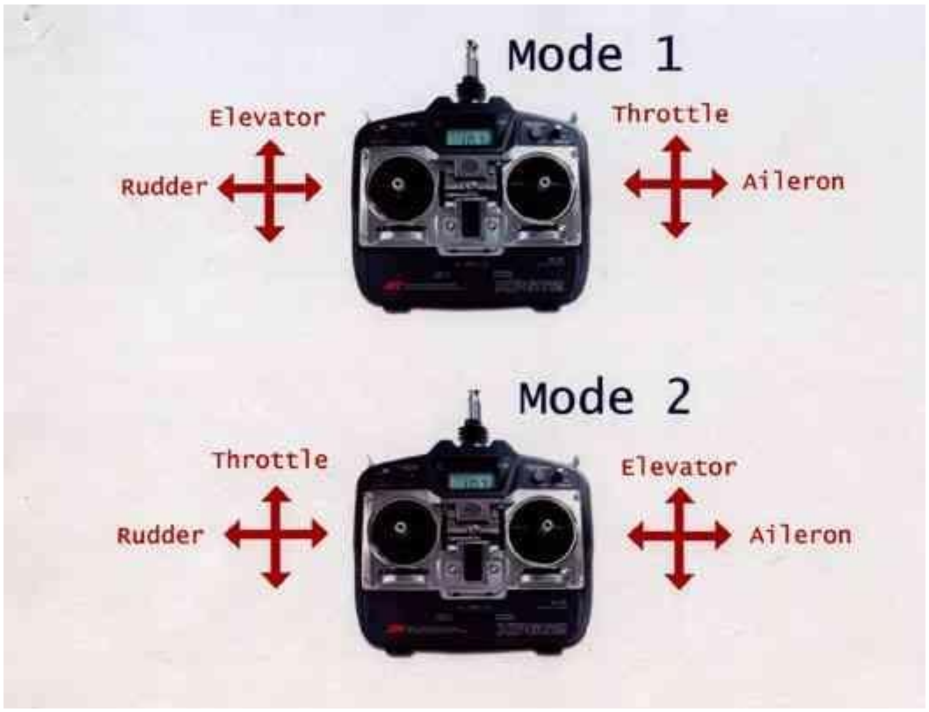


Figure 5-8: Mode One and Mode Two example. Permission from Oscar Liange. See Appendix B for details.

When deciding on one transmitter and receiver to consider purchasing, there are several key factors that need to be considered. The first is most obviously price, as just the transmitter can run from $20 to well over $1000, a huge price that does not fit our budget at all. The second, as mentioned before, is the number of channels. Arducopter recommends that the transmitter have at least six channels in order to efficiently control the copter. Other factors to consider are 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 don't want a receiver that will weigh down the quadcopter too much). Below is a table comparing the different transmitter/receiver combinations that were considered. After weighing the options, we will be going with the Turnigy 6X receiver/transmitter pairing due to the lightweight receiver, cost effectiveness, and FM modulation.

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 5-3: Remote transmitter comparison table

5.1.8 Telemetry Modules

As far as the telemetry radios, our group will be choosing the 3DR radio set, two telemetry radios that operate at 915 MHz (US standard) and allows us to communicate directly with the quadcopter via the Mission Planner. By being able to communicate with the copter, we will be 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 will be using.

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

5.2 Rover

5.2.1 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. The 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 from the rest of the rover.

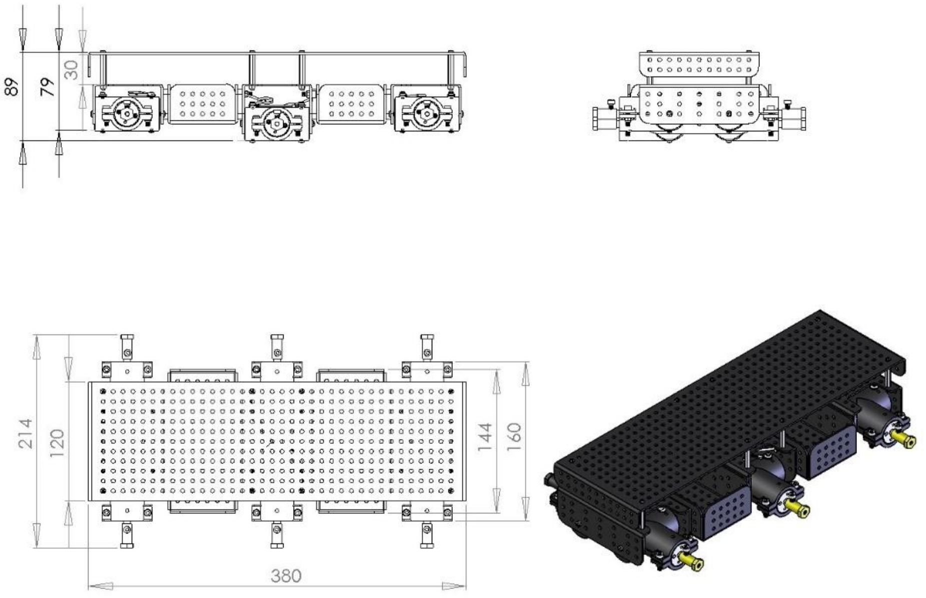


Figure 5-9: Rover chassis dimensions without wheels (Permission to reproduce pending)

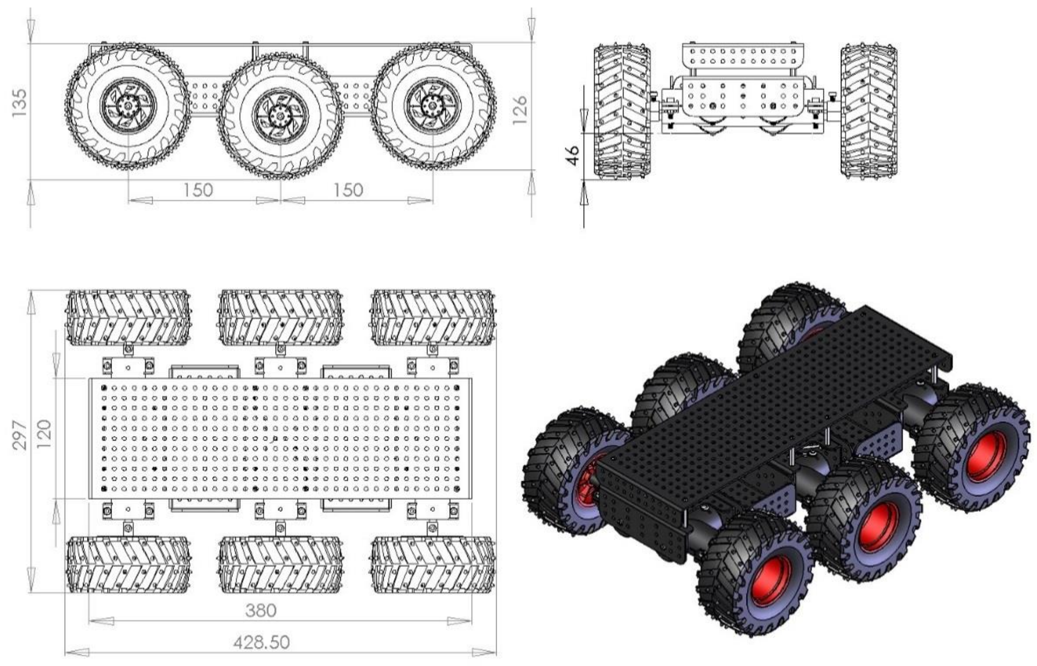


Figure 5-10: Rover chassis dimensions with wheels (Permission to reproduce pending)

5.2.2 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 34:1. The motors have an output shaft speed of 295RPM and a stall torque of 4Kg/cm. Each motor has an independent housing. The housing is designed to prevent the motors from vibrating out of place.

The motors will be wired up to a motor controller for power and control. The selected motor controller is the Dimension Engineering Sabertooth 2x25 motor controller. Table 5-4: 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 will be attached to each control channel and the motors will be 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 will act 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 the entire rover will also be connected to the motor controller which will distribute 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

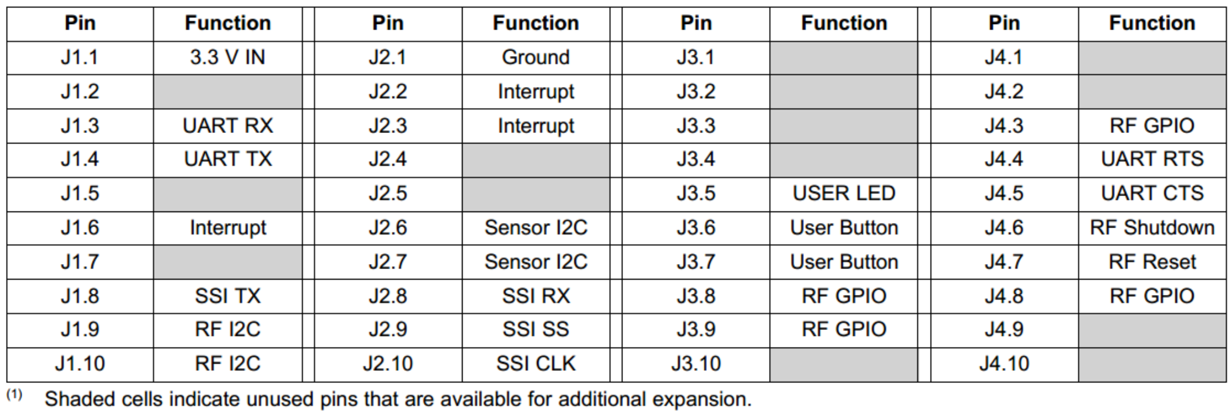
Table 5-4: Sabertooth 2X25 Motor Controller Specifications

5.2.3 Control Boards and Sensors

The control boards act as the brains of the entire rover. The control boards are 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 is 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 can be used to send commands from the Tiva C to the Sabertooth. The next important operation is interpreting input data about wheel rotation speeds. Wheel rotation will be measured using a magnet and a Hall Effect sensor. The Hall Effect sensors will be 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 will be attached to the wheels and the sensor will be made aware of how fast each of the wheels are rotating. This information will be relevant in the software design.

Hand in hand with the Hall Effect sensors is the inertial measurement unit (IMU). This unit is 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^{2}C communications across a set of 40 pins, not all of which are actually in use. These pins are illustrated in Table 5-5.



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 5-5: BOOSTXL-SENSHUB Sensor Pack pin locations (Permission to reproduce pending)

The Tiva C will also be connected to a sensor to be used to detect objects in the travel path of the rover. The SARS Rover will employ a Parallax PING))) Ultrasonic Distance Sensor. The PING))) operates using a pulse in/pulse out pair of pins. It will be powered directly from the Tiva C board. In order to maximize functionality of the ultrasonic sensor, it will be mounted on a small servo that allows it to pivot and better observe the space in front of the rover. This servo will be operated with the control software and manipulated appropriately to allow the PING))) to rotate approximately 120 degrees.

The final sensor to be used with the rover system is the GPS. The GPS module selected for use is the Adafruit Ultimate GPS Breakout. The GPS will connect to the Tiva board using UART. The Tiva C will use 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 will allow for seamless operation when approaching the target object. Relevant data for this module is available in Table 5-6.

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 5-6: Adafruit Ultimate GPS Breakout Specifications

The last remaining aspect of the control system for the rover is the communications module. The XBee communications module will connect to the Tiva C using UART. Once the XBee is configured it will allow for wireless communication with the quadcopter and Android to send and receive the necessary data. Most of the XBee functionality will be based on the software implementation of its features.

Figure 3-1 shows the overall wiring design of the rover system. It illustrates the directionality of communications between the microcontroller and the individual modules. The only module that the microcontroller will send data out to is the motor controller, the rest of the modules are primarily for receiving data from sensors or outside sources.

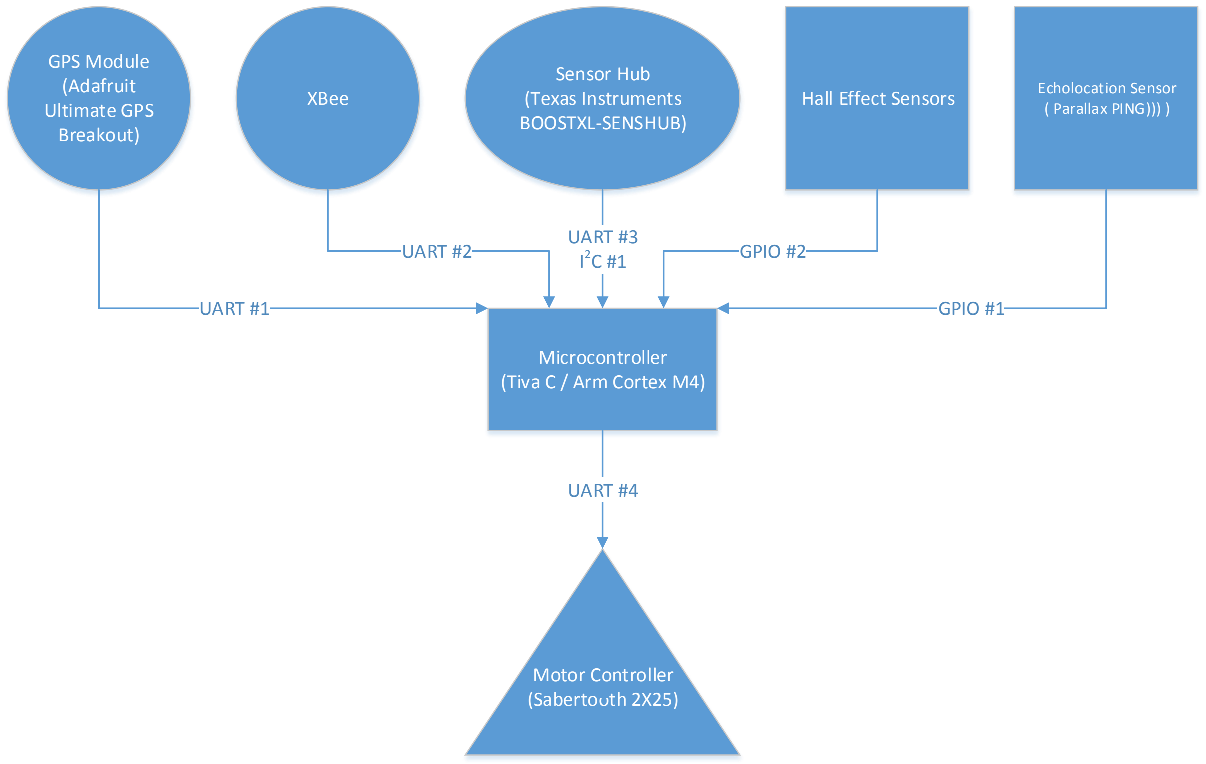

GPS Module (Adafruit Ultimate GPS Breakout)
XBee
Sensor Hub (Texas Instruments BOOSTXL-SENSHUB)
Hall Effect Sensors
Echolocation Sensor (Parallax PING))))
UART #1
UART #2
UART #3
I²C #1
GPIO #2
GPIO #1
Microcontroller (Tiva C / Arm Cortex M4)
UART #4
Motor Controller (Sabertooth 2X25)

Figure 5-11: Rover Internal Communications Wiring Diagram

5.2.4 Object Retriever

At the stage in the development of SARS, the final method of object retrieval has not yet been decided upon. Two designs are being considered for prototyping. Basic design information will be outlined for each of these potential prototypes.

5.2.4.1 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 5-12: Robotic Arm Design below.

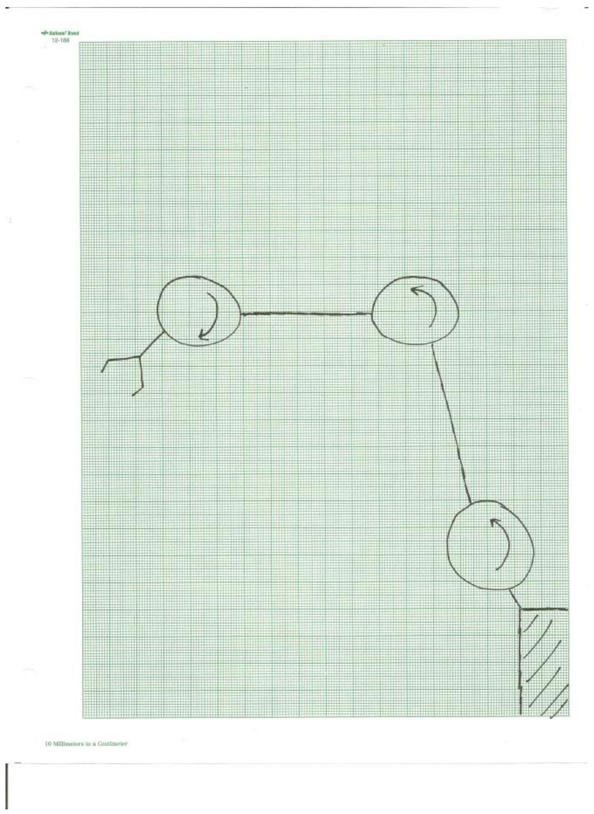


Figure 5-12: Robotic Arm Design

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 *5-13*: 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.

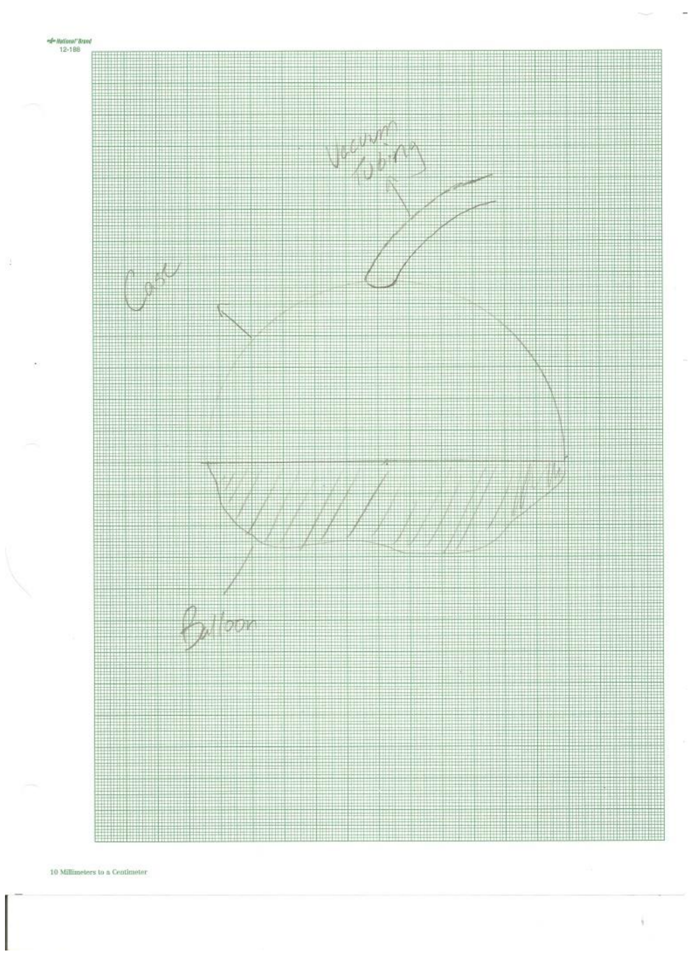


Figure 5-13: Universal Gripper Casing

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 5-14: Vacuum Pump Design.

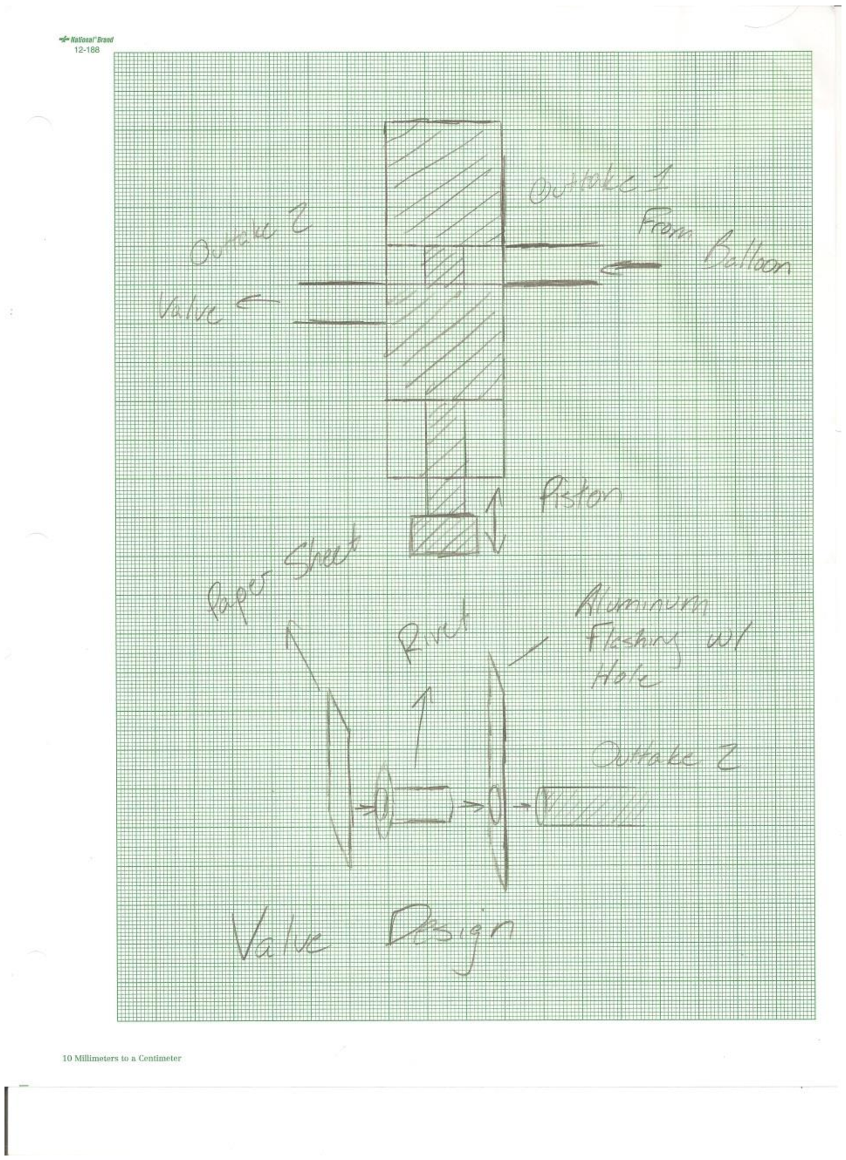


Figure 5-14: Vacuum Pump Design

5.2.4.2 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 5-12: 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.

5.2.5 Object Storage

Object storage is one of the least technically challenging aspects of the rover system, but it is still vital to the system as a whole. The apparatus employed for storing the objects must be firm enough to ensure that objects will not be unintentionally ejected as the rover traverses the terrain, but it must also be lightweight to avoid adding excess weight to the rover as a whole, because the system has a weight capacity after which the motors become encumbered. This makes a plastic bin of some sort the best option for object storage. Fortunately, the chassis is designed to allow for easy mounting of any extra equipment right onto the top of the chassis using the bountiful mounting holes already drilled into it. There is one more important consideration with regards to how the storage apparatus will be mounted onto the chassis. The storage space must be easily accessible by the object retriever in order to minimize unnecessary difficulties involved with moving the target object around while it is being held onto. The storage apparatus must also be mounted to the rover in a way such that it does not interfere with the freedom of motion of the object retrieval arm. This means leaving ample space between where the arm is mounted and where the storage is mounted. This, in turn, presents the challenge of having a large enough storage space to keep any retrieved objects contained, without having the storage space reach past the main boundaries of the body of the rover. If the storage space exceeds the main body of the rover by too much, it will cause unnecessary torque on the frame and may cause decreased mobility and performance.

5.2.6 Power

The finishing aspect of the rover hardware design is the inclusion of the power system for the rover. Power to the rover will be provided by a lithium polymer (LiPO) battery. The selected battery for this system has four secondary cells and one primary cell. It is designed to output 14.8V 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 is a reasonable weight for the battery pack for this system. At roughly half a kilogram, the battery will not cause an excessive amount of torque on the motors it is mounted over. It is 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, had 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 wider than the chassis, it must be mounted so that the long edge of the battery follows the long edge of the chassis. Once a proper mounting space has been selected, the battery can be held in place using screws. Screws along each of the edges of the battery will keep it held in place while allowing for fairly easy removal for charging.

The final aspect of power for the rover is wiring it up to the control system. The motor controller, fortunately, can handle most of the power distribution needs for the system. The Sabertooth has an integrated power stepper to allow for the 5V and 500mA needed for the microcontroller to operate. The battery will be wired directly to the power terminals of the motor controller, and the controller will take care of distribution to the motors and the control boards.

5.3 Wireless Communications

5.3.1 Quadcopter Wireless Communication

Serial communication from the quadcopter to either the rover or Android device will be achieved as follows: The Xbee S6b will be connected to the BeagleBone Black through an Xbee Cape. Wireless communication between the quadcopter and the rover and/or Android device will occur over 802.11n Wi-Fi via the Xbee module. The quadcopter's Xbee module will then route communications to the BeagleBone Black via UART. The Xbee Cape is configured to use the BeagleBone Black's UART2 by default. UART2 transmits on header P9 pin 21 and UART2 receives on header P9 pin 22. The Xbee Cape also has a dedicated 3.3V linear regulator to provide and regulate power to the Xbee. Wireless communication between the quadcopter Xbee and the rover Xbee and/or Android device will be configured using Digi's X-CTU configuration utility.

5.3.2 Rover Wireless Communication

Serial communication from the rover to either the quadcopter or Android device will be achieved with a configuration similar to the quadcopter. The Xbee S6b will be connected to the TI Tiva-C Launchpad through an Xbee SIP Adapter. Wireless communication between the rover and the quadcopter and/or Android device will occur over 802.11n Wi-Fi via the Xbee module. The rover's Xbee module will then route communications to the Tiva-C via GPIO. The Xbee SPI Adapter will be configured to connect to the GPIO pins of the Tiva. The Xbee SPI Adapter has an onboard 3.3V regulator to provide and regulate power to the Xbee. It also has 5V to 3.3V logic translator buffers so that the Xbee (which operates at 3.3V) can properly communicate with the 5V GPIO pins. Wireless communication between the rover Xbee and the quadcopter Xbee and/or Android device will be configured using Digi's X-CTU configuration utility.

6 Software Design

6.1 Quadcopter Software Design

6.1.1 Mission Planner

Because one of the goals for our system is to have quadcopter fly autonomously, we need to have a central control software that is easily integratable and allows us to monitor the telemetry of the copter during the flight. The obvious choice here is to use the Mission Planner platform, a native software/firmware integration tool that allows 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.

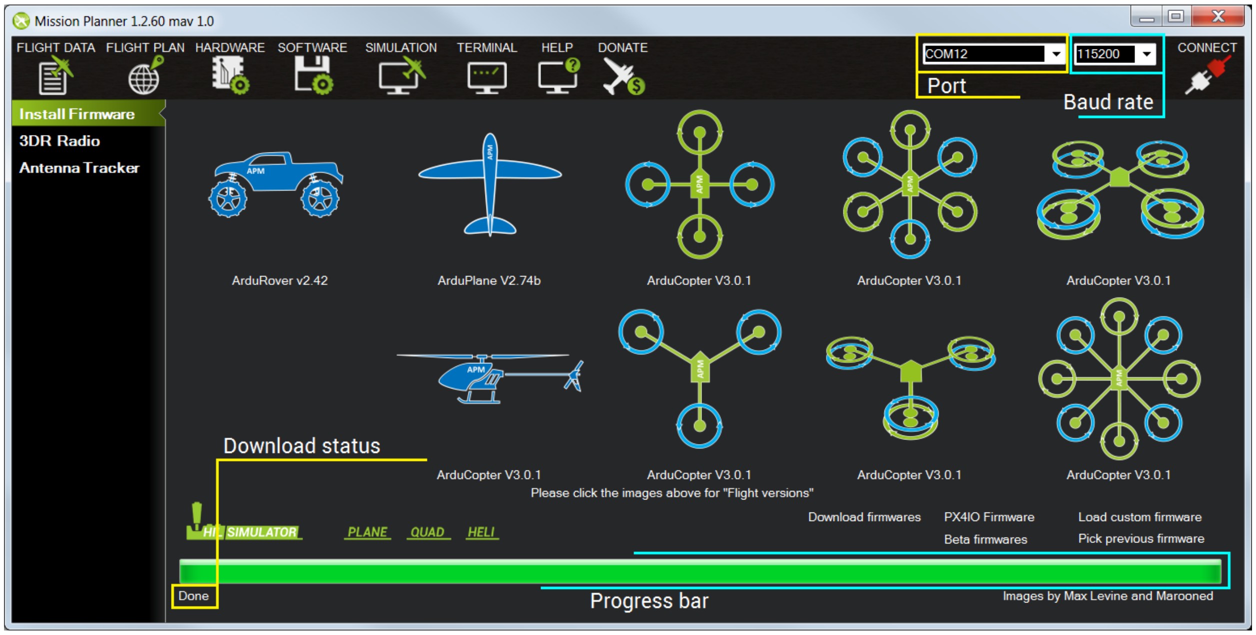


Figure 6-1: Mission Planner

Mission Planner allows 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 provides 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 provides simple calibration for all of the on-board devices

6.1.2 Image Processing Subsystem

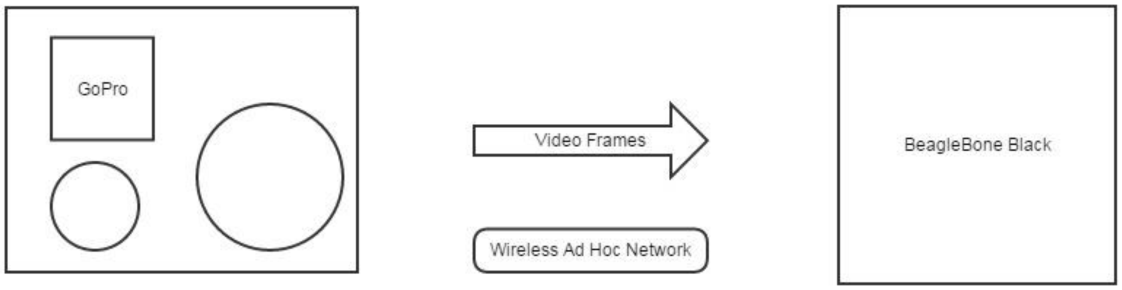


Figure 6-2: Image Processing Subsystem

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 brightly colored, orange tennis ball. As illustrated in Figure 6-2: Image Processing Subsystem, the BeagleBone 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.

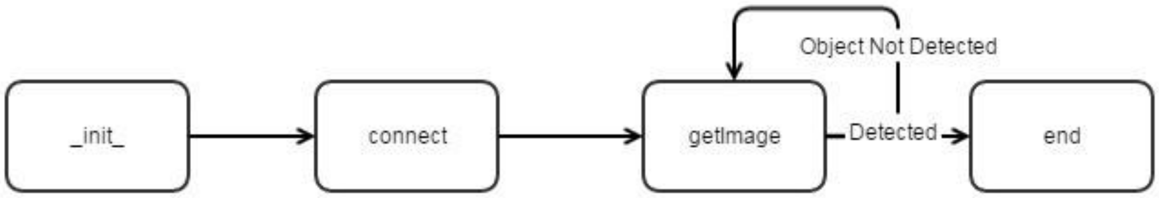


Figure 6-3: GoProController Activity Diagram

The function __init__, as seen in Figure 6-3: GoProController Activity Diagram, creates the application logs and sets up and enables WiFi for the BeagleBone, and the connect function establishes the connection between the BeagleBone 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 that should be 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 behavior depicted in the activity diagram in Figure 6-3: GoProController Activity Diagram. This class will be called GoProFrameAnalysis. It will import GoProController, and once a frame has been saved to an image file, GoProFrameAnalysis will analyze the image using OpenCV to detect the tennis ball. Once the analysis is complete, GoProFrameAnalysis will either extract another frame from the feed if the object is not detected, or it will calculate a heading that the quadcopter needs to take in order to position itself directly over the object if the object is detected.

The detection of the object will be handled by functions and classes defined in the OpenCV library. Each frame extracted from the video feed is tested against a cascade of classifiers. The OpenCV Python application for face detection imports from the numpy and cv2 libraries. Using the xml created by the OpenCV CascadeClassifier application. After reading in a jpg file and converting it to a grey image, the application uses the detectMultiScale function to detect faces within the grey image. These faces are returned as rectangular coordinates.

The output of the Python code is displayed in Figure 6-4: Face Detection Output, reprinted pending permission from OpenCV. This code is not only intended to detect eyes and faces, though. To alter this code to detect a tennis ball, modifications only need to be made to the xml files passed as arguments for the CascadeClassifier method at the top of the image. These xml files contain the classifiers which dictate what passes as a face or an eye and what does not. Classifier training will be covered later on in this section; however, for a more detailed discussion on the training of classifiers, reference Section 4.1.3: Image Processing.

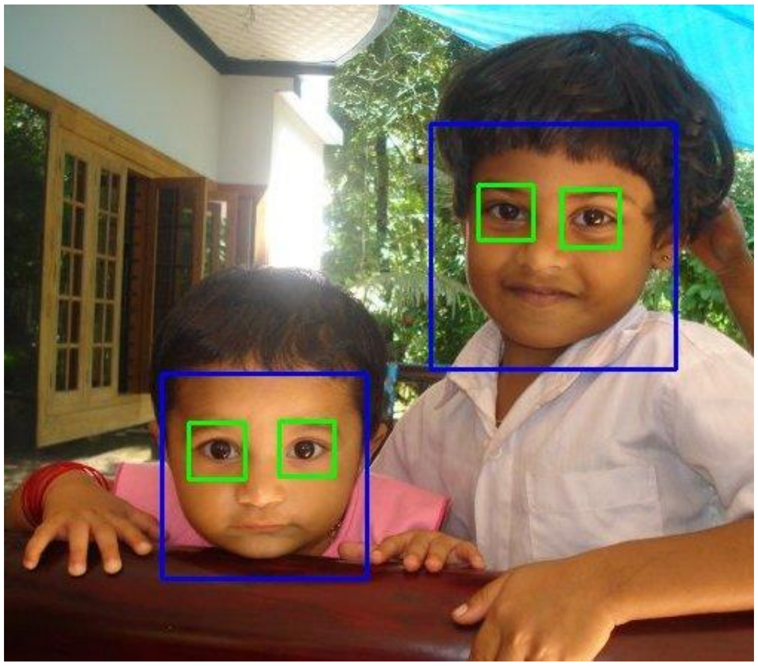


Figure 6-4: Face Detection Output

OpenCV includes two applications for training cascade classifiers, opencv_haartraining and opencv_traincascade. Opencv_traincascade is the more recent of the two and has rendered opencv_haartraining obsolete; therefore, opencv_traincascade is the trainer which shall be used by SARS. The trainer works by analyzing two sets of images, a positive set containing the object to be detected and a negative set not containing the object. The positive set is created by the application opencv_createsamples. This program takes in a source object image and a background description file containing a list of background images to be used randomly with alterations of the source image. After running, it outputs a file containing an arbitrary number of positive samples. The number of samples is determined by the user; however, thousands of samples are sometimes necessary for creating high performance classifiers. The negative samples must be created manually and stored in a folder residing in the same directory as a text file containing the file name and relative path of each image. With the positive and negative image sets created, opencv_traincascade can create the classifiers and store them in an xml file located in a folder specified in the command arguments.

Image set creation and cascade training can be performed on any machine prior to the launching of SARS. The file cascade.xml needs to be transferred to the BeagleBone prior to launch, and the object detection code can run on the BeagleBone as many times as is necessary to detect the tennis ball.

Once the object has been detected, a flag will be sent via serial connection from the BeagleBone to the quadcopter microcontroller to interrupt the current course of the quadcopter. A new heading will be calculated by taking the indices of the central pixel within the rectangle containing the detected object relative to the indices of the image center. Based on the slope and distance between the two pixels, a new heading shall be sent via serial connection from the BeagleBone to the quadcopter microcontroller. The image processing will continue to run until the quadcopter is located directly over top of the target object.

6.1.3 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 is scanning an area for an object and collecting images of the entire field, it is necessary that we plan the waypoints such that the copter scans 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 is perfect for our project.

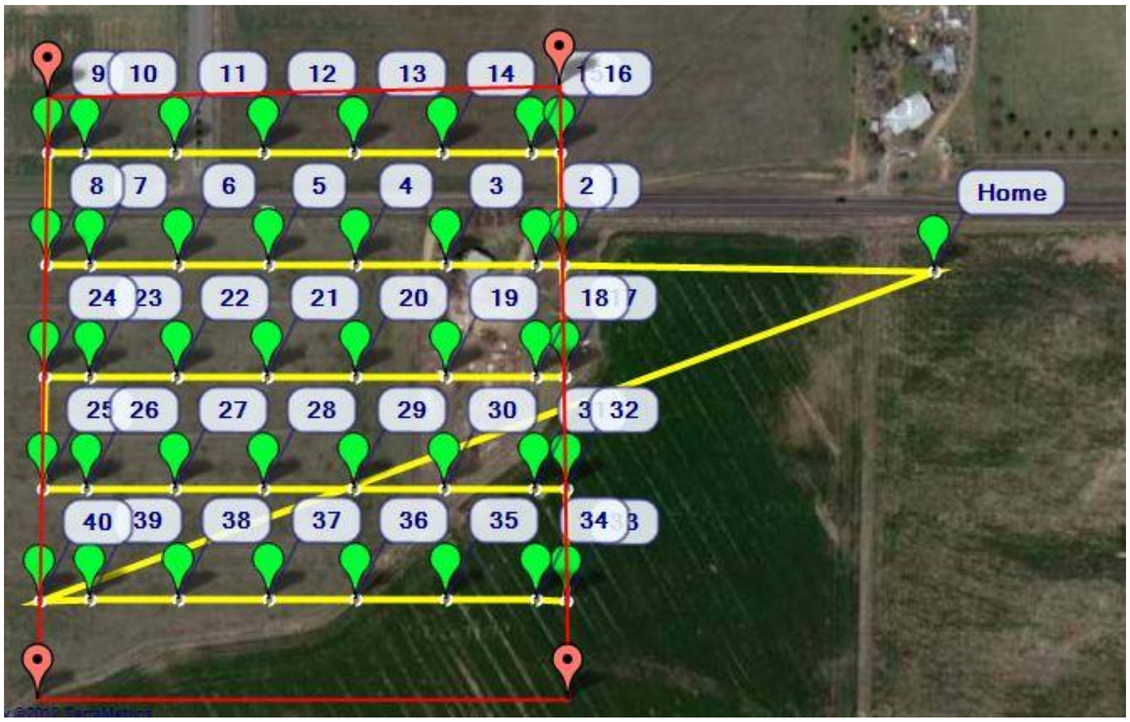


Figure 6-5: Auto Grid

The significant part of the design approaches once we have the detected the image. The copter is 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 is 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.

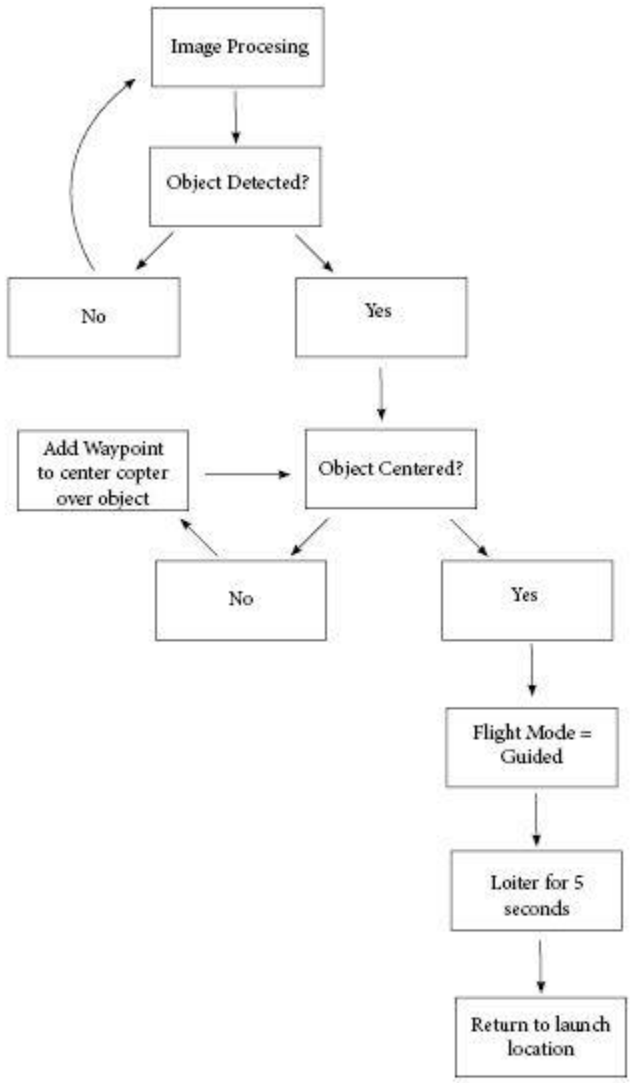


Figure 6-6: Image processing and object detection flow chart

6.1.4 Geolocation Subsystem

Once the quadcopter has identified the target object on the ground and is hovering directly over top of it, it needs to send its GPS coordinates to the SARS rover and to an Android App running the SARS App. This will be accomplished using the Ublox LEA-6H GPS module designed by

3D Robotics Inc. This GPS module has an APM compatible 6-pin DF13 connector which can be used to interface with the Pixhawk controlling the quadcopter. The entire process implemented by the geolocation subsystem is displayed in Figure 6-7: Geolocation Subsystem Activity Diagram.

Once the quadcopter has halted over the target object, its GPS coordinates will be sent from the Pixhawk to the BeagleBone Black via serial connection. This will be handled by Python scripts on both boards which will run after the Pixhawk receives the halt flag from the BeagleBone Black. At this point, the ASCII message received by the BeagleBone Black will be sent to the microcontroller on the SARS rover via Wi-Fi using the XBee module. Once again, this will be handled by a Python script on the BeagleBone Black. Additionally, since SARS is designed to include an Android application for the user to monitor diagnostic information and to view the video feed from the quadcopter, the ASCII message will need to be converted into a readable format if it is to be displayed for the user to see. There is a useful tutorial on the Internet on GPS serial communications which includes a code written in C that can be used to receive and convert the coordinates. This code will be a useful resource as the team attempts to implement the communications and the conversion in Python. After the coordinates have been sent to the SARS rover, they will be sent from the BeagleBone Black to the Android device running the SARS App once again via Wi-Fi using the XBee module.

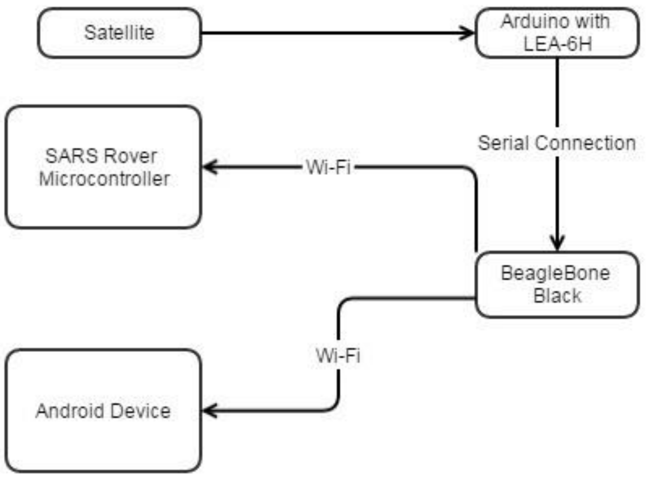


Figure 6-7: Geolocation Subsystem Activity Diagram

6.1.5 Object Detection Interruption

Initially, the course of the quadcopter will be set using Mission Planner. The quadcopter will travel between waypoints along some arbitrary path. In order to avoid missing areas on the ground and to maximize the chance of picking up the target object with the camera, this arbitrary path will more than likely be in the shape of a grid. When the target object is detected in the video feed by the BeagleBone, a new heading will be sent via serial connection from the BeagleBone to the Pixhawk. This flag will halt all Mission Planner processes and direct the quadcopter to follow the new heading. All the variables governing how the quadcopter will be directed to follow the new heading will need to be determined through extensive calibration and testing.

The task of interrupting the Mission Planner application will be accomplished by adding a Python script to the Pixhawk. This script will receive the serial output of the BeagleBone and pause the execution of the current Mission Planner command. For an illustration of how the interrupt is meant to work, see Figure 6-8 below.

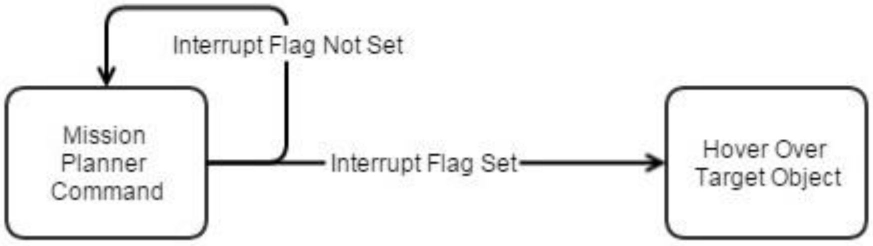


Figure 6-8: Object Detection Interrupt Activity Diagram

The ArduCopter software is open source, which means that a check flag subroutine can be added to the code. The flag will be set when the new heading is received from the BeagleBone. Once this flag is set, the check flag subroutine will interrupt the current Mission Planner Command, and redirect the quadcopter along the new heading. The quadcopter will continue receiving heading information from the BeagleBone until it is positioned over top of the target object with no more than six inches of error. At this point the quadcopter's GPS coordinates will be sent to the BeagleBone via serial communication, and the BeagleBone will transmit them to the land rover using the XBee's Wi-Fi capability. Once all of this has been completed, the quadcopter will be directed to land back in its original takeoff location.

It is important that the check flag subroutine and the Mission Planner interruption do not take so many clock cycles as to disrupt the actual flight mechanics of the quadcopter, which must maintain stability as it changes course. This will require careful optimization of the altered code.

6.2 Rover Software Design

6.2.1 Speed and Direction Control

The first and foremost aspect of rover movement is speed and direction control. It will be 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 will be 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 will involve 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 will be 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.

Once all of the functions to issue commands to the Sabertooth have been constructed, the code for deciding what commands to issue can be defined. Most straight line drive commands will be 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 is on course and not about to collide with something, on each iteration of the control loop the microcontroller will first look to the Hall Effect sensors mentioned in section 4.3.3. The microcontroller will verify that the wheels are moving as expected by comparing the data from the Hall Effect sensors with the data provided by the accelerometer. If the Hall Effect sensors claim that a wheel is rotating, but the accelerometer determines that the rover as a whole is not moving as expected, the software must make adjustments to speed and direction commands issued because the rover is likely to be stuck on some aspect of the terrain. Corrective actions will include turning as well as reversing to find a more easily traversed piece of terrain. These sensor checks are vital in keeping the rover moving forward towards its target and will have the most important say in which commands are issued to the Sabertooth.

6.2.2 Object Detection

While the rover is in motion using the commands defined in the previous section, it is certainly possible that obstacles will appear in the path of the rover that must 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 has been determined, the actual procedure for object avoidance can be detailed.

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

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

6.2.3 GPS Navigation

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

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

Table 6-1 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 6-1: 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 calculations can be 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 can be continually performed using the magnetometer in the sensor hub to verify that the rover is rotating to the proper angle so that it can drive straight towards the target object.

During the majority of the travel time once the proper heading has been determined for reaching the target object, the rover will be traveling at its maximum capable speed. This must change, however, the closer the rover is to its target. Once the rover is within a few meters of the target object, as determined by the distance formula, the rover must slow 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 will no longer be a factor in determining the path moving forwards. GPS is only expected to have an accuracy up to roughly 2 meters, making it useless once the rover has covered the majority of the distance to the target.

6.2.4 Object Retrieval

The software for object retrieval is near the top of the list of technically challenging software development. The more degrees of freedom that are applied to the object retrieval apparatus, the more intricate the control software must be in order to account for it. The object retrieval software will include three main components once the rover is within range for the apparatus to retrieve the target object. The first component is positioning the apparatus so that it can apply the grabber. The next component is applying the grabber and verifying that the object has been secured, and the final aspect is returning the object to the object storage bin.

The first stage of the retrieval software is the most difficult stage. The software must first determine the current status of the retrieval arm. Once the arm's status has been determined, the arm can be commanded to move towards the target object. As the arm is moving, a sensor on the arm must be used to determine whether or not the arm is closing in on the target object. This will be an intricate interaction between the physical sensor and the software. Because the sensor being used will not involve computer vision, distinguishing between the target object and the ground will be a challenge in itself. The arm will likely have to move to where the sensor indicates it is over the object, and from there the arm will have to be moved from side to side and back and forth to determine where the center of the object is for the strongest pickup success potential.

Once the software has determined the arm to be centered over the object to be retrieved, the actual retrieval process must begin. The software must command the apparatus to make contact with the target object and engage its grabbing feature. The biggest question to arise at this point is how long must the grabbing command last to ensure that the object is successfully grabbed? It is unlikely that this can be theoretically determined and will in turn require testing to effectively discern. The software must be designed with this in mind. The software must be capable of having constants easily modified to maximize success rates when grabbing target objects.

The final step to the software for object retrieval is returning the object to the storage bin once it has been grabbed by the grabbing apparatus. This will also be a somewhat complicated step because it requires the software to be aware of the arm's current location with respect to the object storage bin. The sensor attached to the arm can be used in conjunction with the list of commands issued to navigate to the object with the arm to return the arm to release the item into the storage bin. Assuming the arm starts centered over the storage bin, if the list of commands issued to reach the object are stored, the commands can be reversed to approximate movement back to being centered over the storage bin. The sensor can also be used to determine where the bin is. Once the arm is determined to be centered over the bin, the release command can be issued from the software to drop off the item

6.3 Android Application Design

6.3.1 Application Requirements

The Android application will serve as the interface between the human users and the robotic subsystems that form SARS. The Android application performs 3 major tasks. First, it initiates, aborts, and reports progress on a search and retrieve mission as it progresses. Second, it provides access to the live video stream of the quadcopter's camera, so that a mission can be observed in real time. Third, it provides diagnostics and telemetry data from the quadcopter and the rover so that mission conductors can monitor each subsystem during a mission and determine how to proceed at each step of the mission. To accomplish these tasks, the Android application needs to have a simple and intuitive text-based interface with touch controls that allows users to easily monitor SARS as it performs a mission. As each subsystem of SARS is physically independent of one another, the Android application needs to communicate with each subsystem wirelessly using 802.11 Wi-Fi. The success of a SARS mission is critically dependent on fast and accurate communication between each system, so the Android application must be designed to connect and communicate with each system as quickly and efficiently as possible. As the SARS application will potentially be viewed by demonstration participants, there must be a system in place to lock the user interface when viewing the video stream to prevent anyone other than the mission conductors from interacting with the SARS subsystems. The key application requirements are summarized below.

Application Requirements

- Initiate and abort SARS missions
- Monitor mission progress
- View camera live stream quickly and easily
- Communicate with other SARS subsystems wirelessly
- Limit administrative access to essential mission personnel only

6.3.2 Users and Modes of Operation

There are two types of users of the SARS application. Users who are actively conducting a mission with SARS, and users who are passively observing a SARS mission take place.

Mission Conductors: The mission conductors need the SARS application primarily for diagnostics and mission control. They will use the application to initiate missions, abort missions, monitor mission progress, view the mission live stream, and monitor the quadcopter and rover diagnostics and telemetry.

Passive Observers: The SARS application will also be used for passive observation by demonstration participants. They will primarily use the application to view the live stream from the Go-Pro camera. They should not be able to access other parts of the SARS application interface when viewing the camera stream so as not to disrupt any aspect of a mission in progress.

There are two modes of operation for the SARS application that directly corresponds with the user type. Mission Conductors need to have full administrative access to the entire SARS system in what is termed "Administrative Mode." Passive Observers have limited access to the rest of the application when viewing a mission live stream. In this scenario, the application will be in "Guest Mode."

Administrative Mode: In this mode, users have unrestricted access to every menu of the SARS application. They can initiate missions, abort missions, view mission progress reports, view live video streams, and view quadcopter and rover diagnostics and telemetry.

Guest Mode: In this mode, users only have access to the live video stream and the quadcopter and rover diagnostics and telemetry. Access to the mission control menus of the application (where missions can be initiated, aborted, and monitored) will be prohibited.

6.3.3 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 Nexus 5 running Android 5.0 (Lollipop)
- TCP/IP based wireless communication with the quadcopter's Xbee module using 802.11 b/g/n WiFi
- TCP/IP based wireless communication with the rover's Xbee module using 802.11 b/g/n WiFi
- HTTP/HLS (TCP/IP) based wireless communication with the Go-Pro Hero 3 camera's web server using 802.11 b/g/n WiFi

- Touch buttons for initiating and aborting 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
- Toggle for switching between Administrative and Guest modes
- Password lock for Administrative mode

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:

- Application runs on any Android phone running Android 4.0 (Ice Cream Sandwich) and higher
- Application runs on any Android tablet running Android 4.0 (Ice Cream Sandwich) and higher
- SARS mission live stream is viewable by multiple concurrent Android 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

6.3.4 Implementation

The SARS Android application will be designed using Android Studio on a Windows 8.1 based desktop/laptop development environment. The application will initially be designed for API level 19, which supports Android 4.4 and 5.0. If time permits, the API level will be lowered to 14, which supports all devices running Android 4.0 and higher. A Git repository will be established to track and control every version of the application that is built. This Git repository will be managed locally by Erick Makris, the lead engineer for Android application development. In the event that other SARS members become more involved in the Android application development, the Git repository will be moved to GitHub to provide centralized control and distribution between all group members. The application will be written in Java, the programming language used for all Android applications. Version 8 of the Java Development Kit will be used for application development.

6.3.5 High Level System Design

The SARS application will be designed as a Client-Server architecture. The Android application acts as the "server" in this model as it is both the interface between SARS and its human users and the central hub for inter-device communication. The quadcopter's and rover's Xbee modules as well as the human users act as the "clients" that interact with the Android application. The Go-Pro camera's web server contains all of the live video stream files and playlists, so it acts as another "server" in this model. Figure 6-9 below illustrates the design of this architecture. The directional arrows indicate the direction of data flow between each client and server in the system. This diagram only illustrates the client-server nature of the Android application as it relates to the other subsystems of SARS. It does not necessarily reflect the entire SARS system architecture, as the other subsystems will interact with each other directly, independent of the Android application.

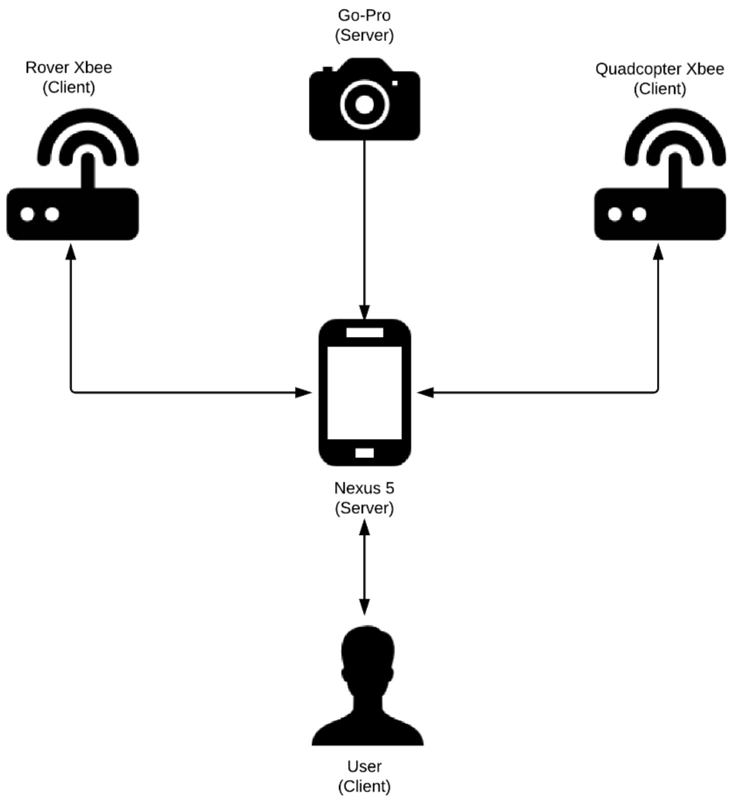


Figure 6-9: High Level System Architecture

6.3.6 Design Issues

Reusability
The SARS application is designed for a very specific purpose. It is unlikely that any of the user interface or mission control specific code can be reused. However, the socket programming code for communicating with the Xbee modules and the Go-Pro web server could probably be repurposed for future projects involving Xbees or a Go-Pro camera. It would thus be in the best interest of the team to develop the socket programming code to be as nonspecific and modular as possible to facilitate reuse of the code in future projects.

Maintainability
SARS is a Senior Design project that will be designed, developed, and tested over the course of two semesters. At the end of the second semester, it will be presented to a review board. It is unlikely that development on the project will continue after it is presented for two reasons. 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 is critical to the success of SARS as a whole. The entire system will be continually and thoroughly tested throughout the development process. Thus, it is critically important that the Android application be designed with testability in mind. The application should be able to start and stop missions as many times as necessary. Each mission reset should properly reinitialize each SARS subsystem as well as the Android application. The application should not need to be closed or stopped in order to properly reset for a new mission.

Performance
SARS must have real-time communication performance to be successful. Luckily, the data overhead is not large for wireless communications between subsystems. The 802.11b/g/n protocol will provide more than enough bandwidth and low enough latency to provide real time communications to and from the Android application. However, the application should be optimized for performance wherever possible by minimizing operational complexity and eliminating redundant or unnecessary data transmission.

Portability
As stated previously, the application will initially only be designed for a Nexus 5 running Android 5.0 and will be back-ported to older versions of Android with a wider array of device support if time permits. As such, it is recommended that the application be designed to use as few API level 19 and above specific development features as possible. If little to no API-specific features are implemented, then porting to other versions and devices may be as simple as enabling them in the list of supported devices in the Android Studio project.

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. It must be readily apparent in the application how to abort a mission and the commands must be sent as soon as possible to the quadcopter and rover to abort their current mission.

7 Integration Summary

The SARS integration plan has not been completely established yet; however, some of the wiring diagrams have been created, and certain aspects of the hardware/software integration have been considered. These diagrams are included in Section 5: Hardware Design. The BeagleBone Black and the Pixhawk both have local Python support; therefore, the SARS Group has decided to use Python to implement as many of the image processing and quadcopter functionalities as possible.

The diagram for the UART connection between the BeagleBone Black and the XBee module has yet to be created; however, the BeagleBone Black will communicate with the GoPro camera using the XBee module. The power source for the BeagleBone Black has not been decided upon as of this stage in the development. The BeagleBone may simply be powered using 6 AA batteries and a voltage regulator; however, if mounting these extra batteries on the SARS Copter becomes a problem, the BeagleBone may split the power source with the Pixhawk.

Once a frame has been extracted from the GoPro feed and stored on the BeagleBone Black, the OpenCV Python library shall be used to process the image and identify the target object in the image. Once the object has been found in a video frame, the BeagleBone shall notify the Pixhawk by sending a flag via UART. Prior to this interruption, the SARS Copter shall navigate between waypoints using the Mission Planner application which is compatible with ArduCopter. The wiring diagram for this connection between the two microcontrollers is displayed in Figure 5-2: Wiring diagram between BeagleBone and flight controller.

The Pixhawk will need to be wired to the IMU and the Ublox GPS unit as well as to the power source. This diagram has not yet been created; however, extensive information on the various ports of the Pixhawk have been included in the research and design sections of this document.

The Tiva board used as the microcontroller for the SARS Rover will need to be wired to the XBee module, the Adafruit Ultimate GPS breakout, the motor controllers, and the sensors used for object detection and for GPS navigation. 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 BeagleBone or the Pixhawk. It and all of its functionalities are to be coded in C. The XBee module will be used to receive the GPS coordinates of the target object from the BeagleBone Black mounted on the quadcopter. The SARS Rover will use its GPS module along with the magnetometer in the sensor hub to navigate to the target object’s location. Along the way it will be programmed to avoid obstacles, and 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 and its implementation have not yet been decided upon. This aspect of project development shall be revisited in early January when more progress has been made with the build of the other SARS subsystems. Various potential prototypes are discussed in Section 5.3.4: Object Retriever.

The remainder of this section will be devoted to listing the parts necessary for the implementation of SARS. This is not the team’s final list, as it is possible that certain necessary parts have been overlooked. 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

- BeagleBone Black

- XBee S6B
- XBee Cape
- GoPro Hero3 White Edition with mount
- Lithium powered battery
- 4 electronic speed controllers

SARS Rover

- Chassis
- Frame
- 6 motors
- 6 wheels
- 6 motor Controllers
- Tiva C Series
- XBee S6B
- Adafruit Ultimate GPS Breakout
- 6 Hall Effect sensors
- 6 magnetometers
- Parallax Ping))) ultrasonic distance sensor

The only hardware necessary for the implementation of the Android application is a device running the Android operating system. For testing purposes, the team will be using a Nexus 5.

8 Prototype Construction and Software Development

8.1 Quadcopter Parts Acquisition

8.1.1 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 quadcopter already has a prefabricated GoPro mount built into it; 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 boards 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 have elected to pay for the GoPro as they are interested in keeping it once the project has been completed.

8.2 Quadcopter Assembly

8.2.1 Camera

The GoPro camera will be mounted underneath the quadcopter using the prefabricated mount. The camera angle will be adjusted to 90 degrees so that the field of view is directed straight down towards the ground with an acceptable error of 1 degree. One foreseeable complication could be that the prefabricated mount may not be designed to support a 90 degree adjustment. If this is the case, a flat panel of plastic will be mounted in the prefabricated mount at an angle of 0 degrees. The flat, adhesive mount which came with the GoPro shall be attached to this panel, and the GoPro shall be mounted here with its field of view directed towards the ground. If a flat panel of plastic cannot be manufactured to fit the prefabricated mount, a second weatherproof GoPro case may be acquired and the adhesive mount attached to the flat backside of this case. For a design diagram of this setup, see Figure 8-1: GoPro Mount Design below.

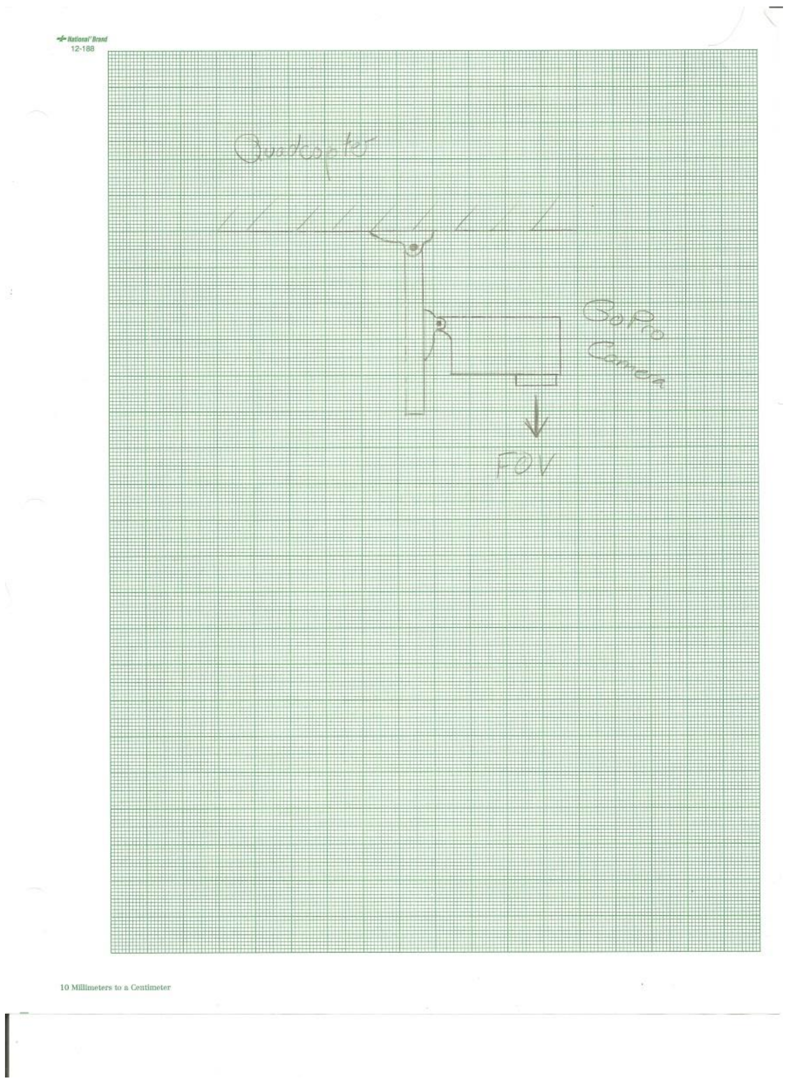


Figure 8-1: GoPro Mount Design

8.3 Rover Parts Acquisition

8.3.1 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 34:1 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 will be ordered early to ensure timely arrival. It is available for $125 regardless of the purchase source.

8.3.2 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 will be used for the majority of the design and testing process. As the completed prototype deadline approaches, this microcontroller will be transferred to a printed circuit board along with any sensors that can be integrated into a PCB.

8.3.3 Sensors

Many of the major sensors for the rover system are included in a single chip from Texas Instruments. This chip is the Sensor Hub Booster Pack, designed to be used with the Tiva C development board. This board includes the accelerometer, gyroscope, magnetometer, as well as some other sensors that will not be used. This sensor set was acquired at the same Texas Instruments workshop that the main microcontroller for the rover was acquired at. This sensor hub will be included in the PCB for the microcontroller.

The Hall Effect sensors and 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 will be purchased directly from Parallax. It comes packaged in a kit that includes mounting brackets and a 180° servo motor for $45. The same is true of the Adafruit Ultimate GPS breakout chip which is priced at $45.

8.3.4 Retrieval Apparatus

Parts acquisition for the retrieval apparatus will be slightly different from the rest of the Rover's acquisition process. Much of the hardware needed to construct the arms will be purchased at a hardware store such as Lowe's or Home Depot. High torque servos will be purchased from a website such as sparkfun.com or servocity.com.

The gripper will be designed using some parts that are to be 3D printed. There is a 3D printer available for use in the Texas Instruments Innovation Lab on the UCF Campus that will be used for this purpose. Other materials needed such as the balloon and filter can be obtained from any of many general retail stores.

8.3.5 Communications Module

8.3.6 Power Source

The main power source for the rover is a single Lithium Polymer (LiPO) battery pack. When selecting one of these battery packs, there is a huge range of available products. The website hobbyking.com has the best selection of these batteries for the best prices. The battery selected in the design section is available for $33 from the Hobby King website.

In addition to needing a battery, a battery charger is also needed. It is important to note that charging LiPO batteries requires an electronic controller to ensure that the batteries are not overcharged. Such a charger is available from Hobby King's website for $20, and will be purchased from there.

8.4 Rover Assembly

8.4.1 Chassis, Motors, and Motor Controller

Assembly of the main chassis for the rover will require very little. The chassis comes 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.

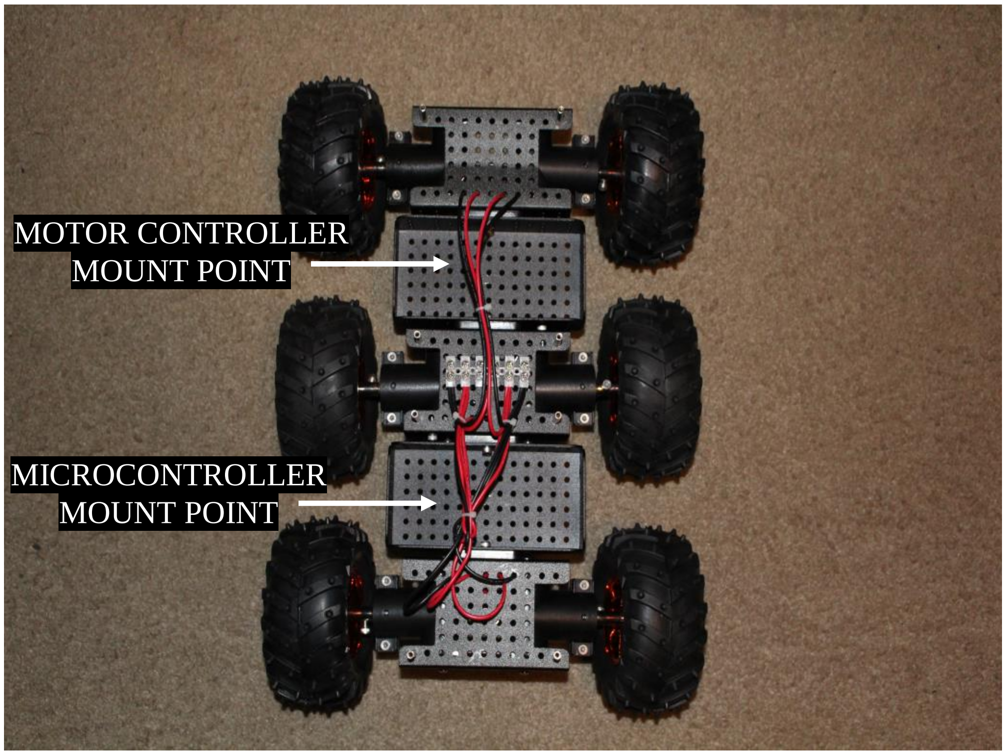


Figure 8-2: Motor Controller and Microcontroller Mount Points

Some of the edges of the board must be shaved down in order to allow the board to fit properly. The other task in mounting the board is 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 do not line up with the available holes on the motor controller for mounting. Once the appropriate holes are drilled through the chassis, a small screw will be used in conjunction with a corresponding nut to secure the motor controller in place. The mounting location is illustrated in Figure 8-2.

After mounting is complete, the initial wiring can take place. Wiring will be done as specified in the design section. The motor controller can first be connected to the differenct motors. This can be done very simply through the central wiring hub on the chassis. This hub is part of the chassis' initial configuration and is visible in Figure 8-3. 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 will be connected in parallel, allowing for equal voltage across all of them.

8.4.2 Microcontroller, Sensors, and Communication

After the main chassis components are assembled, the electronics can be mounted and installed. The first of the electronics to be installed will be the microcontroller. The mounting point for the microcontroller is visible in Figure 8-2. In the finished prototype, the microcontroller will be implemented with a printed circuit board. Creating the PCB will eliminate the mounting issues presented by the motor controller. With a PCB, the design can intentionally leave space in the

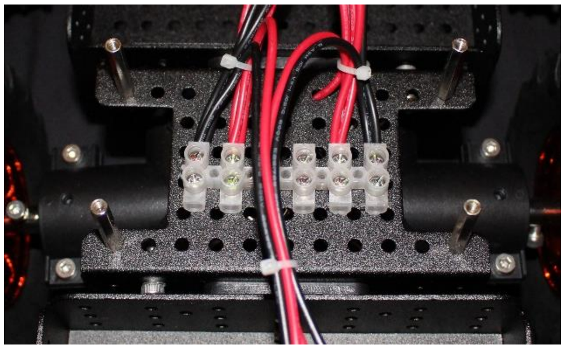


Figure 8-3: Central wiring hub for Rover

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

Now that the PCB is in place and powered, the sensors can be addressed. The main sensor hub, as discussed in the design section, will be included in the PCB, so it will not be need to be further addressed during assembly. The sensor that requires the most assembly is the PING))) ultrasonic sensor. The first task is to mount the sensor onto a small servo. This will be achieved using an adhesive such as a two part epoxy. Once the sensor is mounted, the servo can be mounted onto the chassis. The servo will be mounted very close to the front of the chassis to prevent the sensor from getting interference from within the Rover system. Both the servo and the sensor must be wired to the PCB for data and power.

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

Mounting the GPS module will require slightly more effort because it must be slightly elevated from the rover chassis to allow for better signal reception. It will first be mounted onto a pair of rods to extend it off of the chassis. These rods will also function as a route for the wiring to follow down to the PCB. After mounting is completed, the wiring will be run to a UART connection on the PCB.

Finally, the communications module can be mounted. It will be attached directly to the chassis and will likely require new holes to be drilled so that it can be secured. Its mounting will be oriented so that its antenna will have optimal reception. Once mounting is complete, it will be wired to another of the UART connections of the PCB.

8.4.3 Retrieval Apparatus

Much like the elevated level of design required for this aspect of the Rover, assembly is also a daunting task for the retrieval apparatus. The first stage of assembly is constructing the object grabber. As described in the design section, a hollowed out semi-circular piece of plastic for the balloon will be created with a 3D printer. The balloon will be appropriately filled with coffee grounds to allow for the gripping property. The balloon then must be fed through the opening in the semicircle and the filter must be installed. Next the vacuum tube must be attached securely so that no air can escape.

Once the main gripper apparatus is assembled, next is the actual arm. The arm will have to be assembled in pieces. Each section of the arm will have two aluminum beams separated by enough space for the servos to allow for the degrees of freedom. The arm sections will be connected at both ends to another section with the servo in position to allow each joint to rotate. At the end of the arm where the gripper is attached, the joint will be connected onto the short arm with the gripping apparatus on the end. The end of the arm that attaches to the rover must be attached to a small plate and that plate must be mounted to the servo for rotation.

After completing assembly of the mechanical aspects of the retrieval apparatus, the vacuum tubing and wiring can be completed. The vacuum pump must be mounted directly to the chassis and the tube currently connected to the gripper must be run along entire length of the arm and secured in place using loose fitting zip ties. The tube must then be secured onto the vacuum pump. Next, we move on to the wiring. Control wires for each servo must be run to the appropriate PCB connections. Similar to the tubing, the wires should be loosely attached with zip ties. Power must also be connected to the servos. Lastly, the vacuum pump must be connected to the power and wired to one of the GPIO ports of the PCB for control.

8.4.4 Power Source

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

Battery mounting will begin by first installing the necessary threaded spacers for the frame around the battery. Once installed the spacers should be far enough off of the body to allow for the height of the battery with very little extra space. Three spacers will be installed along each long side of the battery and two spacers across each short side of the battery. Next, the top securing panel must be drilled to line up with the layout of the spacers. A small amount of cushioning will be adhered to the top panel to absorb some of the shock on the battery from the movement of the Rover. Once the top panel is prepared, place the battery down into the frame and verify there is enough space for the lead wires to be connected and secure the top panel.

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

8.5 Android Application Development

In Android application development, any screen or menu of the application interface that is accessible to users is known as an activity. The SARS Android application will consist of 5 main activity screens. Each screen is illustrated and explained below.

Main Menu
When the application first launches, users will be presented with the Main Menu activity. From here, users can select to view the Mission Control, Video Stream, Quadcopter Diagnostics, and Rover Diagnostics activities. A screenshot of the current prototype UI for the Main Menu activity appears below in Figure 8-4 for reference.

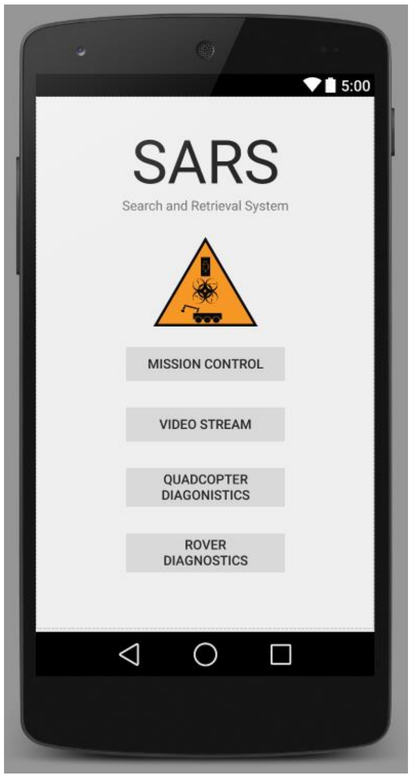
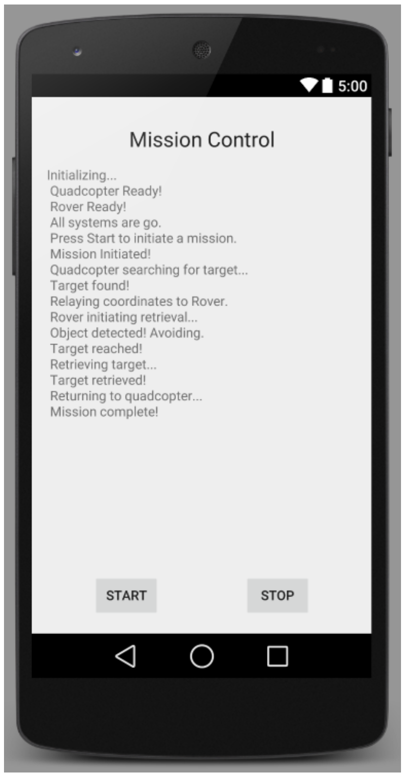


Figure 8-4: Main Menu Screen

If the user presses the Mission Control button, they are taken to the Mission Control activity. From this screen, users can initiate and abort missions. System status as well as mission progress is also displayed to users via text in the center of the screen. A screenshot of the current prototype UI for the Mission Control activity appears below in Figure 8-5 for reference.



Initializing...
Quadcopter Ready!
Rover Ready!
All systems are go.
Press Start to initiate a mission.
Mission Initiated!
Quadcopter searching for target...
Target found!
Relaying coordinates to Rover.
Rover initiating retrieval...
Object detected! Avoiding.
Target reached!
Retrieving target...
Target retrieved!
Returning to quadcopter...
Mission complete!

Figure 8-5: Mission Control Screen

If the user presses the Video Stream button, they are taken to the Video Stream activity. This activity connects to the Go-Pro's web server, accesses the appropriate M3U playlist, and opens this playlist in the native video player in full screen. There are no user controls on this screen, it is simply a full screen live video feed. However, if the user taps the screen, video controls as well as the Android navigation soft buttons will appear on screen. If the user wishes to exit the video player, they simple press the soft back button, and it will hierarchically navigate to the previous screen (Main Menu). Figure 8-6 illustrates what the video feed looks like on the Nexus 5 when in full screen. Figure 8-7 illustrates what happens when the user taps the screen while in a full screen video.

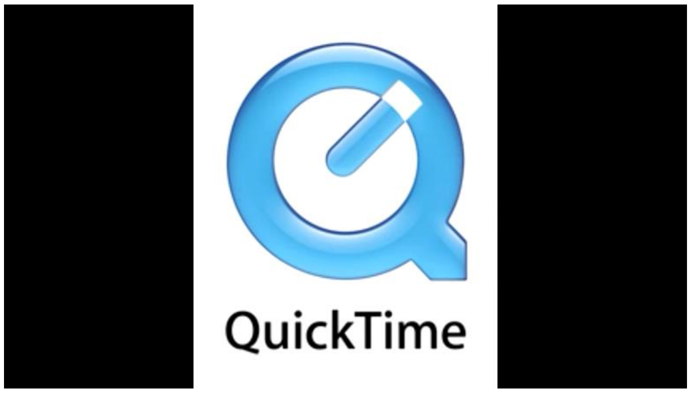


Figure 8-6: Full Screen Video (No Controls)

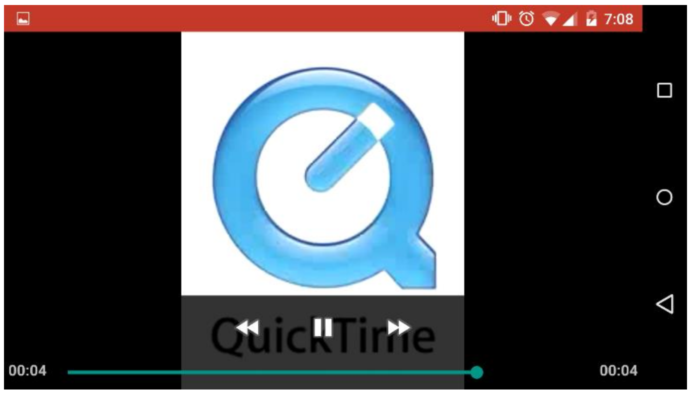


Figure 8-7: Full Screen Video (With On-Screen Controls)

If the user presses the Quadcopter Diagnostics or Rover Diagnostics buttons, they are taken to the respective Quadcopter Diagnostics or Rover Diagnostics activity. On this screen, the quadcopter's or rover's diagnostics and telemetry data are displayed to the user in text form in the center of the screen. A screenshot of the current prototype UI's for the Quadcopter Diagnostics and Rover Diagnostics screens appears below in Figure 8-8. As the exact information that will be displayed has not yet been decided, the UI prototype currently uses placeholders.

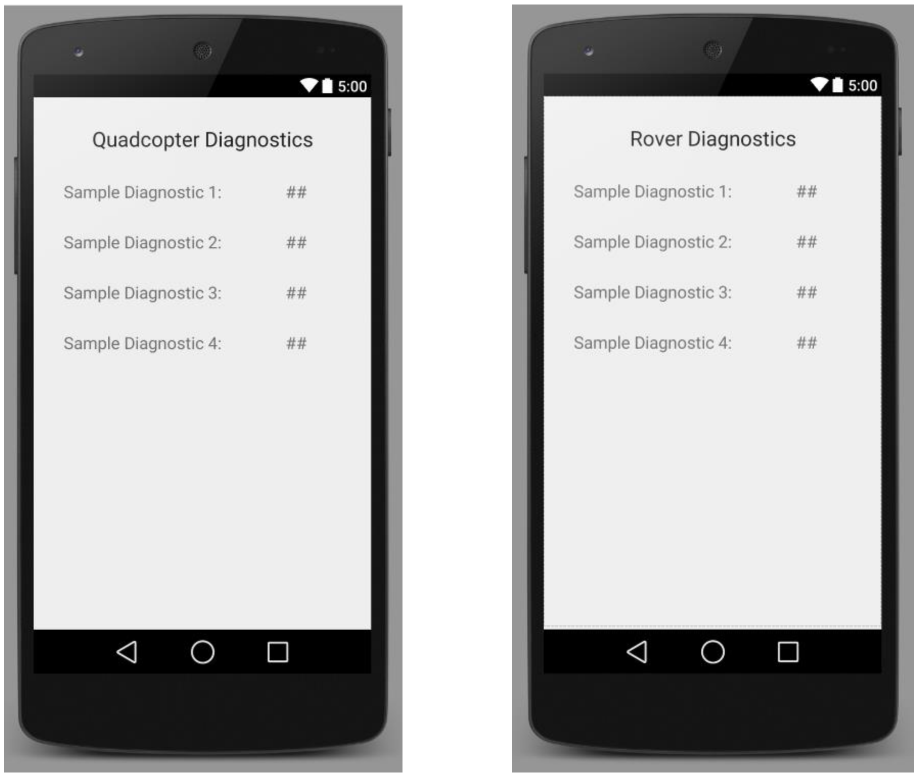


Figure 8-8: Quadcopter and Rover Diagnostics Screens

9 Prototype Testing

9.1 Hardware Test Environments

9.1.1 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.

9.1.2 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.

9.2 Hardware Test Cases

9.2.1 Quadcopter

9.2.1.1 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

9.2.1.2 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

9.2.1.3 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

9.2.1.4 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.

9.2.2 Rover

9.2.2.1 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
  - Motion sensor servo able to rotate as specified
  - All degrees of freedom for the arm move as specified
  - Object grabber operates as specified
  - Control boards power on and able to execute orders

9.2.2.2 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

9.2.2.3 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

9.2.2.4 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)

9.2.2.5 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

9.2.2.6 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

9.3 Software Test Environments

9.3.1 Quadcopter

9.3.1.1 Image Processing Test Environment

The Image Processing Subsystem will be tested on a BeagleBone Black; although, any Linux machine capable of connecting to an ad hoc Wi-Fi network would technically be sufficient. The complete list of materials for this test is as follows: BeagleBone Black, XBee module, GoPro Hero3 White Edition, power source. The application retrieving the frames from the GoPro camera's live feed and processing these images will be written in Python. The BeagleBone Black has native Python support. The application 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 will be performed indoors. Positive and negative sample images will need to be supplied for a tennis ball in an indoor setting. Section 6.1.2: Image Processing Subsystem gives detailed information on how to generate the samples. These first tests do not need to be extensive. For example, the tennis ball simply needs to be detected while sitting alone on the floor. Once the script has been verified to detect the object indoors, the tests shall be moved outside. The object detection will be tested with a tennis ball in the grass as well as on asphalt. This test will also require many positive and negative samples. 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 serial communication and waypoint interruption.

Testing for this subsystem will begin in mid to late November 2013 and will continue until integration in February and March of 2014.

9.3.1.2 Geolocation Subsystem Test Environment

The testing of the Geolocation Subsystem will be done across multiple microcontrollers. The list of parts involved in this subsystem is as follows: Pixhawk, BeagleBone Black, U-blox LEA-6H GPS module, Android device, Tiva C Series Launchpad. Tests for the Geolocation Subsystem will be conducted in an outdoor setting, where the GPS module will most easily receive satellite signals. Initial testing will be done on the ground, and to reduce the risk of damaging hardware, aerial testing will only be conducted once integration is ready to take place after the subsystem functions have been verified and validated. As these tests mainly involve communications between microcontrollers, they will be conducted later on in the development and production of SARS. The target start date for the testing of the Geolocation Subsystem is in mid-January; however, depending on the success of the Image Processing Subsystem tests, the geolocation testing may commence in late December or early January. Certain aspects of the testing may be conducted on isolated microcontrollers; for instance, the conversion of data to a readable format can be done on the BeagleBone Black with no communications with other devices. Tests of these isolated functions shall be conducted first.

9.3.1.3 Waypoint Interruption Test Environment

The testing of the waypoint interruption will be done across multiple microcontrollers. The list of parts of this subsystem is as follows: BeagleBone Black, Pixhawk, quadcopter hardware, GoPro Hero3 White Edition. The testing of this subsystem shall be conducted in an outdoor setting, as it involves the flight of the quadcopter. It may be possible to simulate the testing without flying the quadcopter, in which case, initial tests could be conducted anywhere. Group 4 will have to judge the feasibility of performing these simulations as well as their effectiveness in testing the subsystem before deciding to pursue this course of action. One definite benefit of running simulations would be to reduce the number of aerial tests, thus, reducing the risk of quadcopter malfunctions and damage to hardware components. The first tests may be conducted in the absence of the Image Processing Subsystem. The BeagleBone Black would send the flag to interrupt the waypoint after a predetermined amount of time. Eventually this process, however, will be integrated with the Image Processing Subsystem along with the rest of the SARS subsystems.

As the waypoint interruption involves the integration of several subsystems as well as communications between the Pixhawk and the BeagleBone Black, its testing will be conducted later on in the development and production of SARS. The estimated commencement date of these tests is early February, and the estimated conclusion date, at least for the initial testing is mid to late February, this leaves plenty of time for integration testing.

9.3.2 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.

9.4 Software Test Cases

9.4.1 Quadcopter

9.4.1.1 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^o
- Distance accurate to within ±1 meter

9.4.1.2 Microcontroller Software Test

Purpose: Test that the OS is correctly installed and flashed to the eMMC and that the software is running correctly on the BeagleBone Black.

Supplies:
- BeagleBone Black Microcontroller
- SD card with Ubuntu eMMC flasher written to it
- Laptop
- USB Cable

Procedure:
1. Insert SD card into powered-off BeagleBone Black
2. Hold down “boot” button and power the board by plugging the BeagleBone MCU into the laptop.
3. Let the board sit for approximately 10 minutes while the OS is being flashed
4. Verify all of the LED’s are in a solid state and not flashing

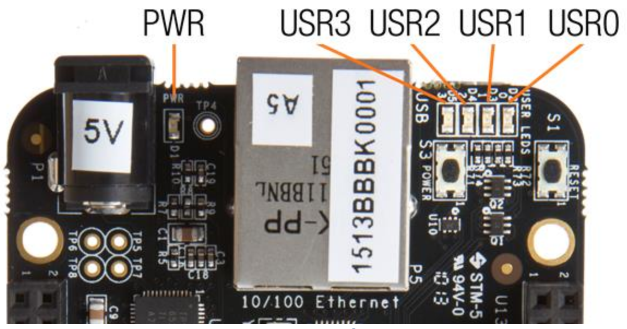


Figure 9-1: BeagleBone Black reference LED locations

5. Login to Ubuntu

    a. User: ubuntu
    b. Password: temppwd
6. Run test script to blink external LED

Expected Result: LED blinks correctly

Conditional Requirements:
Test blinking at various speeds to ensure board is able to display status of various flight modes during flight tests

9.4.1.3 Microcontroller Interrupt Testing

Purpose: Verify that the BeagleBone Black is able to send an interrupt to the Pixhawk flight controller that signals a Return_To_Launch command after the object has been detected.

Supplies:
- Pixhawk flight controller
- BeagleBone Black MCU
- Laptop (with Mission Planner)
- 2 micro USB cables
- I^2C splitter

Procedure:
1. Connect Pixhawk flight controller to laptop via the micro USB cable and power on quadcopter
2. Connect BeagleBone Black to laptop via micro USB cable and power on BeagleBone Black and confirm Ubuntu boots correctly
3. Verify serial communication interface connection between BeagleBone Black and Pixhawk flight controller
4. Set up auto-grid flight plan on Mission Planner and execute mission simulation
5. During the mission, run Return_to_Launch script on BeagleBone that sends mode change to Pixhawk

Expected Result: Mission Planner updates and shows mode as Return_To_Launch

9.4.1.4 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.

9.4.1.5 Image Processing Test Environment

The Image Processing Subsystem will be tested on a BeagleBone Black; although, any Linux machine capable of connecting to an ad hoc Wi-Fi network would technically be sufficient. The complete list of materials for this test is as follows: BeagleBone Black, XBee module, GoPro Hero3 White Edition, power source. The application retrieving the frames from the GoPro camera’s live feed and processing these images will be written in Python. The BeagleBone Black has native Python support. The application 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 will be performed indoors. Positive and negative sample images will need to be supplied for a tennis ball in an indoor setting. Section 5.1.2 on the Image Processing Subsystem design gives detailed information on how to generate the samples. These first tests do not need to be extensive. For example, the tennis ball simply needs to be detected while sitting alone on the floor. Once the script has been verified to detect the object indoors, the tests shall be moved outside. The object detection will be tested with a tennis ball in the grass as well as on asphalt. This test will also require many positive and negative samples. 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 serial communication and waypoint interruption.

Testing for this subsystem will begin in mid to late November 2013 and will continue until integration in February and March of 2014.

9.4.1.6 Geolocation Subsystem Test Environment

The testing of the Geolocation Subsystem will be done across multiple microcontrollers. The list of parts involved in this subsystem is as follows: Pixhawk, BeagleBone Black, U-blox LEA-6H GPS module, Android device, Tiva C Series Launchpad. Tests for the Geolocation Subsystem will be conducted in an outdoor setting, where the GPS module will most easily receive satellite signals. Initial testing will be done on the ground, and to reduce the risk of damaging hardware, aerial testing will only be conducted once integration is ready to take place after the subsystem functions have been verified and validated. As these tests mainly involve communications between microcontrollers, they will be conducted later on in the development and production of

SARS. The target start date for the testing of the Geolocation Subsystem is in mid-January; however, depending on the success of the Image Processing Subsystem tests, the geolocation testing may commence in late December or early January. Certain aspects of the testing may be conducted on isolated microcontrollers; for instance, the conversion of data to a readable format can be done on the BeagleBone Black with no communications with other devices. Tests of these isolated functions shall be conducted first.

9.4.1.7 Waypoint Interruption Test Environment

The testing of the waypoint interruption will be done across multiple microcontrollers. The list of parts of this subsystem is as follows: BeagleBone Black, Pixhawk, quadcopter hardware, GoPro Hero3 White Edition. The testing of this subsystem shall be conducted in an outdoor setting, as it involves the flight of the quadcopter. It may be possible to simulate the testing without flying the quadcopter, in which case, initial tests could be conducted anywhere. Group 4 will have to judge the feasibility of performing these simulations as well as their effectiveness in testing the subsystem before deciding to pursue this course of action. One definite benefit of running simulations would be to reduce the number of aerial tests, thus, reducing the risk of quadcopter malfunctions and damage to hardware components. The first tests may be conducted in the absence of the Image Processing Subsystem. The BeagleBone Black would send the flag to interrupt the waypoint after a predetermined amount of time. Eventually this process, however, will be integrated with the Image Processing Subsystem along with the rest of the SARS subsystems.

As the waypoint interruption involves the integration of several subsystems as well as communications between the Pixhawk and the BeagleBone Black, its testing will be conducted later on in the development and production of SARS. The estimated commencement date of these tests is early February, and the estimated conclusion date, at least for the initial testing is mid to late February, this leaves plenty of time for integration testing.

9.4.2 Rover

9.4.2.1 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

9.4.2.2 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

9.4.2.3 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

9.4.2.4 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

9.4.2.5 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.

9.4.2.6 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

9.4.2.7 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*9.2.2.5*

9.4.2.8 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

9.4.2.9 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

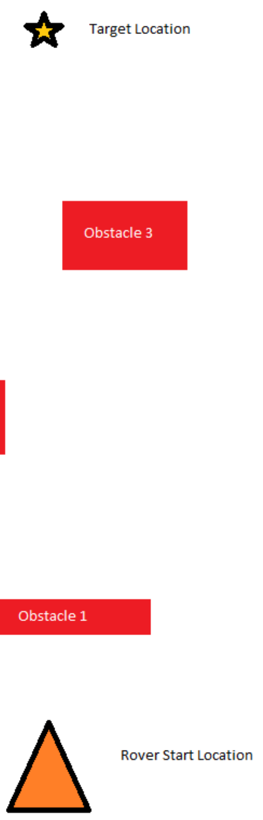


Figure 9-2: Obstacle layout for Object Detection/Avoidance Test

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

9.4.3 Image Processing Subsystem Test

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

9.4.3.1 Testing the GoProController Python Script

Purpose: To access the GoPro video feed over the camera's ad hoc network using a BeagleBone Black with a Wi-Fi attachment and to verify that a frame may be extracted from the feed.

Materials:

- BeagleBone Black
- XBee S6B (for testing purpose any Wi-Fi adapter will do)
- XBee Cape
- Power supply
- 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 BeagleBone Black 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.

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 BeagleBone Black. Furthermore, the following message should be stored in the log: "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.

9.4.3.2 Testing the Object Detection

Purpose: To verify that the BeagleBone Black can detect a brightly colored target object in the frames extracted from the GoPro video feed.

Materials:

- GoPro Hero3 White Edition
- BeagleBone Black
- XBee S6B
- XBee Cape
- SARS Copter
- Power Supply

Test Procedure 1: After creating a set of classifiers, test OpenCV's object detection application on the target object while it is indoors.

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

Test Procedure 2: After creating another set of classifiers, test OpenCV's object detection application on the target object while it is outside. Perform this test on both grass and asphalt using the same classifiers.

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

Test Procedure 3: After creating another set of classifiers, test OpenCV's object detection application on the target object while the GoPro and the BeagleBone Black are 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.

Test Procedure 4: Using the same set of classifiers as in the third test procedure, test OpenCV's object detection application in conjunction with the waypoint interrupt process.

Expected Results: This test should yield no less than a 99% detection rate and no more than a 1% false positive rate. Furthermore, once the target object has been detected, the SARS Copter should reposition itself over top of the object with an acceptable error of 6 inches.

Conditional Requirements: The first tests of this functionality should be conducted indoors. After the subsystem has passed the indoor test, it shall be tested outdoors with the target object sitting on the grass and on the asphalt. All tests shall require the collection of positive and negative samples for the creation of classifiers. Section 6.1.2: Image Processing Subsystem illustrates how these collections are built. For the initial tests no less than 500 positive samples and 500 negative samples should be collected; although, to get the desired detection and false positive rates, more samples may be necessary. For later tests, when the GoPro is mounted to the SARS Copter, no less than 2000 positive samples and 2000 negative samples should be collected.

9.4.3.3 Image Processing Subsystem Integration Test

Purpose: To verify the function of the subsystem once it has been integrated first with the waypoint interruption, then 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 waypoint interruption. The quadcopter shall travel from a starting position to a waypoint. This waypoint can be specified using the Mission Planner application that comes with ArduCopter. Somewhere between the starting position and the waypoint, the quadcopter shall pass over the tennis ball on the ground. This test shall be conducted following the initial testing of the waypoint interruption.

Expected Results: The SARS Copter should abandon the waypoint and hover over top of the tennis ball, adjusting its position so that it comes to within 6 inches of being directly over top of the target object. After maintaining this position for approximately 10 seconds, the SARS Copter should land in the immediate vicinity of the tennis ball. A success rate no less than 95% should be achieved.

Test Procedure 2: This test involves the integration of the image processing subsystem with the geolocation subsystem. The same testing procedure shall be followed. The SARS Copter shall be traveling from Point A to Point B when it passes over the target object. This test shall be conducted following the initial testing of the geolocation subsystem.

Expected Results: The SARS Copter's initial mission shall be interrupted when the BeagleBone detects the tennis ball on the ground. Once it has zeroed in and is hovering directly over top of the tennis ball, the copter shall send its GPS coordinates either to the Tiva Launchpad or, depending on the progress made with the Android application, to an Android device running the SARS App. If the SARS App is running, it shall be fairly simple determining if the GPS coordinates were relayed correctly. Otherwise, the coordinate information will need to be accessed on the Tiva Launchpad. After sending the GPS coordinates, the quadcopter should return to its starting position and land. A success rate of no less than 95% should be achieved.

Test Procedure 3: 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 and second 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.

9.4.4 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.

9.4.4.1 Testing the BeagleBone Black/Pixhawk Serial Communications

Purpose: To verify that the Pixhawk can send GPS coordinate data to the BeagleBone Black via serial connection without data corruption or loss of information.

Materials:

- Pixhawk microcontroller
- BeagleBone Black
- Ublox LEA-6H
- Power supply

Test Procedure: Have the Pixhawk send the GPS coordinate data. Check the data once it has been received by the BeagleBone Black. This test is scheduled to commence in mid-January 2015.

Expected Results: The data received must be 100% accurate. It is imperative that the correct GPS coordinates are received.

Conditional Requirements: This test may be conducted on the ground, and for the sake of receiving a strong satellite signal, the test shall be conducted outdoors.

9.4.4.2 Testing the BeagleBone Black/Tiva Launchpad Wi-Fi Communications

Purpose: To verify that the BeagleBone Black can send GPS coordinate data to the Tiva C Series via wireless communications without data corruption or loss of information.

Materials:

- BeagleBone Black
- Tiva C Series
- 2 XBee S6B’s
- XBee Cape
- Power supply

Procedure: Have the BeagleBone Black send the data. Check the data once it has been received by the Tiva C Series. This test is scheduled to commence in mid-January 2015.

Expected Results: The received data must be 100% accurate. It is imperative that the correct GPS coordinates are received.

Conditional Requirements: This test may be conducted indoors as it does not actually involve receiving GPS information via satellite.

9.4.4.3 Testing the GPS Data Conversion

Purpose: To verify that the GPS data received by the BeagleBone Black from the Pixhawk is being converted to an easily readable format. This is simply a matter of parsing the string of characters received over the serial connection.

Materials:
- BeagleBone Black
- Pixhawk microcontroller
- Power supply

Procedure: A test function shall be provided in the Python script responsible for handling this conversion.

Expected Results: The test function shall print to the terminal the new coordinates in the standard format.

Conditional Requirements: This test may be conducted indoors as it does not actually involve receiving GPS information via satellite.

9.4.4.4 Testing the BeagleBone Black/Android Wi-Fi Communications

Purpose: To verify that the BeagleBone Black can send GPS coordinate data to an Android device running the SARS application via wireless communications without data corruption or loss of information.

Materials:
- BeagleBone Black
- Nexus 5 running the SARS application
- XBee S6B
- XBee Cape
- Power supply

Procedure: Have the BeagleBone send the converted GPS coordinate data. Check the data once it is displayed by the Android device. The commencement date of this test is TBD, as it depends upon the progress made on the Android application.

Expected Results: The received data must be 100% accurate. It is imperative that the correct GPS coordinates are displayed by the application.

Conditional Requirements: The test may be performed indoors, as it does not actually involve receiving GPS information via satellite.

9.4.4.5 Testing the Rover GPS Navigation

Purpose: To verify that the SARS Rover can navigate to a specific GPS coordinate while avoiding obstacles along the way.

Materials:

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

Test Procedure 1: Given a set of GPS coordinates as a target, the rover shall be instructed to travel to those coordinates. No obstacles shall be placed in its path.

Expected Results: The SARS Rover shall travel to within 9 feet 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.

Test Procedure 2: Given a set of GPS coordinates as a target, the rover shall be instructed to travel to those coordinates while avoiding obstacles along the way.

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

9.4.4.6 Geolocation Subsystem Integration Testing

Purpose: To verify the function of the subsystem once it has been integrated first with the image processing subsystem, waypoint interruption, 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, the waypoint interruption, and the wireless communication between the BeagleBone Black, the Tiva C Series, and the Android device. This test shall have the same procedure as the second 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 zeroed in on the target object and is hovering directly over top of it, verify that the Android device is displaying the correct GPS coordinates of the tennis ball. Once the quadcopter has safely landed, verify that the Tiva Launchpad received the correct coordinate data from the BeagleBone Black. Finally, once the GPS coordinates have been sent, the quadcopter should return and land in its original starting location. A success rate no less than 95% must be achieved.

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.

9.4.5 Quadcopter Waypoint Interruption Test

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

Initial Interruption Test: The purpose of this test is to verify that the BeagleBone Black may interrupt the quadcopter’s progress towards a destination specified by Mission Planner. This test may be done in the absence of the image processing system. It would involve sending a flag from the BeagleBone Black while the quadcopter is traveling from Point A to Point B after a predetermined amount of time has passed. When this flag is received, the quadcopter will be directed to pause and hover briefly and then to land at its current location. Depending on the feasibility of simulating the quadcopter flight using text outputs, Group 4 may opt to conduct this test indoors; however, it is highly likely that an effective simulation will not be easily achieved, in which case the test would need to be conducted outdoors. In either case, a 95% success rate must be achieved for this test to pass. The initial interruption tests are scheduled to commence in early February.

Waypoint Interruption Integration Tests: The purpose of this test is to verify the function of the waypoint interruption subsystem once it has been integrated first with the image processing subsystem, then with the geolocation subsystem, and finally with all of the SARS subsystems. These tests must be conducted outside, as they require the flight of the quadcopter and accurate receipt of GPS coordinates via satellite.

All three of these integration tests and their requirements are defined in the integration testing part of Section 8.4.1.

9.4.6 Quadcopter Waypoint Interruption Test

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

9.4.6.1 Initial Interruption Test

Purpose: To verify that the BeagleBone Black may interrupt the SARS Copter's progress towards a destination specified by Mission Planner.

Materials:
- BeagleBone Black
- SARS Copter
- Power supply

Procedure: This test may be done in the absence of the image processing system. It would involve sending a flag from the BeagleBone Black while the SARS Coper travels from Point A to Point B after a predetermined amount of time has passed since takeoff. The initial interruption tests are scheduled to commence in early February.

Expected Result: When the flag is received, the SARS Copter will be directed to pause eand hover briefly and then to land at its current location. A 95% success rate must be achieved for this test to pass.

Conditional Requirements: Depending on the feasibility of simulating quadcopter flight, the SARS Group may opt to conduct this test indoors; however, it is highly likely that an effective simulation will not be easily achieved, in which case the test would need to be conducted outdoors.

9.4.6.2 Waypoint Interruption Integration Tests

The purpose of these tests is to verify the function of the waypoint interruption subsystem once it has been integrated first with the image processing subsystem, then with the geolocation subsystem, and finally with all of the SARS subsystems. These tests must be conducted outside, as they require the flight of the quadcopter and accurate receipt of GPS coordinates via satellite.

All three of these integration tests and their requirements are defined Section 9.4.1.3: Image Processing Subsystem Integration Test.

9.4.7 Quadcopter/Rover Communications Test

Purpose:
To test if the quadcopter's BeagleBone Black microcontroller and the rover's Tiva C microcontroller can communicate wirelessly through the Xbee WiFi modules.

Supplies:

- Windows desktop or laptop environment
- PuTTY Client
- BeagleBone Black Microcontroller
- Tiva C Microcontroller
- Xbee Wifi Module (2)
- Micro-USB Cable (2)

Procedure:

1. Ensure the Xbee is connected to the BeagleBone according to the design in section 5.4.1.
2. Ensure the Xbee is connected to the Tiva according to the design in section 5.4.2.
3. Ensure that the BeagleBone has its serial communications test script loaded.
4. Ensure that the Tiva has its serial communications test code flashed.
5. Connect the BeagleBone to the desktop/laptop environment with a Micro-USB cable.
6. Connect the Tiva to the desktop/laptop environment with a Micro-USB cable.
7. Open a serial communications connection to the Beaglebone using an instance of PuTTY.
8. Open a serial communications connection to the Tiva using another instance of PuTTY.
9. Send a test message to the Tiva's serial communications console from the BeagleBone's serial communications console. Determine if the message arrives on the Tiva. Measure the time delay between transmission and receipt.
10. Send a test message to the BeagleBone's serial communications console from the Tiva's serial communications console. Determine if the message arrives on the BeagleBone. Measure the time delay between transmission and receipt.
11. Repeat steps 9 and 10 3 more times.

Expected Results:

- Messages sent from the BeagleBone to the Tiva successfully register on the Tiva's serial communications console in PuTTY.
- Messages sent from the Tiva to the BeagleBone successfully register on the BeagleBone's serial communications console in PuTTY.

Conditional Requirements:

- The messages must arrive intact and in order.
- The delay between transmission and receipt of messages must not exceed 250ms.

9.4.8 Android/Quadcopter Communications Test

Purpose:
To test if the quadcopter's BeagleBone Black microcontroller can communicate with the Nexus 5 Android device.

Supplies:

- Windows desktop or laptop environment
- Nexus 5 Android device with the SARS Quadcopter test application installed
- PuTTY Client
- BeagleBone Black Microcontroller

- Xbee Wifi Module
- Micro-USB Cable

Procedure:

1. Ensure the Xbee is connected to the BeagleBone according to the design in section 5.4.1.
2. Ensure that the BeagleBone has its socket communications test script loaded.
3. Connect the BeagleBone to the desktop/laptop environment with a Micro-USB cable.
4. Open a serial communications connection to the Beaglebone using an instance of PuTTY.
5. Open the SARS Quadcopter test application on the Nexus 5.
6. Click the Test button on the SARS Quadcopter test application running on the Nexus 5.
7. A test message should appear on the BeagleBone's serial communications console in PuTTY. Determine if the message arrives on the console. Measure the time delay between transmission and receipt.
8. Send a test message to the Android application from the BeagleBone's serial communications console.
9. A test message should appear in the center of the Android application interface. Determine if the message arrives on the application. Measure the time delay between transmission and receipt.
10. Repeat steps 9 and 10 3 more times.

Expected Results:

- Messages sent from the BeagleBone to the Nexus 5 successfully register on the Quadcopter test application's interface.
- Messages sent from the Nexus 5 to the BeagleBone successfully register on the BeagleBone's serial communications console in PuTTY.

Conditional Requirements:

- The messages must arrive intact and in order.
- The delay between transmission and receipt of messages must not exceed 250ms.

9.4.9 Android/Rover Communications Test

Purpose:
To test if the rover's Tiva C microcontroller can communicate with the Nexus 5 Android device.

Supplies:

Windows desktop or laptop environment
- Nexus 5 Android device with the SARS Rover test application installed
- PuTTY Client
- Tiva C Microcontroller
- Xbee Wifi Module
- Micro-USB Cable

Procedure:

1. Ensure the Xbee is connected to the Tiva according to the design in section 5.4.2.

2. Ensure that the Tiva has its socket communications test code flashed.
3. Connect the Tiva to the desktop/laptop environment with a Micro-USB cable.
4. Open a serial communications connection to the Tiva using an instance of PuTTY.
5. Open the SARS Rover test application on the Nexus 5.
6. Click the Test button on the SARS Rover test application running on the Nexus 5.
7. A test message should appear on the Tiva's serial communications console in PuTTY. Determine if the message arrives on the console. Measure the time delay between transmission and receipt.
8. Send a test message to the Android application from the Tiva's serial communications console.
9. A test message should appear in the center of the Android application interface. Determine if the message arrives on the application. Measure the time delay between transmission and receipt.
10. Repeat steps 9 and 10 3 more times.

Expected Results:

- Messages sent from the Tiva to the Nexus 5 successfully register on the Rover test application's interface.
- Messages sent from the Nexus 5 to the Tiva successfully register on the Tiva's serial communications console in PuTTY.

Conditional Requirements:

- The messages must arrive intact and in order.
- The delay between transmission and receipt of messages must not exceed 250ms.

9.4.10 Android/Go-Pro Video Streaming Test

Purpose:
To test if the Nexus 5 can successfully connect to and live stream from the Go-Pro camera both manually and through the SARS application.

Supplies:

- Nexus 5 Android device with the SARS application installed
- Go-Pro Hero 3

Procedure:

1. Ensure that WiFi is enabled on the Go-Pro.
2. Connect to the Go-Pro's web server on the Nexus 5 by navigating to Settings > Wi-Fi. The Network ID should be "Hero3."
3. Open the Chrome browser on the Nexus 5 and enter the URL http://10.5.5.9:8080/ in the address bar.
4. Click the folder named live, then the file named amba.m3u8.
5. The live stream should open in the native Android video player. Determine if the live stream is working as intended.
6. Close the Chrome browser and reconnect to the original Wi-Fi network.
7. Open the SARS application on the Nexus 5.

8. Click the Video Stream button. The native Android video player should launch the live stream exactly as before with the browser. Determine if the live stream is working as intended.

Expected Results:

- The live stream successfully plays when accessed from the Chrome browser on the Nexus 5.
- The live stream successfully plays when accessed from the SARS application on the Nexus 5.

Conditional Requirements:

- The live video must stream with acceptable quality and latency.

9.4.11 Android User Interface Test

Purpose:
To test if the Nexus 5 user interface is easy to use, intuitive, and functions according to the specifications laid out in section 6.3.

Supplies:

- Nexus 5 Android device with the SARS application installed

Procedure:
Open the SARS application on the Nexus 5. Navigate every screen of the user interface and click every on screen button. Determine if the application is working as intended.

Expected Results:

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

10 Administrative Content

10.1 Milestones

Design documentation will be 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 10-10-1: Quadcopter Milestones below displays the intended completion dates for all major development phases of the SARS Copter. Development of this project component is currently on schedule. The design stage was very brief, as the team decided to go with a prefabricated quadcoper, and the build stage is expected to be completed on time.

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 10-10-1: Quadcopter Milestones

Table 10-10-2: Image Processing Milestones below displays the intended completion dates for all major development phases of the SARS image processing subsystem. Development of this project component is currently on schedule. The design stage was very brief, as OpenCV provides most of the libraries necessary for object detection and as the SARS Group has found an open source Python script which will facilitate the interfacing of the BeagleBone Black with the GoPro Hero3. The build stage is expected to be completed on time.

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 10-10-2: Image Processing Milestones

Table 10-10-3: Quadcopter Communications Milestones below displays the intended completion dates for all major development phases of the SARS Copter communications interface. Development of this project component is currently 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 10-10-3: Quadcopter Communications Milestones

Table 10-10-4: Ground Rover Milestones below displays the intended completion dates for all major development phases of the SARS Rover hardware. Development of this project component is currently 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 10-10-4: Ground Rover Milestones

Table 10-10-5: Rover Item Detection Milestone below displays the intended completion dates for all major development phases of the SARS Rover item detection subsystem. Development of this project component is currently 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 10-10-5: Rover Item Detection Milestones

Table 10-10-6: Rover Communications Milestones below displays the intended completion dates for all major development phases of the SARS Rover communications interface. Development of this project component is currently 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 10-10-6: Rover Communications Milestones

Table 10-10-7: Android Application Milestones below displays the intended completion dates for all major development phase of the SARS Android application. This project component is currently 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 10-10-7: 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 10-1: Quadcopter Milestones below displays all major tasks necessary for the completion of the build stage of the SARS Copter development.

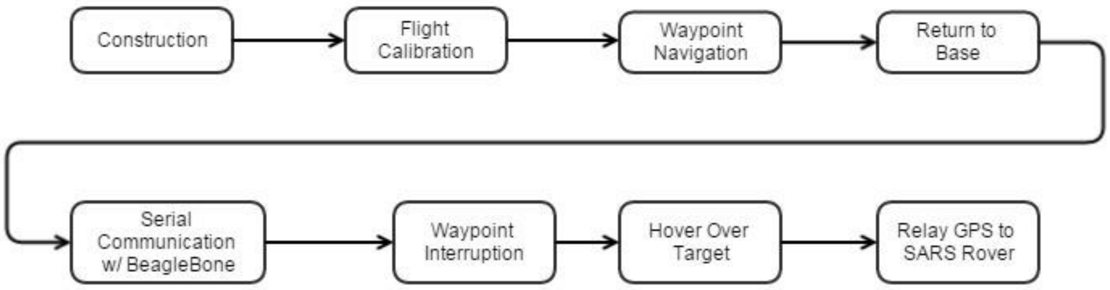


Figure 10-1: Quadcopter Milestones

Figure 10-2: Image Processing Milestones displays all major tasks necessary for the completion of the build stage of the SARS image processing subsystem development.

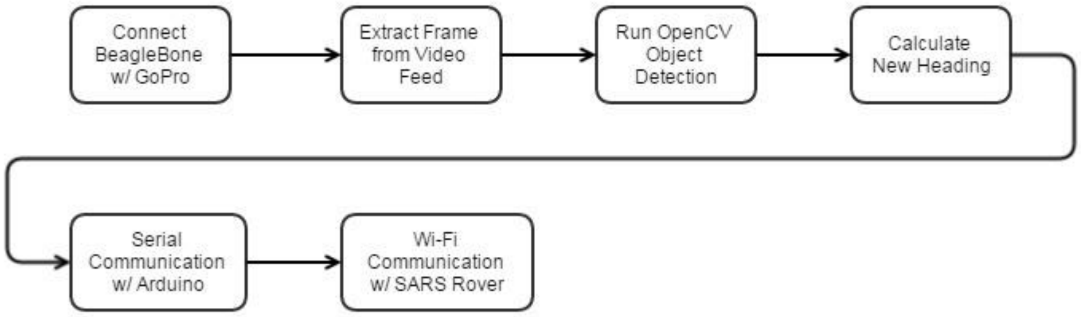

Connect BeagleBone w/ GoPro
Extract Frame from Video Feed
Run OpenCV Object Detection
Calculate New Heading
Serial Communication w/ Arduino
Wi-Fi Communication w/ SARS Rover

Figure 10-2: Image Processing Milestones

Figure 10-3: Rover Milestones below displays all major tasks necessary for the completion of the build stage of the SARS Rover development.

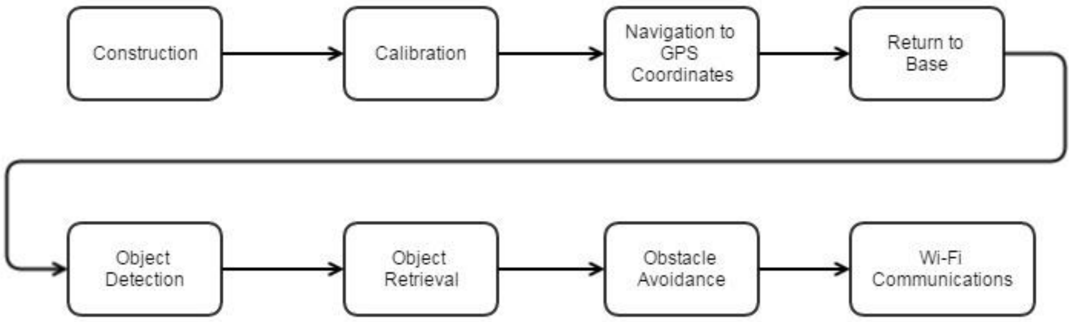


Figure 10-3: Rover Milestones

Figure 10-4: Android Application Milestones below displays all major tasks necessary for the completion of the build stage of the SARS Android application development.

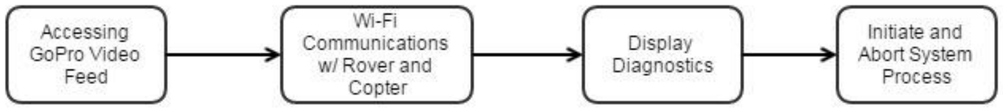

Accessing GoPro Video Feed
Wi-Fi Communications w/ Rover and Copter
Display Diagnostics
Initiate and Abort System Process

Figure 10-4: Android Application Milestones

10.2 Budget and Financing

Group 4 has received $500 from SoarTech and $1000 from Boeing in sponsorships for a grand total of **$1500** in funding. This amount of funding should be more than enough to cover all hardware and testing needed to implement the complete search-and-retrieval system. The largest portion of project expenses comes from the quadcopter. Depending on the pre-installed capabilities of the copter and whether or not it is pre-assembled, it can realistically run up to $1000. Group 4 decided to go with a quadcopter costing $550. This cost will be 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 need 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 will be the rover chassis, motors, power supply, and motor controller, along with the sensors for object detection and the object retrieval apparatus. So far, 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 will not be paid for using sponsored funding, there is not currently a significant risk of running out of funds. In the event that all sponsored funding is depleted, Group 4 has decided to pool personal funds to cover any remaining costs.

10.3 Bill of Materials

Below in Table 10-8 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	Cost	Notes
Quadcopter	$550.00	Copter frame, Pixhawk flight controller, 3DR power module, 4 ESC's, propellers, u-blox GPS, wiring and screws
QC Camera	$200.00	Go-Pro Hero 3
Xbee Wifi Module (2)	$60.00	
Rover	$400.00	Chassis, motor, motor controller, and sensors
Object Retrieval Apparatus	$50.00	
Rover Microcontroller	$20.00	TI Tiva C Series
Xbee SIP Adapter	$31.50	For mounting the Xbee Wifi Module
Ping))) Ultrasonic Sensor Kit	$45.00	Includes Mounting Bracket and 180° Servo
GPS Chip	$50.00	
PCB	$60.00	
Tennis Balls	$30.00	Items being detected
Xbee USB Development Board	$80.00	
Beaglebone Microcontroller	$0.00	Donated
BeagleBone Xbee Cape	$9.00	For mounting the Xbee Wifi Module
BeagleBone Power Source	$50.00	BeagleBone case, voltage regulator, 6 AA batteries, OMRON A3DT-7111 push-button switch, OMRON A3DT-500GY LED
3DR 915 MHz Telemetry Radios	$100.00	
Turnigy 6X Radio Control Transmitter and Receiver	$30.00	
Total	**$1,765.00**	

Table 10-8: Bill of Materials

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

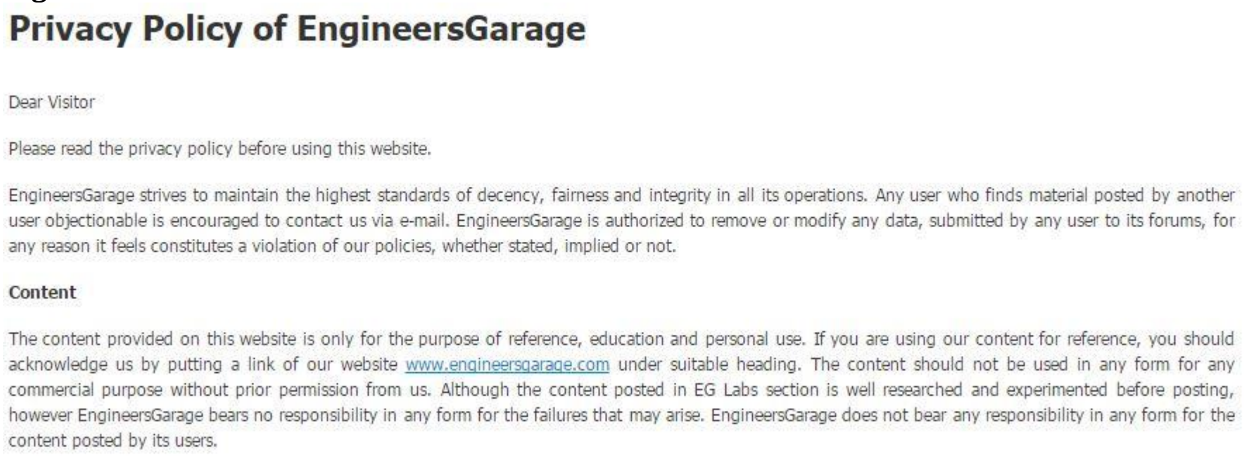
Figure 4-2: "Cascade Classifier" by Paul Viola and Michael Jones

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 10-5: “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