Military Surveillance Robotic Vehicle

The University of Central Florida
Department of Computer Science and Electrical Engineering
Dr. Lei Wei
Senior Design I Group 23

Austin King	CpE
Kevin Plaza	CpE
Adam Baumgartner	EE
Ryan Hromada	EE

Table of Contents

1. Executive Introduction

Military personnel face many unique challenges on a daily basis in their work compared to civilians. Many of these challenges can be aided or overcome by having more information at the right times. Intelligence is generally gathered in many ways including covert operations, communication interception, interrogation, aerial surveillance, and ground surveillance. All of these techniques help gather intelligence that can promote successful missions and save the lives of troops on the ground. One of the most direct and relied upon forms of this intelligence is ground surveillance. Ground surveillance is so key because there is no more reliable intelligence than a person on the ground having visual confirmation of a target. Unfortunately, ground surveillance can also be the most dangerous form of intelligence gathering because of the proximity to the target that is required by the personnel on the ground. Covert operations are designed to be secretive, interrogation doesn't involve risk to the personnel conducting it, and aerial surveillance can now be easily done by unmanned drones. Ground surveillance however typically requires a soldier or reconnaissance team to physically see the target, putting them at risk. This creates a need for an unmanned device that can be used on the ground to perform the reconnaissance without the need of a human being with it.

The simplest solution is to have a trained personnel operate a robotic vehicle remotely to gain a visual of the target area with a camera. The simplest design of this would be to just have a remote controlled vehicle with a camera strapped to it. Some of the problems of this are that the range of the vehicle is severely limited due to the use of RF to communicate to the vehicle. This means that once the vehicle reaches its maximum range, the operator would have to move closer to regain control. The second problem is that the operator is required to be sitting with the remote controlling the vehicle while viewing the video that is being sent. This means that a dedicated operator must be selected and trained with the vehicle. If the remote could be even more portable by removing the use of physical controls from the remote and instead making the vehicle autonomous, it would free up the operator. Also, if the vehicle could perform certain tasks completely autonomously, this would take away a lot of the responsibility of the operator. The operator could simply tell the robotic vehicle where to go and what task to perform, and the vehicle will do the rest.

Our design seeks to improve on the simple solution by allowing for an autonomous robot to be given commands via a portable handheld device using only voice controls. This robotic vehicle will take in the voice command from the operator, and using its built in software, will carry out the command until the operator gives it a new command. The vehicle will have the ability to carry out three tasks, which are to scout a desired area, track a designated target, or find a designated target. These functions are key in any ground reconnaissance mission to allow for appropriate information to be gathered. For example, a user

could first give the command to scout an area by giving it a distance and direction, causing the vehicle to travel in the specified direction and distance. Once it reaches the desired area, it will perform a routine to move around the area to collect a full range of visual data with its camera. The user can then give the command to find a target (based on a pre-programmed designation, such as a color or specific target), which will then make the vehicle search the area again until it sees the target. Once the target is sensed, it will lock its camera onto the target. Finally, the track command can be given to get the robot to move in the direction of the target, so that the robot can follow it even if it tries to move.

2 Project Description

2.1 Project Motivation and goals

Our project seeks to improve the solution of robotic ground surveillance by developing an autonomous robotic vehicle using computer vision and ultrasonic sensors that will aid the military personnel in finding a point of interest such as an injured individual, bomb, enemy combatant, vehicle, and much more. The robotic vehicle will have three primary functions to demonstrate how our robot could achieve this. The functions being survey for a point of interest (POI), track a POI, or find a POI. Surveying a POI means that the robot will move to the target location and use its camera to gain a visual of an entire area. This is a basic reconnaissance mode that is useful gaining information on an area and for finding a potential target. The “track” command will be used to follow a desired target. A color or special target will be designated as the tracked object, and the robot will be programmed to recognize the designation and once it is locked, it will follow the target and keep a visual of it in the camera. The “find” command will cause the robot to find a target with the designation. Along with the survey function, the robot can move to a desired location and search an area for a target, then with the track function the robot can lock onto that target and follow it until the command is canceled.

All of the mentioned functions would aid in ground reconnaissance by making it so that the robot can autonomously perform the reconnaissance without having a dedicated operator constantly controlling the robot. The controlling device would be simple to use, so that anyone could operate the robot, and the commands would allow for the user to simply give a command and the robot will automatically perform the task while freeing the operator to do another task while the robot performs its mission. These three functions mentioned will be activated from a voice command given into a smart phone or device by the military personnel.

Our motivation for choosing this project is that we would like to incorporate a project the includes equal elements of Computer Engineering and Electrical Engineering so that we can use the full spectrum of our degrees equally since we have two members from each discipline. We feel that his project offers unique challenges by designing smart algorithms that allow the user to interact with the vehicle in a much more unique fashion. Unlike some other robotics projects, our design is aimed at being a functional real world device that can be a useful tool for military or police.

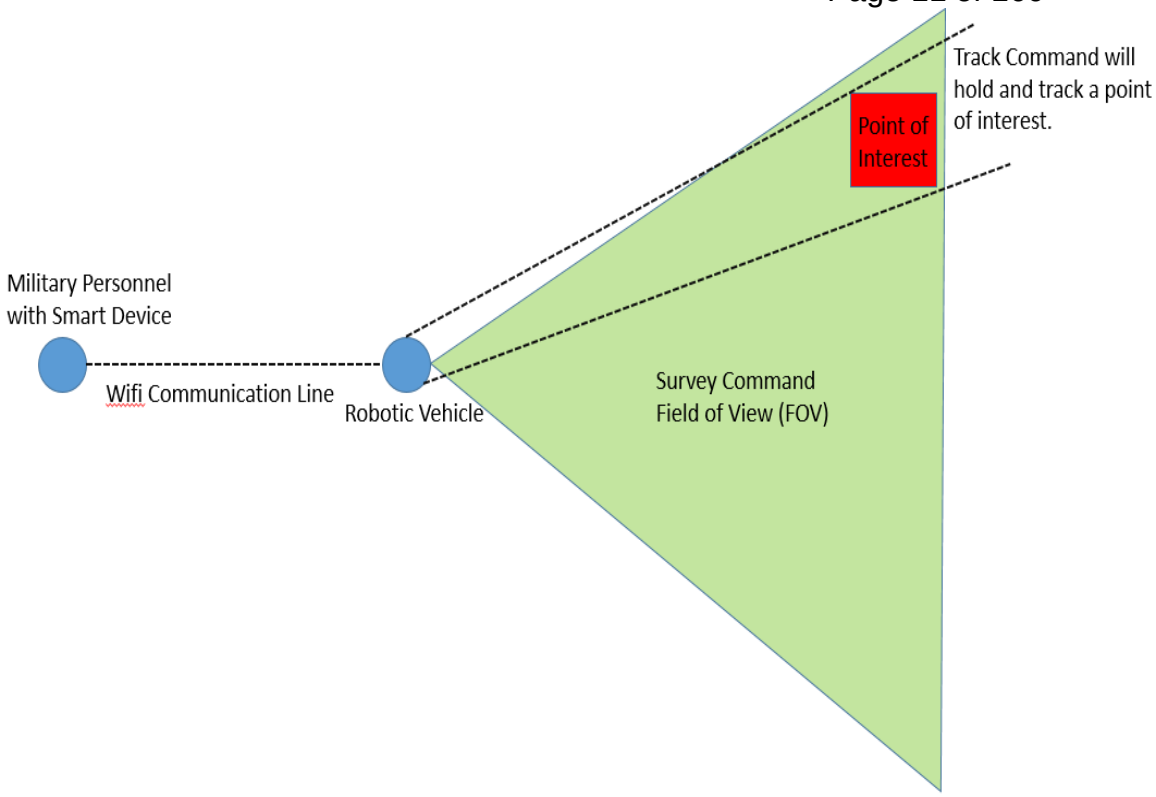


Figure 2-1: Functional Diagram

The robotic vehicle will be Wi-Fi enabled to allow for communication of the voice commands from the user to the vehicle for which mode to enter. This will also be used for the capability to provide camera feed back to the user. This will provide the user with the ability to see what is being surveyed, tracked, and/or found. The camera mentioned will also be used for the computer vision section of our project allowing the robot to distinguish between what is a POI and what is not. The last addition to the base robotic vehicle is ultrasonic sensors that will be positioned around the vehicle to allow it to run autonomously without collision.

2.2 Objectives

Below is a list of objectives that we have determined are a requirement for our vehicle to meet. Each of these objectives are will be explained in the following subsections.

2.2.1 Autonomous Operation with voice commands

One of the major components of our design is the ability for the robotic vehicle to be controlled through voice commands. This is desirable because it allows the operator to command the vehicle without the use of manual controls. This means that the controller can be a light and small device that leaves the operator free to perform other tasks while still having control of the vehicle. The vehicle must also be able to operate autonomously. The more that the vehicle can operate on its

own, the less attention that the operator has to put into controlling it. The operator should be able to simply give a command and then the vehicle should be able to carry out the command without extra input from the user. The operator can then simply watch the display to gain the visual information without having to worry about steering the vehicle.

2.2.2 Operation in different modes

Our design will feature three modes of operation to carry out different types of tasks. These three modes will be “survey”, “find” and “track”. The function of these three modes will be implemented in the software of the vehicle with the use of a microcontroller. The external components such as the servos and camera shall be interfaced with the software to serve as inputs or outputs. For example, when a command of “track” is given through the controller device, the software shall recognize the keyword and execute a subroutine that reads the input from the camera to find the designation. Then the software shall output to the servos the signals required to move the vehicle in the direction of the target while reading the input from the ultrasonic sensors to avoid collisions. A similar procedure shall be executed for the other two modes to ensure correct operation.

2.2.3 Integrating multiple components onto a Printed Control Board (PCB)

From the requirements of the design project, we must incorporate a PCB into the design. With multiple sensors and servos utilized in our design, the PCB is required for us to combine all the electronic components together and provide power to the system. Some of the components can be directly interfaced with the microprocessor so they will not be included directly in the PCB design. The overall block diagram of the system is shown in Figure 2-2.

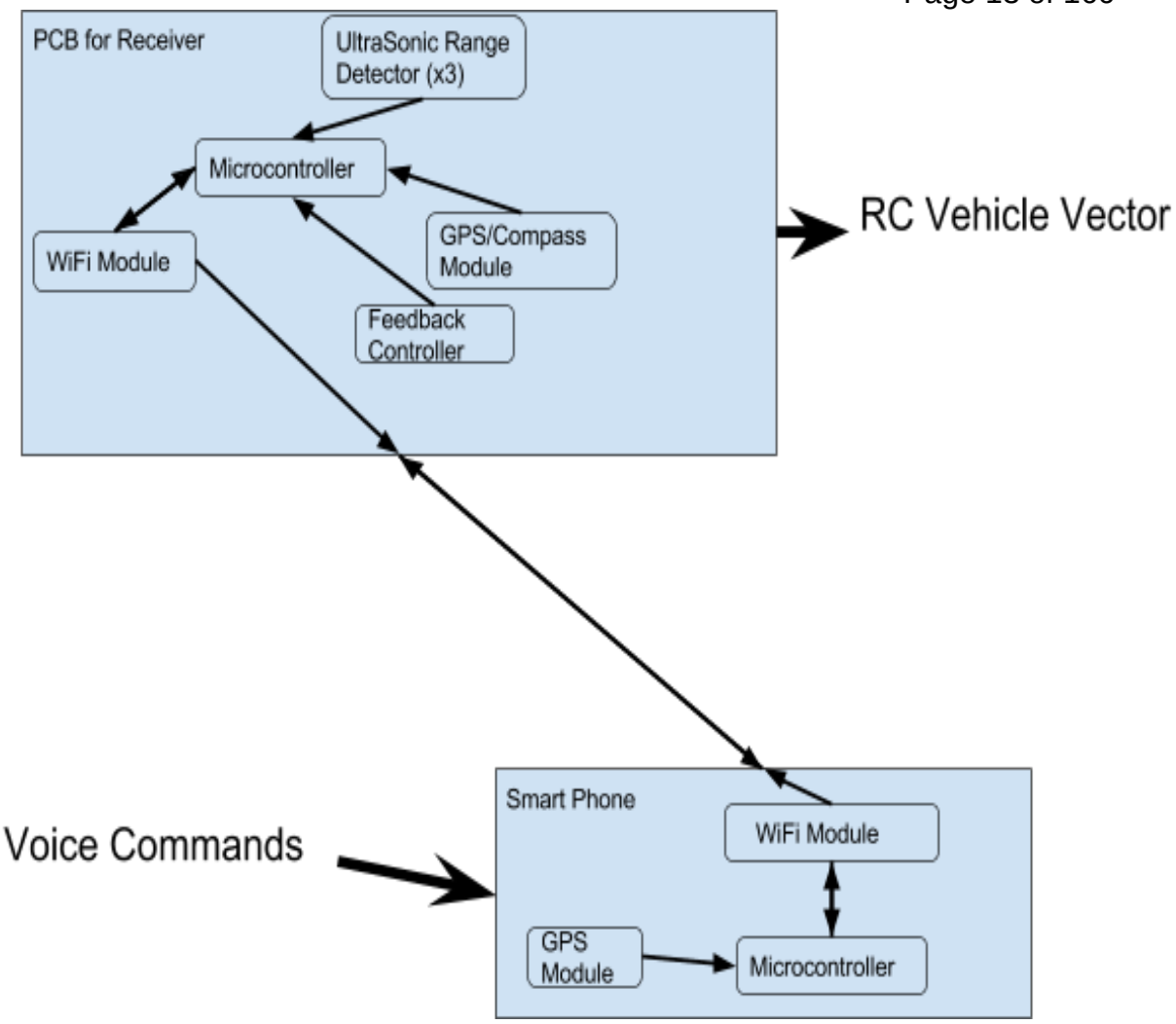


Figure 2-2: Functional Diagram

2.3 Specifications/Requirements

The specifications to achieve our objectives will give a more detailed approach on how we will achieve our final design. These specifications and requirements are of much greater detail then the objectives and will influence the strategic parts selection that we discuss later in this report.

2.3.1 Physical/Electrical components

- The robotic vehicle shall utilize a printed circuit board (PCB)
- The robotic vehicle shall be no larger than 2’/2’/2’ (L/W/H)
- The robotic vehicle shall have an internal battery
- The robotic vehicle receiver will have a feedback control system
- The robotic vehicle shall utilize sensors for collision detection
- The robotic vehicle shall be Wi-Fi enabled.

- The robotic vehicle receiver shall utilize a microcontroller to process data and communicate to the smart device.
- The robotic vehicle shall utilize a camera for basic vision

2.3.2 Functional Requirements

- The robotic vehicle shall accept functional commands over Wi-Fi
- The robotic vehicle shall be able to scan a 120-degree area in front of itself.
- The robotic vehicle shall stop when it reaches the specified location or is prompted to by voice command.
- The smart device shall communicate to the robotic vehicle wirelessly
- The portable device shall receive voice commands including: survey, track, find, and stop
- The robotic vehicle shall have a response delay of less than 3 seconds from the user giving the command to the vehicle responding.
- The robotic vehicle shall have a maximum range of at least 25ft
- The robotic vehicle shall provide imaging back to the user from the vehicle's camera.
- The robotic vehicle shall detect a predefined point of interest.
- The robotic vehicle shall be able to find a point of interesting that is 5ft away from the vehicle.
- The robotic vehicle shall hold a track on a POI for at least 30 seconds.
- The robotic vehicle will stop within 10ft of giving the stop command.

2.3.3 Software requirements

- The smart device shall receive voice commands as input.
- The software shall command the vehicle through Wi-Fi based on the input command
- The software shall determine whether a track is being held or if the vehicle needs to reacquire.
- The software shall prevent the robotic vehicle from colliding with objects.
- The software shall identify a predetermined point of interest (POI).
- The software shall allow the robotic vehicle to track a POI.

2.3.4 Marketing requirements

- The robotic vehicle shall be portable
- The device shall be intuitive for a new user
- The robotic vehicle shall be accurate in terms of tracking and response time
- The robotic vehicle shall have a cost of no more than $450
- The device shall be programmed with smart functionality, ability to prevent collision, run autonomously after a given command until command is

stopped or new command is issued.

2.3.5 Project Constraints

- Financial constraints due to no sponsorship and average funds of college age persons.
- Time constraints 8 months are given for the development of an idea, documentation, and finished product.
- Experience constraints, members have little to no industry experience and are unfamiliar with project management, design analysis, requirement standards, documentation, and others that will be dealt with during this project cycle.

2.3.6 Project Standards

- Group will use Google documents for easy access and updating for all.
- IEEE will be researched and referenced if applicable to our project.
- Lei Wei's designated documentation standards will be follow

2.4 House of Quality

The House of Quality shown in figure 2-1 represents how each of the engineering functionalities of our design affect the marketing variables. Each engineering feature is based off our requirements specified in section 2.3. Understanding how each feature affects the marketability of our design is important to creating a useful product.

			Marketing				Targets
			Cost	*Accuate*	*Intuitive*	*Portable*	
			-	+	+	+	
Engineering	*Dimensions*	-	↓	-	-	↑↑	24''/24''/24'' (L/W/H)
	Wifi Enabled	+	↓	↑	↑	↑	≤ 1Kg
	Voice Control	+	↓↓	↑	↑↑	↑	Accurate voice commands
	Battery Operated	+	↓	-	-	↑	>2 Hours
	Range	+	↓	-	-	-	150 ft
	Response Time	-	↓	↑	-	-	≤ 3s
	Collision Detection	+	↓	↑↑	-	-	Avoid collisions causing loss of track or damage to the vehicle
	Cost	-	↓	-	↓	↓↓	≤ $350

Table 2-1: House of Quality

2.4.1 Dimensions

The dimensions of the vehicle affect the portability and cost of the design. The size of the vehicle directly correlates to the overall weight and viability of the design. Since the idea of our design is to create a small and efficient design that could be easily deployable in the field, the dimensions must be kept small enough to make it feasible to transport and deploy the vehicle with only one operator. Because of this, our requirement specifies that the vehicle shall be smaller than 24''/24''/24''. This was determined to be small enough to deploy easily. The larger the design, the harder it is to carry which reduces portability. Therefore, the dimensions have a strong positive relationship to portability. The tradeoff is that lowering the dimensions requires smaller components and a more compact design, which can be more expensive, thus giving a negative relationship to cost.

2.4.2 Voice Control

With voice commands being a critical part of the design, the impact they have is large in regards to the marketing requirements. The better the voice control is, the less effort is required by the operator to control the vehicle, making it more portable and much more intuitive. Having effective voice control also makes it less likely for miscommunication which helps the accuracy of the vehicle. The resources required to make more effective voice control highly depend on the technology of the smart device and communication lines between the device and

the vehicle. Better technology can improve the voice control, but at the cost of more expensive components for both the device and the vehicle.

2.4.3 Battery Operated

One requirement that was set was for the vehicle and device to be powered by self-contained batteries. This means no external cords or wires should be necessary to power the device or the vehicle. The battery life of the self-contained power source should also last enough to carry out a reasonable mission, which can be set as two hours. The longer the batteries last, the longer that the vehicle can be deployed without having to return to the user to be charged. This helps the portability of the product. The tradeoff is that a larger battery would be more expensive.

2.4.4 Wi-Fi enabled

With Wi-Fi now easily available and an effective form of communication, using Wi-Fi will help make our design much more marketable. The wireless form of communication allows for greater portability, and the ease of setting it up makes it more intuitive for the operator. The effectiveness of Wi-Fi will also make the vehicle more accurate since it can communicate better to the controller device. However, the use of Wi-Fi requires specialized modules which will increase the cost.

2.4.5 Range

The range of the vehicle with respect to the smart device is another factor in making the device useful for military applications. The range of the vehicle will be dependent mostly on the Wi-Fi connectivity. Having this increased Wi-Fi depends on the technology in the Wi-Fi module, which can be more expensive.

2.4.6 Response Time

Having a reasonable response is required for effective communication between the operator and the vehicle. If the operator needs to switch to a different mode, they must be able to do so in a short amount of time to be sure no unwanted behavior is seen with the vehicle. Thus, the response time needs to be lowered as much as possible. This will increase the accuracy by making commands happen as soon as the operator gives them. Any delay in response could reduce the effectiveness of the vehicle. Lowering the response time requires better communication technology and higher processing speed, which causes the need for a better microcontroller.

2.4.7 Collision detection

Collision prevention is a necessary component for an effective reconnaissance robot. Not only is it required for effective movement, but in order to track a target it must be able to consistently move towards the target without being blocked by

obstacles, as hitting an obstacle could cause the robot to get stuck. The better the detection capability, the more reliable and accurate the vehicle will be with all of its modes. It also prevents the vehicle from suffering damage by running into a wall or into another moving object.

3 Research related to Project Definition

3.1 Existing similar products

The need for an unmanned reconnaissance vehicle has been one the military has been trying to fulfill for a while. They search for robotic vehicles that are small enough to be easily deployable and be able to carry reconnaissance in a stealthy manner. Many of these products already in use are large, clunky robots that can weigh between 40-80 pounds, making them hard to deploy as they require special transportation. A cutting edge product called the Throwbot XT made by ReconRobotics fills a similar role that this project seeks to create. As shown in figure XIt is a small, lightweight reconnaissance system that uses an infrared optical system and video to provide valuable intelligence. This system however requires an operator to deploy it and then manually control it, so it is not autonomous.

Figure 3-1: Throwbot XT vehicle made by ReconRobotics [1] (Permission Granted)

As for voice controlled robotic systems, there aren't many in existence for military or other practical applications. Most voice controlled robots currently available are simply toys with very little voice control functionality. This shows that there is an opening in the market for a robot that can commanded by only voice for practical applications.

The closest existing product that is similar to the product we have conceived is

an urban reconnaissance robot developed by the California Institute of Technology. This robot is made to be robust to survive dangerous urban environments. It combines obstacle avoidance and programmed goal seeking behaviors to be able to carry out user defined missions. It can operate in multiple modes, ranging from manual control to fully autonomous. It uses a sonar proximity sensors and a camera for video. It also incorporates a laser rangefinder to assist in obstacle detection. As previously discussed, the disadvantage of this robot is portability. It weighs roughly 44 pounds, making it difficult for a person to carry and deploy anywhere. It typically must be transported by a vehicle and then deployed.

3.2 Relevant Technologies

There are several industry standards that will be explored in the following sections when discussing the components. These standards are often the relevant technologies used and adopted by many manufacturers. These technologies allow for ease of use in the design of the device and the speed of the data. These technologies are i2c, CSI, MIPI, PWM and Analog based signals. These communication protocols determine how the devices interact with each other and ultimately impact the overall design structure. Certain communication protocols will utilize different areas of the MCU and based off that will determine the PCB board layout structure as well as the physical device structure.

3.3 Strategic Components and Part Selections

The strategic component and part selection process is meticulous in nature in that it must satisfy the requirements of the RSV in order to properly fulfill the objectives of the design. Ideally we want to be on the lower end of power consumption and size. The components won't necessarily take a minimalist approach, however, more weight will be given to components with smaller sizes and power consumption except in the instances where more processing power is required.

The below sections will cover the components that were selected to be used and why they were chosen. The sensor arrays were chosen for their abilities to perform the desired role of collision avoidance, the ability for the device to interpret movement axially and for the device to understand the speed of the individual electric motors. These three roles are the primary ability for the device to function in the test environment. The MCU was chosen on its ability to interface with coding. For this project a higher end MCU will be chosen to ensure that the device is capable of operating autonomously. Any future revisions of this project should consider the use of a lower powered MCU as some of the features of the device will not be used.

3.3.1 Ultrasonic detector

In order to safely operate autonomously the vehicle needs the ability to actively avoid object detection. The ability requires some sort of sensor feedback to the MCU in order for the processor to determine whether the object is within range of collision with the vehicle. There are variety of methods for collision detection used in robots, however, the main methods are Infrared, Sonar, and Laser. Ideally to keep a lower power setup you would choose the method to best suit the project's goals. The key characteristics to judge the viability of the component for the project are the detecting distance of the component, the power consumption of the component and the. This means where the detection can start and where the detection can stop. Some devices are highly accurate at close ranges of around 1-2 cm whiles others can range up to 900 cm. Some of the advantage to using the Sonar is that the average angle for measurement is 30 degrees from the point of detection and minimal power consumption of around 15mA. The figure below shows that as distance increase the device becomes less precise with its readings and has a reduced margin of error.

Below shows a picture of the Ultrasonic detector connected to a breadboard used to test the effective range and width of detection. The picture is used for reference only and does not represent the actual testing implementation of the device. As you can see in the below picture the device has a reasonable degree of confidence up to 6 feet and a width of about 30 degrees. However, the further the object is from the device the less accurate the device will be if the object isn’t directly in front of it.

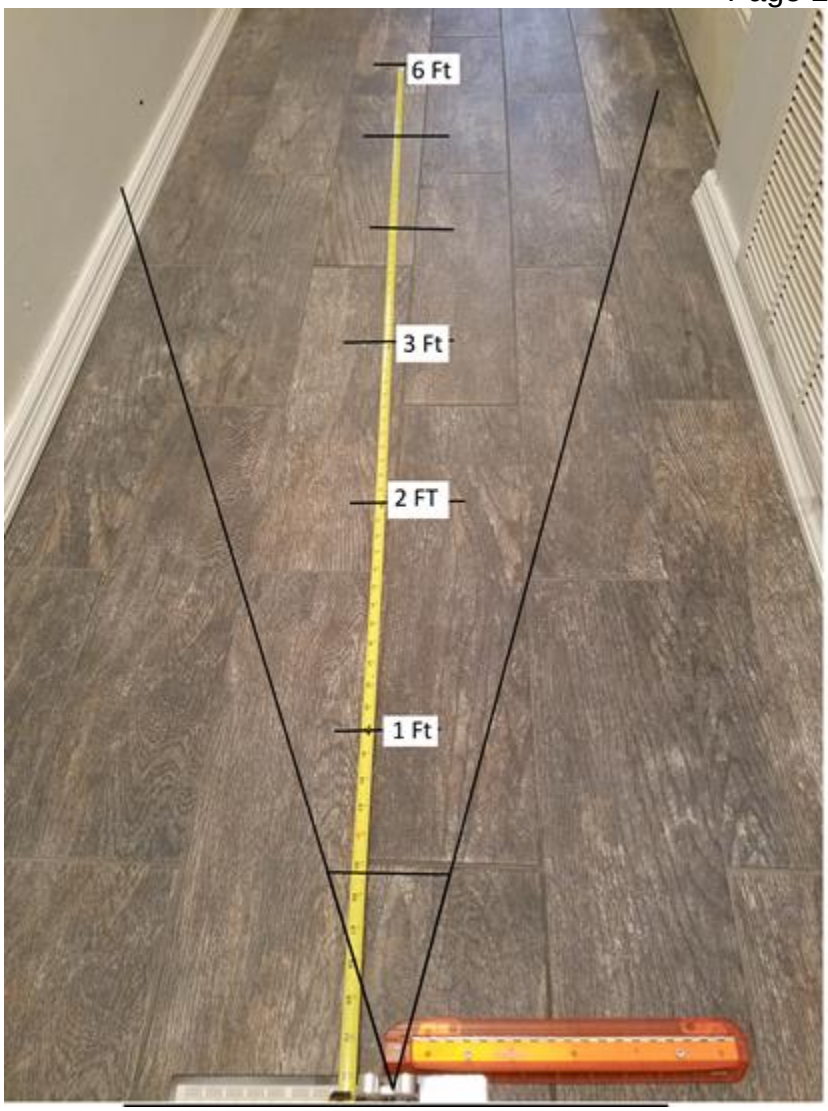


Figure 3-2: Example of viewing angle of ultrasonic sensor

When looking into the laser based applications most of the laser detection tends to be a narrow focused beam which eliminates the ability for the sensor to be stationary mounted. The power consumption for the systems tend to be higher when receiving data but have a low idle point. Since the device needs to be turned on and the angle of detection needs to be sufficiently wide the laser based sensor was not selected.

The infrared based sensor use much less power in comparison to the Ultrasonic or sensor based sensor and would not be susceptible to matching frequency interference. The infrared sensors also take up less over all space and are efficient at close range detection. These sensors would be ideal for our

applications however their range is shorter in comparison to the ultrasonic sensor with long range sensors reaching up to 5 feet. The infrared sensor needs a minimum distance to function of 10-15 cm while the ultrasonic sensors can typically operate at a minimum distance of 2 cm.

After choosing the ultrasonic sensor for its better fit qualities for the project design the next important consideration is the method of communication to the MCU. The Ultrasonic sensors can come in a variety of output methods, Pulse Width Modulation, Analog and serial communication. Depending on the quality of the one or more of these options may be available. We chose a module the specified the Pulse Width Modulation method as this correlated with a lower cost component. The PWM works by initializing the unit through the sending of a 5-volt source for at least 10 µs. This triggers the unit to emit a series of pulses at a 40 kHz frequency. The pulses then bounce off an object and return to the receiver where they are echoed back to the MCU via 5-volt source for a period of time. The length of the time corresponds to the distance where the distance in centimeters is equal to the time output divide by 58. The time of the pulse is measured in a range of 150 µs – 25 ms. The formula is:

$$\frac{Time * 10^6}{58} = distance(cm)$$

The signal output is detailed below. The output response signal shows the length of the Pulse that is used to determine the distance of the object.

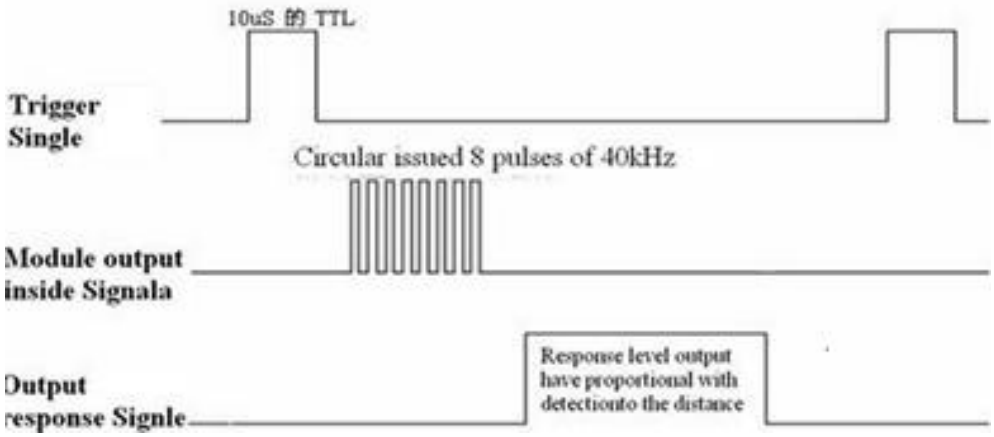


Figure 3-3: Timing diagram for ultrasonic sensor [15] (Permission Pending)

Based off the selections for vehicle collision avoidance we will need three sensors to accurately guide the vehicle through its directives while avoiding obstacles. By placing them on the sides and the front of the vehicle the vehicle will be able to determine an obstacle upcoming in its path.

Since we do not intend for the vehicle to work in a dynamic environment we intentionally left off a fourth detector in the back of the rear of the vehicle as the tests will be done in a controlled and static environment. The one important consideration is that the pulse needed to activate the sensor is 5 volts, found on most MCU's however the return pulse will be 5 volts so we will need a simple voltage divider to reduce the voltage as most MCU I/O pins operate at 3.3 volts or lower. When introducing this component to the PCB board we will need to ensure that this remains separate from the components that utilize the i2c protocol. As this device communicates exclusively with a basic PWM I will need to be connected to the GPIO line of the MCU.

3.3.2 Camera Module

When determining the camera choice for the autonomous robot design the choice was slightly more difficult than expected. While many devices make use of optical sensors and camera modules with varying pixels, it was difficult to find an industry standard when it came to MCU's. There some Display Serial Interface standards utilized by various manufacturers for Cameras one of which is the Camera Serial Interface (CSI) which is a specification of the Mobile Industry Processor Interface Alliance (MIPI). These were developed to define an interface between a camera and a host processor. The idea for the industry alliance was to allow the system integration of the interfaces to have less involved and make the processor easier and smoother for the companies involved. [13]

When it comes to the MCU's a few of the products have adopted the while some products have adopted the i2c serial interface to allow fewer input lines and faster communications. There are some modules that also utilize traditional UART interfaces allowing the user to customize the Baud rate based on communications protocols. Given that most of the cameras designed to interface with the various MCU's are sufficiently small we chose to go with the Raspberry Pi Camera for two reasons. The main reason is that the camera was designed to interface with our MCU choice. This allows for less debugging and less initialization of the camera based software. This particular camera meets a minimalist design and weight while having a robust interface following the MIPI Alliance standard. Our second is reason is that not only does it follow the MIPI standard is that it also follows the CSI specification. By using this specification, the camera can connect directly to the Raspberry PI GPU and bypass the CPU. This gives an increased performance as you can have the data sent straight to the GPU for encoding.

Due to the length of the cable we will most likely need to add extensions to the length. This will allow the camera to be mounted on a separate physical platform that can be raised above any jumper wires and subsystems of the RSV. The below picture illustrates the size of the camera relative the Chassis of the RSV.

The overall size of the camera has to be small enough as to not interfere with the multiple subsystems

Figure 3-4: Raspberry Pi Camera

Ideally the camera should operate as its own independent smaller subsystem with direct connection to the MCU for both power and data. The reason is that this will allow the camera to operate optimally via the CSI interface directly to the GPU.

3.3.3 Processing Unit

In order for our robotic vehicle to accomplish its complex tasks, a fairly powerful controller is required. This controller should have Wi-Fi compatibility as well as enough I/O pins to support having a camera, ultrasonic sensors, and servo control. It should also be inexpensive and fairly easy to integrate into our design. Since both digital and analog signals will be used, the microcontroller must be able to interface both signals. The two controllers that may meet these criteria and were considered are the Raspberry Pi and the Arduino. Specifically, the Raspberry Pi 3 and Arduino UNO.

3.3.3.1 Arduino

The Arduino UNO, shown in figure 3-5, is a true microcontroller that is made to

be simple to use and offers basic functionality of a microcontroller to help reduce power costs. It specializes in interfacing with sensors and devices and running simple programs that can monitor the states of these sensors and devices. This makes the Arduino a good selection for embedded systems, where a low power microcontroller that can perform a simple task is required. However, the Arduino is capable of performing greater tasks. The Arduino has available “shields”, which are expansions that offer different functionalities to the base Arduino board. These can be used to obtain specific functionalities based on the desired application. Some examples of shields are breadboard shield that allow for prototyping, or a shield that allows for Wi-Fi or Ethernet connectivity. The Arduino UNO has an operating voltage of 5V with a recommended input voltage of 7-12V. The I/O pins draw 20mA of current per pin. The 3.3V pin draws a current of 50mA.

Advantages:

- In terms of software, the Arduino uses its own IDE that is based on java with its own compiler. This makes it relatively simple to code basic programs into the controller.
- comes with both digital output pins and analog input pins. This would make it so that both digital and analog signals could be interfaced easily.
- Some of our components are also built for the Arduino, which would make it much easier to interface them with the board as well as program with the board.

Disadvantages:

- The downside of the Arduino is that the hardware limits software compatibility due to low memory size and processing speed, as indicated in the specification table 2.
- Since the board was meant for more simple embedded processes, it can’t handle large data manipulation or calculations. Since our project will require graphical processing and constant pinging of input/output devices, a powerful processor and large memory size are required that can handle the amount of data and processing required to carry out a surveillance vehicle with collision prevention and target tracking.
- To support such processes as computer vision on an Arduino, there are solutions. A separate image processing module could be attached, which would process all imagery and recognition and then send signals back to the Arduino board to then send the outputs to the servos. This will add an extra step and component to the design which could lead to increased power consumption or increased complexity.

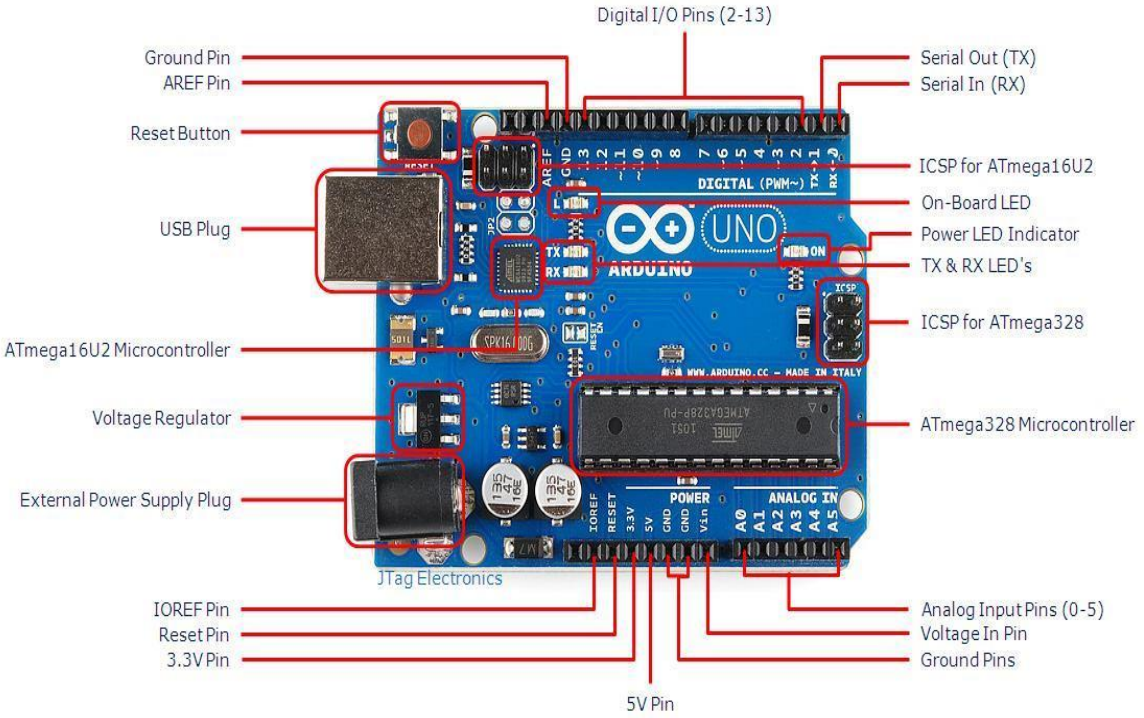


Figure 3-5: Arduino UNO REV 3 [2] (Permission Pending)

3.3.3.2 Raspberry Pi:

The Raspberry pi 3 shown in figure 3-6 is a single board computer that is intended to be cheap and powerful. Unlike the Arduino, the Raspberry Pi is not a typical microcontroller. The Raspberry pi operates similarly to a normal desktop or laptop computers, as it can support specialized operating systems and has enough processing power and memory to run large programs. It is also capable of utilizing an SD card for external storage to store its programs and operating system. The latest release of the Raspberry Pi is the Raspberry Pi 3 model B. This board offers the same features as the Raspberry Pi 2, but with improvements in processing speed and main memory size. It also offers built in Bluetooth and Wi-Fi capability, which is extremely useful for our application and helps reduce costs. The full specifications are listed below. The Raspberry Pi 3 requires a 5V, 2.5A micro USB power source.

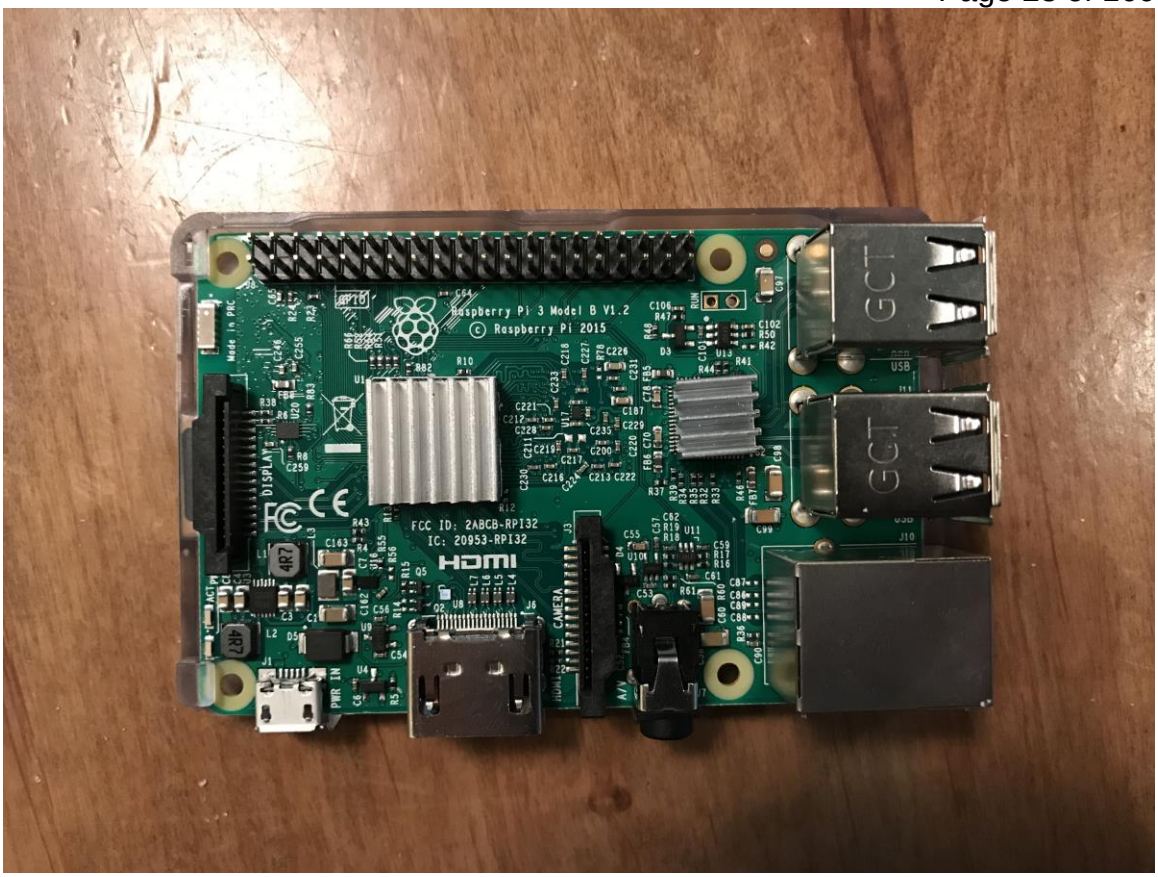


Figure 3-6: Raspberry Pi

Advantages:

- The Raspberry Pi 3 is essentially a full computer packed into a single board.
- It is more powerful, which allows for more flexibility to program complex software.
- Since it is possible to connect a keyboard and monitor to it directly and uses known operating systems, a programmer could conveniently program the board directly in a somewhat familiar environment.
- The Pi also features many built in capabilities, such as Wi-Fi and Bluetooth as well as ports for a camera, audio, and digital I/O pins as shown in figure 8.
- It uses the Inter-Integrated Circuit (i2c) communication protocol, which will allow for separate controllers to be interfaced with the Pi.
- With the high processing power and memory size, the Raspberry Pi 3 is able to perform computer vision and image processing without the use of an external module
- Large external memory size, large capacity micro SD cards can be bought for cheap

Disadvantages:

- One of the main disadvantages to the Raspberry Pi 3 is that all of its input and output are digital. Our project uses several analog signals for components such as the servos and accelerometer. This means it would be required to also purchase an analog to digital converter in order to interface the Raspberry Pi to our components. This could potentially lead to a loss of accuracy, although the loss can be overcome in the software.
- Another Disadvantage is that the cost of the Raspberry Pi 3 is higher than the Arduino. Since the Arduino is much simpler, it's cheaper than the Raspberry Pi 3 as just the boards. The Raspberry Pi 3 also requires a separate power source and SD card, which add to its base cost.
- Complexity is another disadvantage of the Raspberry Pi 3. Unlike the Arduino, the Pi requires much more setup and is more complex to integrate sensors to. Integrating external sensors requires knowledge of I2C communication as well as electrical knowledge to lower the I/O voltages down to 3.3V as required to sense low/high readings.

3.3.3.3 Summary of Controllers

The specifications for both controllers are shown in table 2 for comparison purposes. Looking at both controller's side by side, one can see the key differences in the pure numbers for both controllers. These specifications however do not solely determine if a controller is better or worse for our purposes.

Specification	**Raspberry Pi 3**	**Arduino UNO**
Processor Chipset	Broadcom BCM2837 64Bit Quad Core Processor powered Single Board Computer	ATmega328P
Processor Speed	QUAD Core @1.2 GHz	16Mhz
Memory	1GB SDRAM @ 400 MHz	2K(SRAM), 1K(EEPROM), 32KB(Flash)
Storage	MicroSD	None
USB	4x USB Ports	1x USB port (for power)
Max Power Draw/voltage	2.5A @ 5V	5V operating,20mA current per I/O port
GPIO	40 pin	12 (6 digital, 6 analog)
Ethernet Port	Yes	No
Wi-Fi	Built in	No
Bluetooth LE	Built in	No

Table 3-1: Specification comparison

In order to gain a fair comparison of both boards, a full comparison of the advantages and disadvantage must be made. Table 3 shows a side by side look at the pros and cons to assist in determining which controller fits our project better.

	Raspberry Pi 3	**Arduino UNO**
Cost	More expensive	Less expensive
Power	Faster processor, more memory	Slower Processor, very small memory
Simplicity	More complicated to program and interface with sensors	Easier to use, compatible sensors are easily available.
Compatibility With project	Requires more work from EE side to interface I/O and sensors to the controller	Easier to integrate hardware. Harder To Integrate software.

Table 3-2: Controller Pros and Cons

3.3.3.4 Final Selection

After weighing the advantages and disadvantages of both microprocessors, it was decided that the Raspberry Pi 3 would be the best choice. This was determined because the Raspberry Pi 3 offers much more power built into the board than the Arduino. In order to perform computer vision tasks, we require processor with high clock speed and a memory size large enough to carry out the tasks without stalling.

The Pi also has the capability for a large amount of external storage with separate SD cards. This means we can store large amounts of data or features in the SD card that far exceed the main memory capacity of either of the boards. The Raspberry Pi also offers much more flexibility with the software. It can be programmed in a familiar environment to make it easier to implement the features required for the vehicle.

There are tradeoffs however that must be addressed. One tradeoff is that analog to digital and digital to analog converters must be used to generate useful outputs and gather useful inputs. Also many of the external sensors and devices

are built to be compatible with the Arduino. This means the hardware side will be more difficult to figure out how to interface these components to the Raspberry Pi as opposed to if the Arduino was used.

3.3.5 Magnetometer

When choosing the digital compass, the primary focus was to have the device to determine its current position relative to the magnetic north. The reason this information is needed is that the device can accurately determine where it went to help created a 2d map of travel. This along with an accelerometer can provide two methods to determine the amount the vehicle rotated. This will allow the vehicle to know the direction it has turned in to help determine its vector when plotting the map. The sensor arrays for the magnetometer has become limited in choices with most variations of the components are of higher accuracy or include extra sensor technology. At first our original thought was to only include a digital compass however many components are sold with triple-axis accelerometers and magnetometers combined.

We have decided on a lower cost solution that has an accuracy relevant for the project and the tasks that are needed to accomplish. The unit comes recalibrated, however, the component can be calibrated using the MCU or a traditional PC. This will help to determine axis offsets for both the accelerometer and magnetometer.

The device uses i2c communication protocol. The i2c communication protocol will reduce the wiring needed between the device and our MCU. Rather than having an input pin for each axis in each component this allows for serial communication between the MCU and the device via to serial ports. The most important about of our calibration is to get the component level on installation so that when the device reads a change in direction its true to our vehicle. Any deviations in the placement of the component could allow for incorrect position recordings and require extra calibration steps. The accelerometer and magnetometer will read out in the X, Y and X axis and allow for the reading of orientation of the device. In the absence of the local magnetic fields the device can read the Earth’s magnetic field (20-60 micro Teslas) and determine the angle of the magnetic field relative to its position. This will allow us to utilize the X and Y axis for the compass component. [11]

At this point there is no information specifying the power consumption of continuous use nor are is there information on the ability to turn off the accelerometer if we find that it is not needed. When building the device and testing we will need to determine the average power output of the component to ensure our power supply will accurately account for the device. We will also research the ability to read only the components required to ensure we have an efficient use of code and power consumption.

The figure below shows the schematic and size of the device, the device occupies a relatively small space with minimal I/O pins. This is important as the more sensors we add to the vehicle the more physical cramping will occur causing our device to become overcrowded and more difficult to repair, troubleshoot and operate.

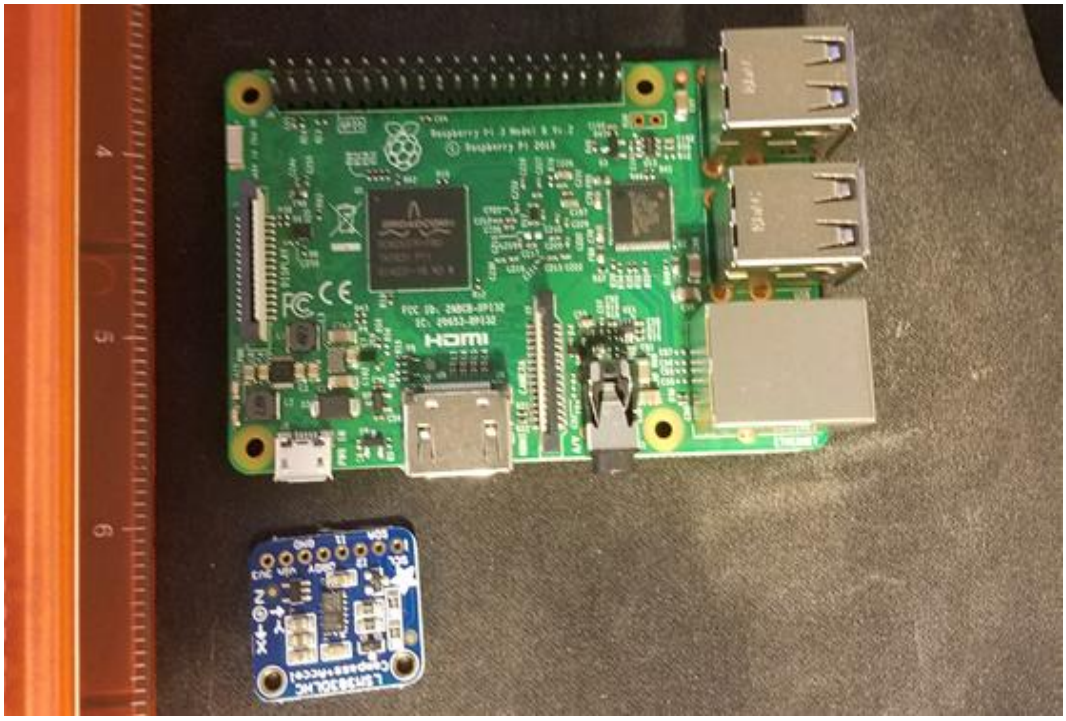


Figure 3-7: Magnetometer size

Due to the relatively small size of the device as shown in figure 3-7, we have the ability to place the component conveniently and with respect to any size constraints of the other systems. This device uses the i2c communication so we will need to consider the wiring layout to the PCB as well as any pull up resistors needed to complete the circuit. Ideally we will have the device mounted directly onto or close to the PCB board to minimize the use of the jumper wires.

3.3.6 Servo Control

The proposed rover will be driven by two electric motors. Each motor controls the tracks on one side of the vehicle chassis. At minimum, the motors need to be operated in forward and reverse modes with independent control for each engine. With this configuration, the robot would be able to drive forwards, backwards and turn. Turning is accomplished by operating one motor in forwards mode while the other is in reverse. This method of turning is known as differential steering, shown in figure 3-8. A primary drawback to differential steering is a lack of precision motion. Differential steering is easy to program and is lower cost

than comparable steering methods. The increased ease of design more than compensates for the slightly decreased precision.

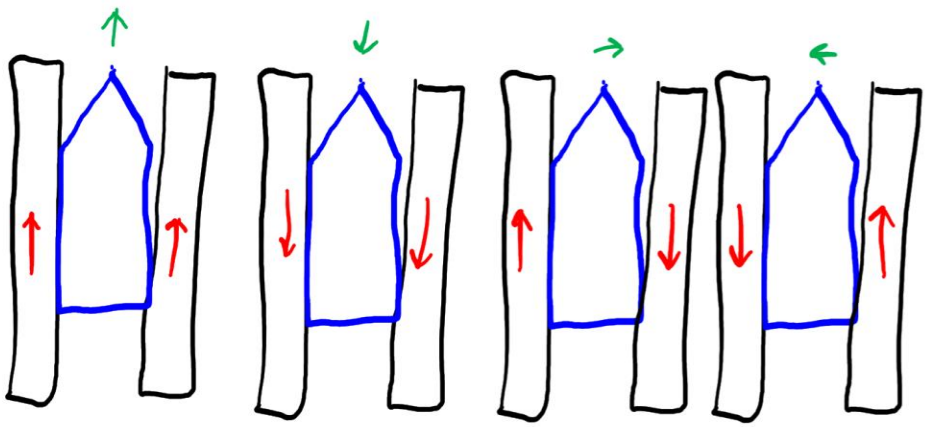


Figure 3-8: Differential Turning Diagram

Ideally the servos could operate at various frequencies to allow for motion at different speeds. The motors are driven by pulse width modulation and this is facilitated by a servo driver. Through variations in the width and frequency of pulses shown in figure 3-9 the engine can be controlled. The control enables the motors to be precisely directed and enables the functionality that makes this design feasible.

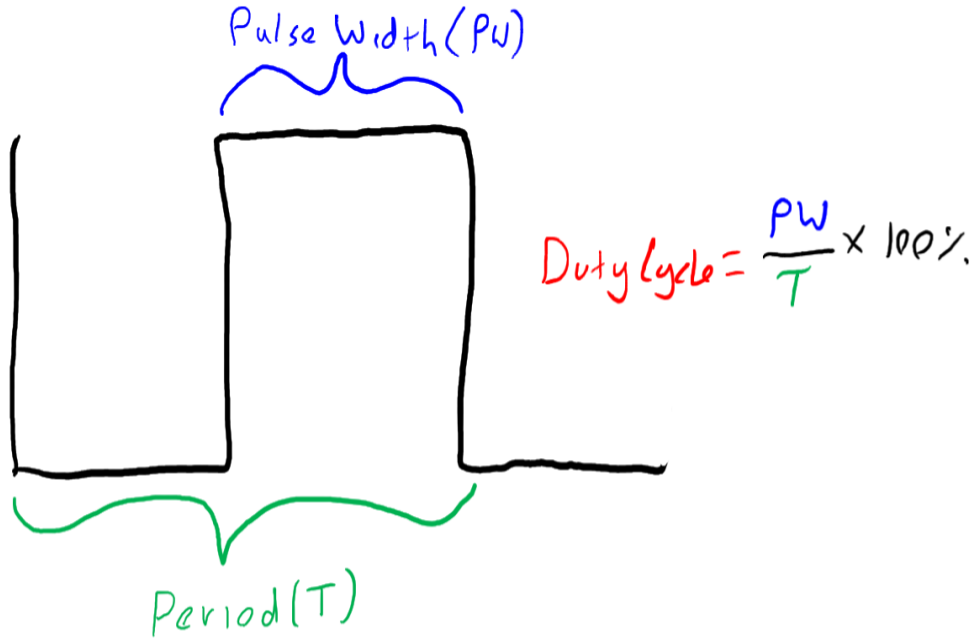


Figure 3-9: PWM Diagram

Further considerations should be assessed when deciding on a servo controller. The servo controller used in this rover contains an internal clock. The propulsion system can be given a command and left to manage its execution until a change is made. Without a clock the processing unit would need to send continuous commands to manage movement. The constant instruction is wasteful and can be a drain on computing power and battery life. Outsourcing some of the functionality allows the design to be more efficient. The servo driver utilizes I2C based communication. Most of the modules in this design rely on this system and it is efficient to have consistency throughout the design.

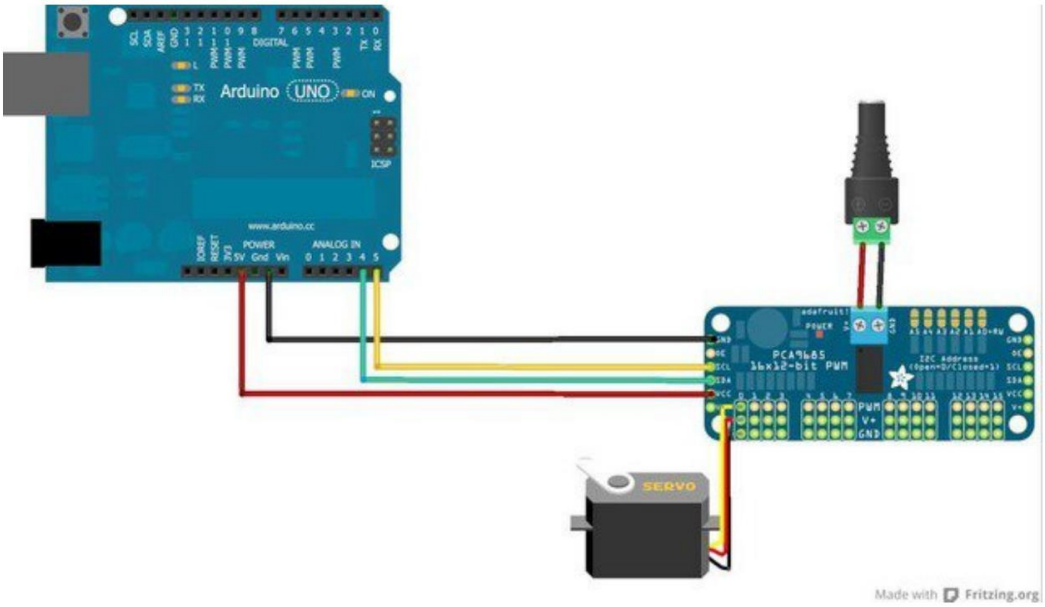


Figure 3-10: Servo Driver Diagram [16] (Permission Granted)

In light of the design advantages of a I2C capable servo driver, it is prudent to include a servo controlling chip with this capability. The Adafruit PCA9685 contained all the desired functionalities, including an internal clock to reduce the computing power required for propulsion. A drawback of the PCA9685 is that the chip is designed to power up to sixteen different servos. It is arguably inefficient to use a component with wasted capabilities. In this instance the criticism would not be accurate. Though not all of the output pins are utilized, all of the features of the PCA9685 are applied to the design and make the robot more functional. The I2C communication and internal clock of this servo driver enable the robot to operate more successfully and to achieve the desired design outcomes.

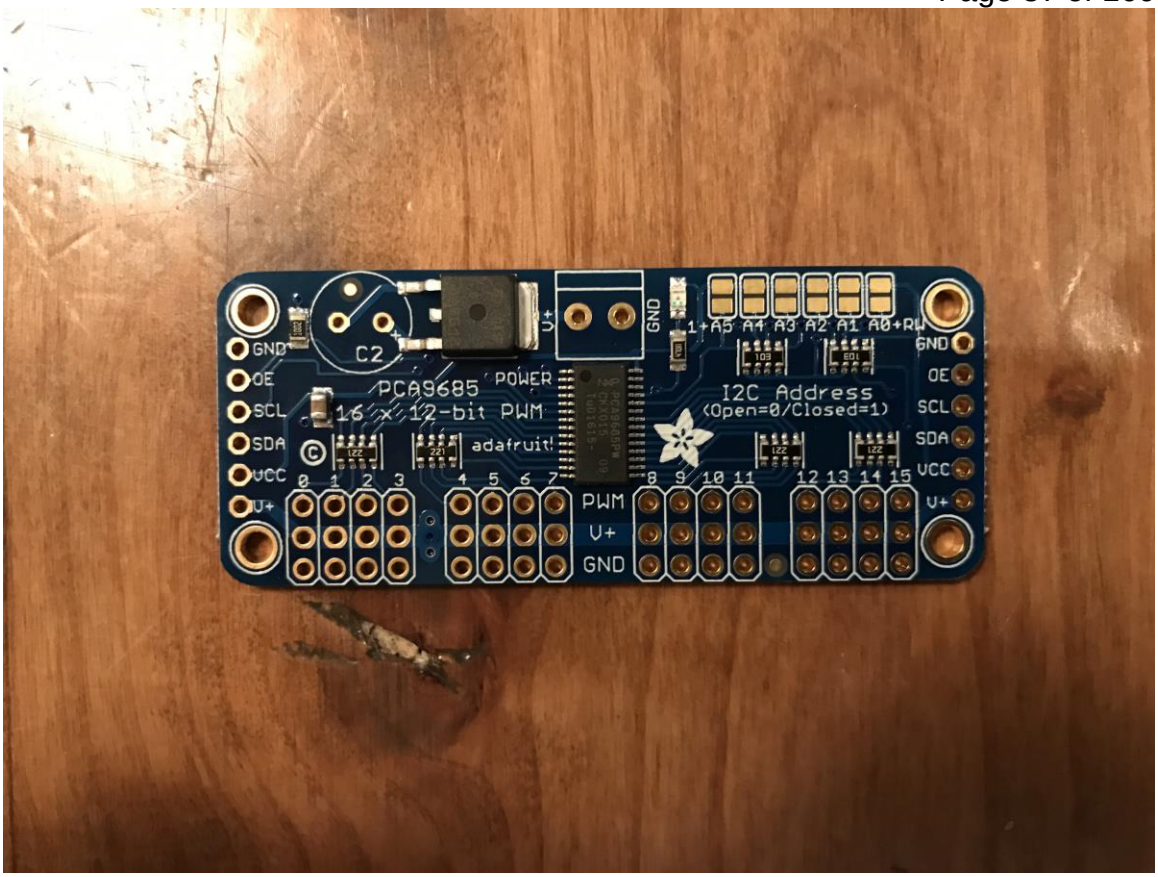


Figure 3-11: PCA9685 Servo Driver

The electric motors in this design have an operating voltage that ranges from 3V to 12V. a wide operating range allows for flexibility with regard to component choices. Most electric motors for small robot design are intended to operate in the 1.5V to 3V range. This is consistent with the voltage range of many microcontrollers. The proposed design uses a raspberry pi based system which operates at 5V. It was expedient to choose a motor that matched the characteristics of other components in the design. Importantly the higher voltage motors were also more efficient. The 1.5V to 3V options consumed 150mA while the higher voltage motors had a free run current at 75mA. With any battery-operated device power consumption is an important concern and this was a significant factor in determining the choice of motor.

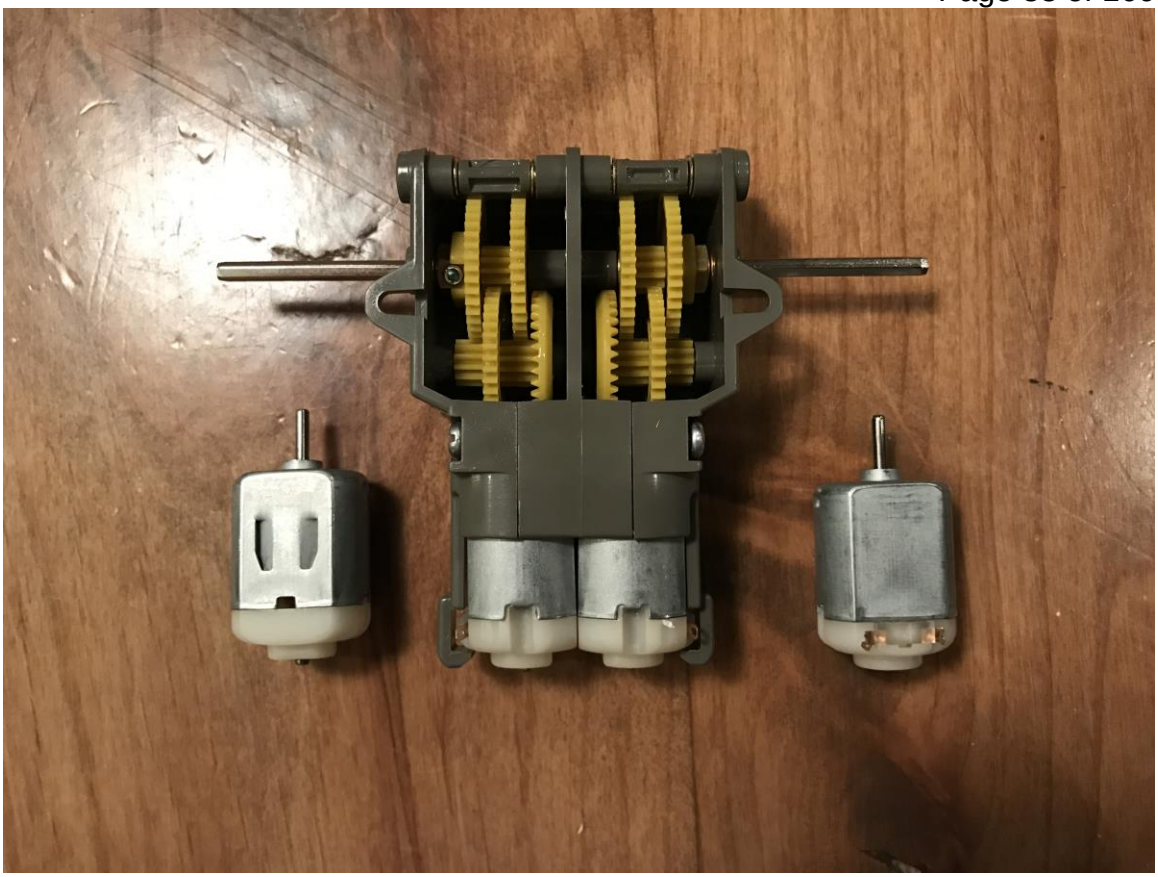


Figure 3-12: Electric Motors and Gear Assembly

3.3.7 Optical Rotary Encoder

Rotary encoders measure the rotation of a shaft, either digitally or through an analog signal. The majority of encoders are either magnetic or optical. Optical encoders are cheap and rely on a photo sensor which measures visual changes on a disk rotating around the axis being measured. The principal drawback of this technology is a susceptibility to conditions that obscure visibility, such as dust. Magnetic encoders are robust, but they are more expensive than optical encoders. The proposed design is intended to work indoors. It is unlikely that dust or smoke would be a significant issue under these operating conditions. An optical sensor is sufficient for the proposed design and it would be superfluous to include a magnetic encoder. For these reasons an optical rotary encoder is the ideal choice for this design.

Encoders can be either absolute or incremental. The absolute rotary encoder measures the specific orientation of the shaft. The angle of rotation is determined by a series of concentric rings which each correspond to a binary value. All bits set to zero might represent the smallest positive angle and the binary number would count towards the largest measured angle. The caveat to this precision is

that each ring requires a separate sensor and increases the design complexity. To justify the increased intricacy, there must be a palpable benefit to the angle information. In the current design the rotary encoder is used to determine information about an axle powering a vehicle. Wheels and tracks, the two proposed systems of propulsion, are uniform along all points of rotation. Both designs are not going to be significantly different at any orientation so there is little justification for recording angular information.

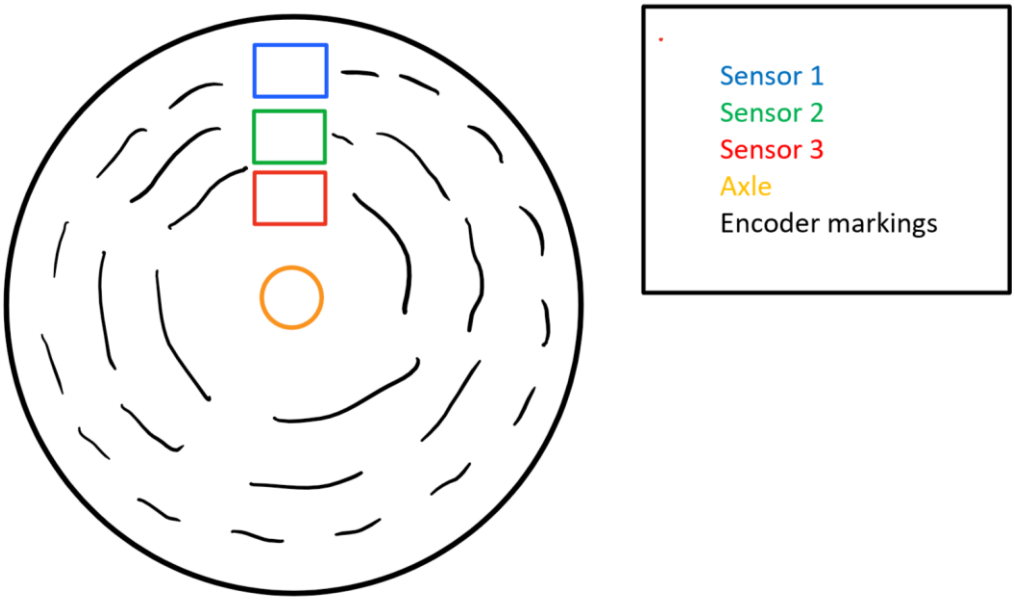


Figure 3-13: Absolute Rotary Encoder Diagram

An incremental encoder measures changes in shaft position without recording the exact location relative to a known value. A general design involves a sensor which detects changes which are evenly spaced around the axle. From the evenly spaced indicators the amount the shaft has rotated can be calculated. Since each increment is identical, it is impossible to know the precise angle of rotation. The benefit of an incremental encoder is that it requires one sensor which is considerably simpler than an absolute encoder. In this design, the rotary encoder is utilized to determine distances traveled. Each rotation can be interpreted as propelling the rover a set distance which should be consistent. The pertinent information is the number of rotations, not the absolute orientation of the axle.

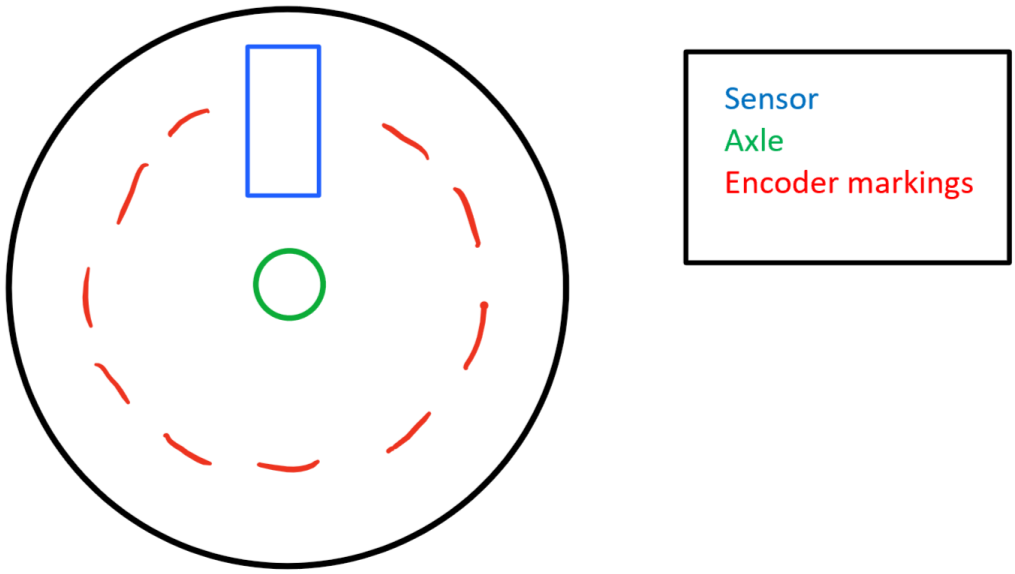


Figure 3-14: Incremental Rotary Encoder

Incremental rotary encoders can be further divided into standard or quadrature devices. The standard encoder records only changes in position with one sensor reading the increments around the axle. A quadrature encoder uses two staggered sensors that are ninety degrees out of phase. Depending on which sensor triggers first the direction of rotation can be registered. As the design relies on the encoder data to determine distances, it is plausible that the quadrature encoder is justified. Such devices would allow the rover to use reverse motion and record the data. A standard incremental encoder would not be able to distinguish between the two drive modes and would categorize both as forward motion.

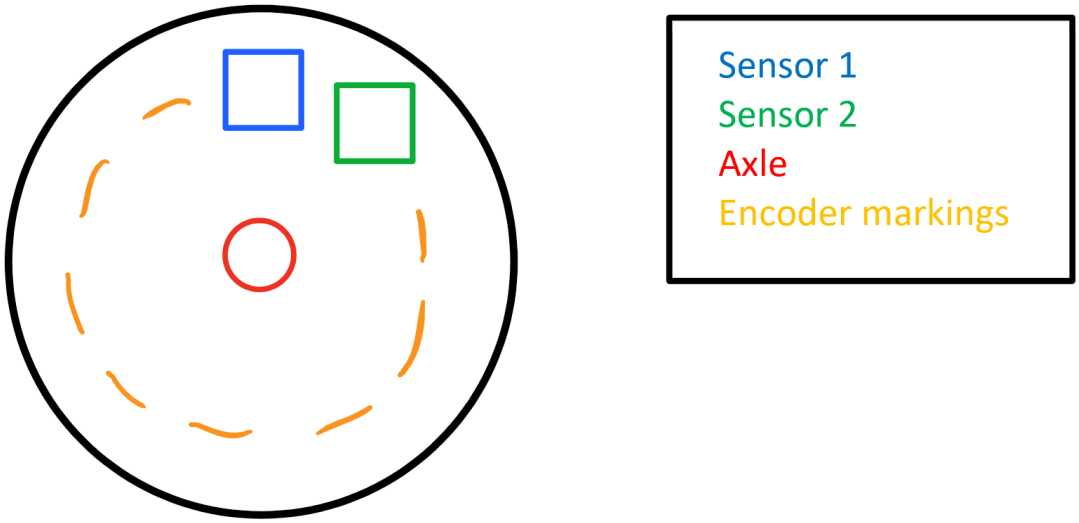


Figure 3-15: Quadrature Rotary Encoder Diagram

Though the extra data is potentially useful it is likely redundant. The rover is providing the propulsion directly through its engines and both the engine and encoder data are available. If the engine is operating in advancing mode, the encoder information should be interpreted as forward movement. If the engine is providing reverse motion the encoder data should be analyzed to indicate retreating distance. As this system accomplishes the same objectives as the more intricate quadrature sensor with a streamlined design, a standard incremental optical rotary encoder should be used.

The most prudent option was the Robot Shop RB-Rbo-122 encoder pair. These are a pair of low cost optical, incremental rotary encoders. The RB-Rbo-122 encoders contain all of the functionality required to implement this design, without excessive inclusions that would overcomplicate the robot. The encoders contain sixteen counts per rotation and it is not a quadrature design. Though not the most precise encoders available, they allow all of the precision required to implement the proposed rover.

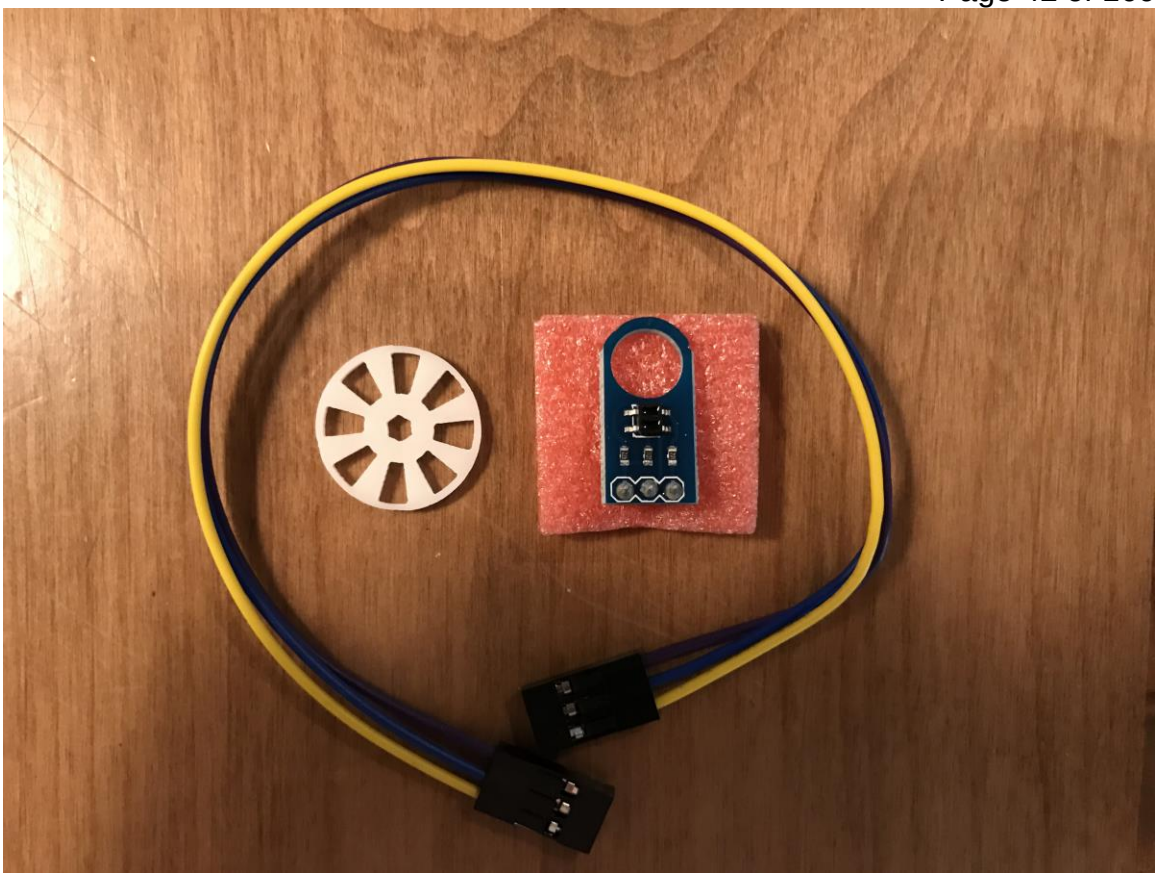


Figure 3-16: Optical Rotary Encoder

While the RB-Rbo-122 encoder is an attractive option for this design, it has a significant drawback. The output from this chip is analog and there are not any analog input pins on the Raspberry Pi. Though analog output without any available input is the primary drawback, it is not a significant obstacle. The inclusion of an analog to digital converter is an easily manageable addition to the current design. The Adafruit MCP3008 is an eight channel, ten bit analog to digital converter and enables the two optical encoders to be used with a Raspberry Pi. In our alternate design plan the analog to digital conversion would be

3.3.8 Battery Power

There are two main components to consider when determining the battery requirements of the device. The first if the Servos and the Second is the Raspberry Pi. The Raspberry pi will be capable of powering the Sensor Arrays and peripherals through the GPIO Pins. Since we have a small number of peripherals we won't be at risk for drawing too much current through the GPIO pins. The raspberry Pi requires a 5-volt power source and can draw up to 2.5 amps of power. Most lightweight batteries that have significant storage capacities won't be able to meet that specification. However, since we do not intend to use

the USB peripherals we won't be drawing that picture. For compact simplicity sake we will utilize a mobile power pack. We will start off with a device around this specification.

Figure 3-17: ROMOSS Mobile Battery [3] (Permission Granted)

The above battery is a typical mobile battery that can be used to satisfy our requirements. It offers two USB ports, both a full USB and a Micro USB port. IT maxes at 1-amp current draw so as long as we don't task the raspberry pi to engage all 4 cores and all the components we shouldn't exceed the maximum of the current output. If needed a similar device can be used but there are slight variations in size and weight as outlined in the sable below.

	Battery Option 1	Battery Option 2
Dimensions (inches)	1.91 x 3.5 x 0.8	5.4 x 2.3 x 0.8
Weight (grams)	126.7	289
Max Current (amps)	1	2
Size (mAh)	4000	10000

Table 3-2: Battery Comparison

As you can see the second option will be significantly longer, with a slightly smaller width and same height. The dimension's difference will have a marginal effect on the device as the chassis is quite long as well.

The second device to consider powering are the servos. The Chassis mount

comes with a built in placement for AA batteries, depending on the final construction the servos can be powered separately through the AA enclosure or through the Raspberry Pi itself.

3.3.9 Servos and Gearbox

The servo selection was limited in that the servos needed to function in the tracked based chassis. We would need a motor for each track as the chassis operates each track separately. This allows for precise turning and is superior in pivoting over other concepts. Since we do not require the device to reach high speeds the overall speed limitation of this device can be negated.

When deciding the servos for the vehicle we had to major selections. We could go with a standard DC motor and no gearbox, or a DC motor with a gearbox. In either configuration we would need to ensure that the device can fit into the chassis and a rotary encoder can be installed to see the movement of the device.

The HS-322 pictured below can be used in the chassis design with some modifications done to the chassis. The advantage is that this a standard design and has good usage and review around the motor design to ensure that this device would be useable in our application and has a high top speed.

Figure 3-18: HS-322 [4] (Permission Pending)

The Tamiya FA-130 motor shown in figure 3-19 is a general purpose that can be used in a variety of applications. The advantage to this motor is that it is less than half the weight of the HS-322. While it doesn't compete in torque in top speed it does compete in size and weight as well as a lower power consumption. Since

we will be utilizing two servos to control each tank tread we chose the smaller servo as the extra power isn't needed in our application. We do not have specific weight requirements or speed requirements of the vehicle so the servo was sufficient for the design and is depicted below.

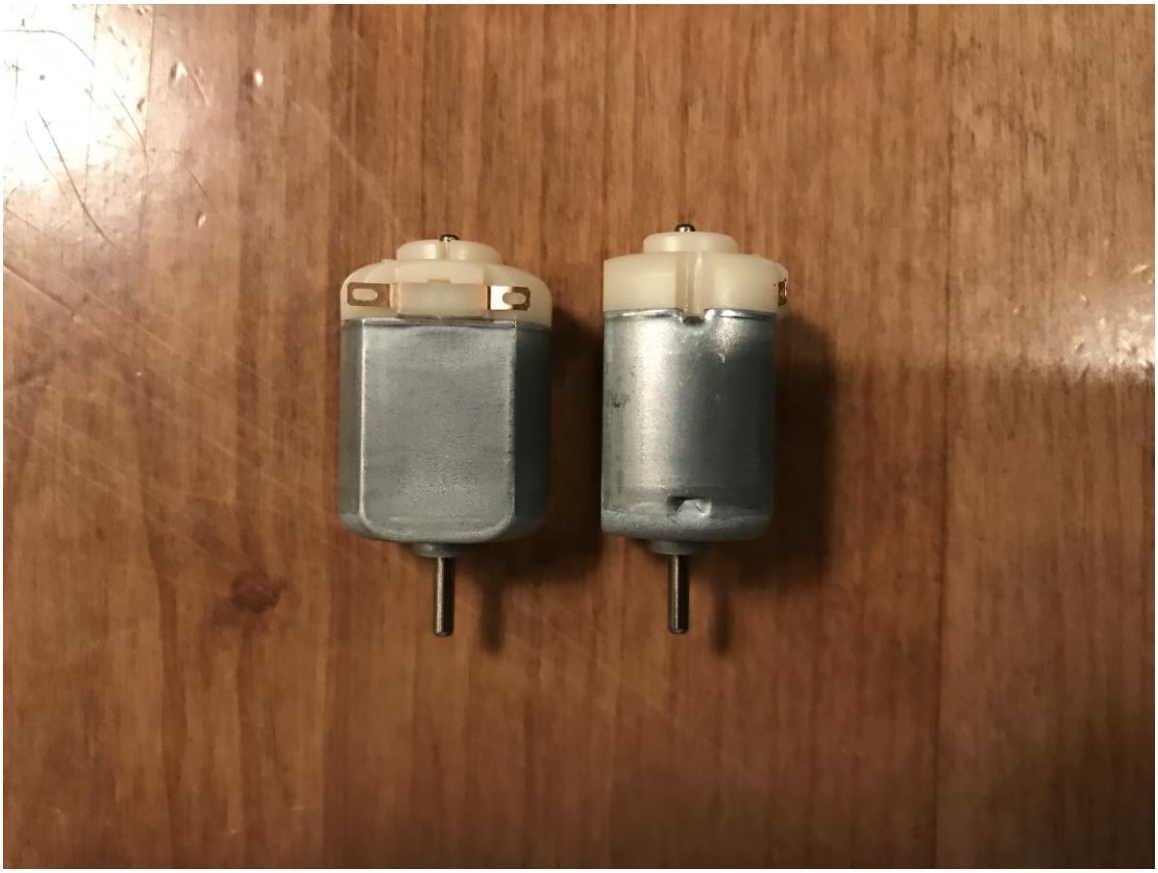


Figure 3-19: Tamiya FA-130

The lone drawback of the Tamiya FA-130 was its power consumption; it draws 150mA while running. Additionally, its operating voltage was at 3V, while the design used a 5V power source. Both of these issues are resolved by using a similar DC motor with a higher voltage and lower current. The Pololu Size 130 Brushed DC motor shown in figure 3-20 is the same size as the Tamiya model but draws only 70mA and operates anywhere from 3V to 9V. This motor is depicted below and meets all of our design specifications, though similar to the Tamiya design they are distinguished by a lighter plastic cap.

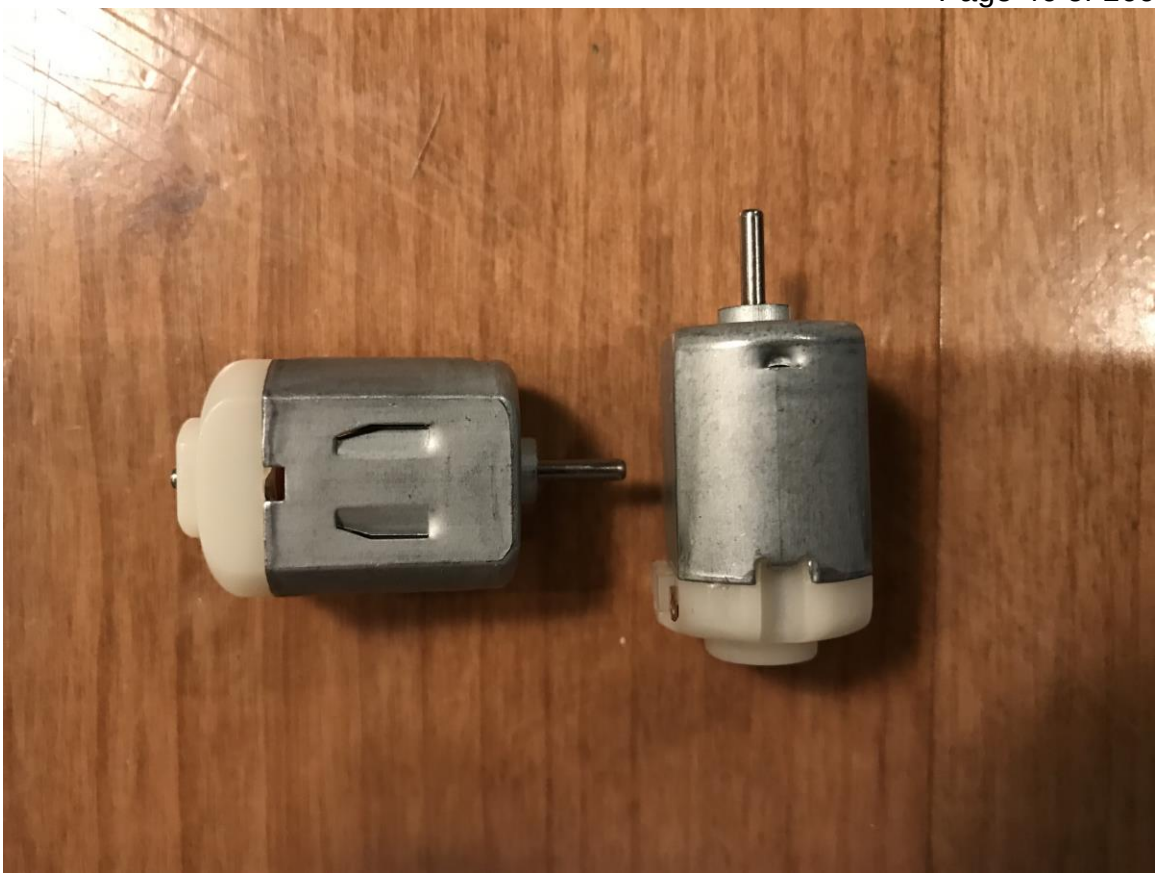


Figure 3-20: Pololu Size 130 Brushed DC motors

The device can be purchased with a gearbox with a twin motor configuration to make the connection to the tracked chassis much easier. Shown in Figure 3-21 is the twin-motor gearbox made by the manufacturer of the FA-130 motors. This is a best fit for our vehicle as it offers a small concise package to house both of the of the motors. We chose this to power the device over the larger more powerful motor.

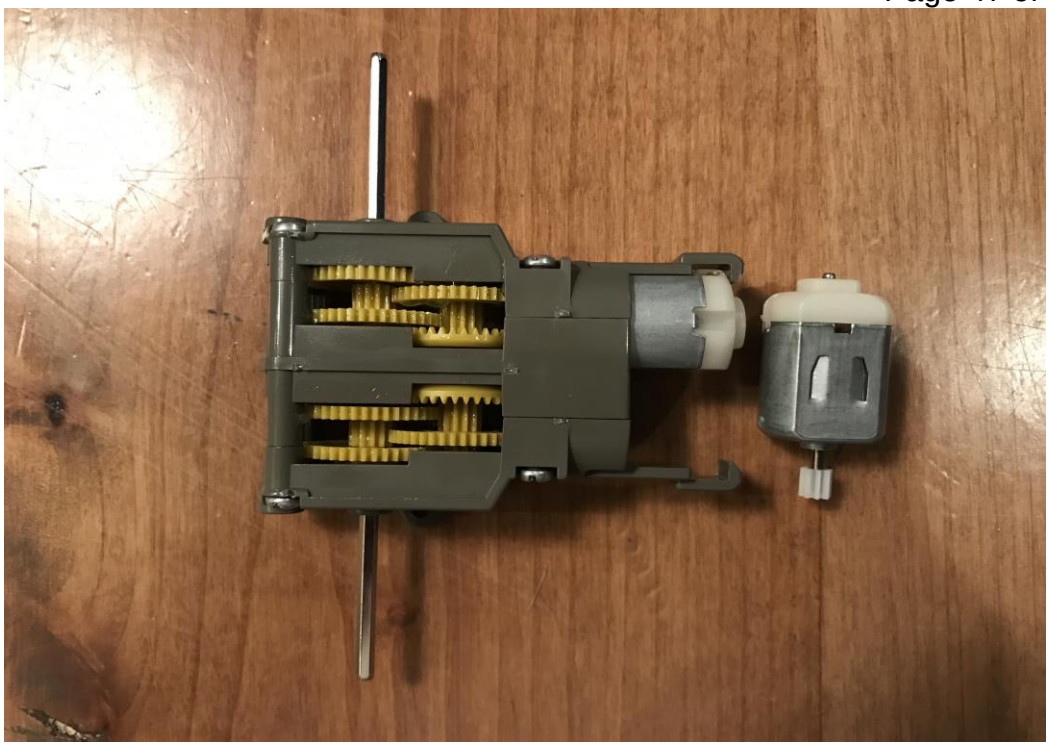


Figure 3-21: Motors with Gearbox

3.4 Parts Selection Summary

Based on the above parts and their needs we have decided on including the following components:

- 3 Ultrasonic Sensors
- 1 Optical Encoder
- 1 PCB to buffer sensors
- 1 Raspberry PI MCU
- Resistors for pull-up and voltage division
- Camera Module with extended cable
- Servo controller to translate analog signals
- Camera Servo's (Optional)
- Robot Vehicle Chassis
- Servo Motor
- Platforms and connecting wires
- Mobile battery pack

While it's possible some substitutions may need to be performed in Senior Design 2 we believe the components listed above will give us the device that functions closely to our required specifications outlined in our requirements section. If any component fails, the testing procedure a new part will be implemented with any changes in the new spec requirements of the device.

4. Related Standards and Realistic Design Constraints

4.1 Standards

Standards promote innovation by facilitating the exchange of ideas. Common terminology and testing methods allows for efficient comparisons between similar products and concepts. The current project requires both physical and coding standards. Combining both types of standards enables a comprehensive set of regulations that would be difficult without consulting both varieties of regulations. Following established standards allows for a consistent design that can be replicated by other investigators.

4.1.1 Design impact of relevant standards

Robotics standards dictate common terminology and consistent assessment methods. Safety is arguably the most important consideration when constructing an autonomous vehicle. Though the current rover is too small to be conceivably dangerous, the concept could be applied to a larger robot. In testing, the rover should be treated as having the potential for harm and this requires safety standards. The device is also a battery operated design. There are important standards regarding battery safety and the potential for fire and other adverse reactions is a serious concern. The following list should be considered as a preliminary list of robotics standards:

- Engineering Standards
    - Navigation
        - Mapping data should be defined by the IEEE Standard for robot map data representation for navigation (1873-2015).
    - Terminology
        - Robot terminology is defined by the IEEE standard ontologies for robotics and automation (1872-2015).
        - Robot motion will be described using the ISO robotics coordinate systems and motion nomenclatures (ISO 9787:2013).
    - General
        - General safety is dictated by the IEC household and similar electrical appliances – safety general requirements (IEC60335-1).

Coding standards ensure the production of consistent and high quality code. Standards enable the code to be efficiently debugged and ensures that the codes

purpose is clear to an outside observer. Intelligibility of the code is essential for a scalable product as it allows the inclusion of other developer. Facilitating an easy understanding of the code enables cooperative work in a team environment. The following list includes relevant coding standards/conventions:

- Coding Standards
    - Functionality Conventions
        - Local variables should be declared as close to their usage as possible
    - Naming Conventions
        - All variable/function names should be descriptive of their purpose
        - Member variables should be prefixed with an “m” (e.g. myVar becomes mMyVar)
    - Comments
        - Public member functions should use Javadocs style comments, with @param and @return included
        - In most cases, code should be self-commenting by using descriptive function and variable names
        - Comments should always have a single space between the comment designator and the start of the comment (e.g. //This is incorrect. // This is correct)
    - Whitespace
        - Tab characters should not be used, 4 spaces should be used instead
        - Braces should be placed on their own line
        - An additional tab level should be used after every opening brace, and removed after every closing brace
        - A single space should follow control statements (before the opening

4.2 Realistic Design Constraints

Constraints will dictate choices through the development process and will have an important influence in shaping the final design. At this early stage in the project it is difficult to have a comprehensive assessment of design constraints. While it is unlikely all constraints can be predicted, by making educated guesses obvious pitfalls can be avoided. The constraints most likely to cause potential issues have been identified and described here. Broadly, the constraints can be divided into four categories.

- Economic and time constraints.
- Environmental, social and political constraints.
- Ethical, health and safety constraints.
- Manufacturability and sustainability constraints.

4.2.1 Economic and Time constraints

The current project is being developed by a team of students and this is reflected in the economic and time constraints. In a corporate environment, it is unlikely either of these constraints would be so stringent. The total design time is dictated by the university and the project is to be completed over a two-semester period. The eight months available for this project includes the development of the idea, documentation of the design and construction of a finished product.

The economic constraints reflect the limited financial means of college age students. No sponsorship was forthcoming and this project will be self-funding. A proposed target has been a budget of no more than $350 for the construction of the robotic vehicle. This is a plausible figure but it is possible that during construction and testing it is revealed that this is not a realistic constraint.

An affordable design allows the project to achieve its intended goals. A military tracking robot is likely following an unwilling target and would be met with hostility if discovered. In this event, the robot would be most likely destroyed. If the rover is prohibitively expensive, consumers may decide that the product is not cost effective.

4.2.2 Environmental, Social, and Political constraints

The design is on a small scale and the environmental impact is likely minimal. Noise pollution is not a plausible concern for a robot propelled by small electric motors. Additionally, the robot is intended to follow an unwitting target and it would be counterproductive for the design to be noisy. If the issue of noise pollution was plausible, the robot would not be able to achieve its primary purpose.

The design is made of inert plastic and aluminum. It is unlikely that this constitutes a source of water or land based pollution. The design is not intended for manufacture on a scale that would enable the volume of plastic to be a significant environmental issue.

The proposed project is a battery powered robot and the battery source will be the largest environmental issue associated with the design. Current battery technologies contain materials that are environmentally sensitive. Most batteries contain heavy metals that and these can have a negative impact if not disposed of properly. California considers all batteries to be hazardous material and they cannot be included in regular trash. It seems reasonable to limit the project to reusable battery technologies.

In addition to environmental concerns, there is a steady decline in disposable

battery use and a continuing increase in rechargeable battery capacity. Since smaller, more efficient batteries reduce weight, it further reduces the need for battery power. For these reasons the project should be constrained to rechargeable battery technologies.

A primary social constraint, when automating an activity, is the potential impact on employment. The current climate has seen growing resistance to policies that impact blue collar workers. This group is currently the most vulnerable to automation related job loss. Due to an increasing reliance on robotic manufacturing, employment in this sector has decreased by five million jobs since the year 2000, while maintaining constant output. Though it is important to be aware of potential concerns, the surveillance robot is not a serious threat to employment. The design is intended to add a new capability, rather than automate an existing one. This robot will aid soldiers in their duties, instead of replacing them.

There are important political and social constraints to consider as part of an autonomous robot design. The constraints are compounded by the potential for political resistance to a device that tracks individuals without human input. Most of the political resistance centered around weaponized autonomous robots, but it is conceivable that the opposition could extend to tracking robots. An open letter signed by one thousand individuals involved in various tech fields, including Elon musk and Stephen Hawking, has called for a ban on autonomous military robots. The primary concern raised in the letter was the danger posed by robots that could autonomously target and engage humans. Specifically, the letter details concern with a platform that has the potential to keep humans out of the decision process.

Though the proposed robot is intended to track a human, it is limited to following a target and is not weaponized. The proposed project has no offensive capability but, it is important to be cognizant of potential concerns. A reasonable constraint, that should hopefully ease potential concerns, is to have human override for all robotic actions. Though the app there will be an ability to stop or override the autonomous activity. A human veto on autonomous decisions should help to address fears of renegade robots.

4.2.3 Ethical, Health, and Safety constraints

Ethical concerns for this design are related to privacy concerns that are inherent in designing a robot that is intended to follow an individual. When working properly the robot will be operating without consent of the target. In this instance the use of a ground based robot helps to resolve this concern. If the robot had been an airborne platform, the robot would be able to more invasively track a target. The current design can only follow a target along relatively flat terrain and

is limited, effectively, to public places. A quadcopter robot would be able to track a target, for instance, into a walled backyard. This could constitute an invasion of privacy, while tracking in a public place would not. When there is no reasonable expectation of privacy there is a diminished ethical problem with tracking a target.

The current robot is a proof of concept, intended to demonstrate the capability to follow a specific human target. Potentially, it could be turned into a toy or game, for domestic consumption. The design, for either the proof of concept or toy variant, needs to be aware of the potential privacy concerns, outlined in the paragraph above. If the robot, following a successful proof of concept, is turned into a functioning military tracking version, it is less restricted by privacy issues. Informed consent is not a part of combat interactions, in the future design would be free to use more invasive tracking methods. In the current proof of concept version, the robot should be designed with personal privacy in mind. To this end the robot is designed to operate indoors in relatively controlled conditions and it is not plausible that it could be used for invasive activities.

There are minimal health and safety concerns for this design. The robot is going to weigh roughly one to two kilograms and it is unlikely that a collision could present a risk to personal safety. The robot is composed primarily of inert plastic and metal so it does not present a risk of toxicity above any other type of household electronics. As a check against any unintended personal risk posed by the robot acting autonomously, a human override will be included in the associated phone application.

4.2.4 Manufacturability and Sustainability constraints

The manufacturability of the robot is not particularly challenging for this design. The robot, in broad strokes, is similar to other commercial robots that are successfully manufactured for domestic use. The tolerances required for the intended implementation are reasonable and within current manufacturing capabilities. The design uses widely available materials and technologies and it is unlikely that manufacturing this robot will be excessively difficult.

Sustainability of this project is an important concern, if the design is going to be realized. If the business plan is not realistic, the robot will not be built. The longer goal of this design is a military application, but in the meantime, it could function as a toy. Military robots are required to operate under rigorous conditions and have low tolerances for failure. In contrast, a domestic toy is required to operate in a narrow window of temperatures under ideal circumstances. While it may be a reasonable criticism that a military robot breaks down in a sandy environment when temperatures reach over 100 degrees, this is not a reasonable critique of a household toy. Targeting the underlying technology at two markets, first as a toy and then as a military robot,

makes the business plan more robust. After the proof of concept there is a potential source of revenue as a toy while the more involved challenges of implementing a military robot are addressed.

5 Project Hardware Design Details

5.1 Research of initial project design

5.1.1 Interface for Microcontroller

The interface for the microcontroller from the hardware aspect will consist of various connectors to interface with the physical components as well as the PC interface. The main user interface will be via USB for programming and debugging. This will allow the programming of the device and the testing of subsystem designs. The raspberry pi 3 utilizes a standard USB hubs and an HDMI port for a Mouse, Keyboard and Monitor as well as a micro SD card to program the Operating System. Once the device is configured, calibrated and tested the user will primarily interact with the device via Wi-Fi through a phone application that will be linked to the Raspberry Pi. The microcontroller interface diagram is shown in figure 5-1.

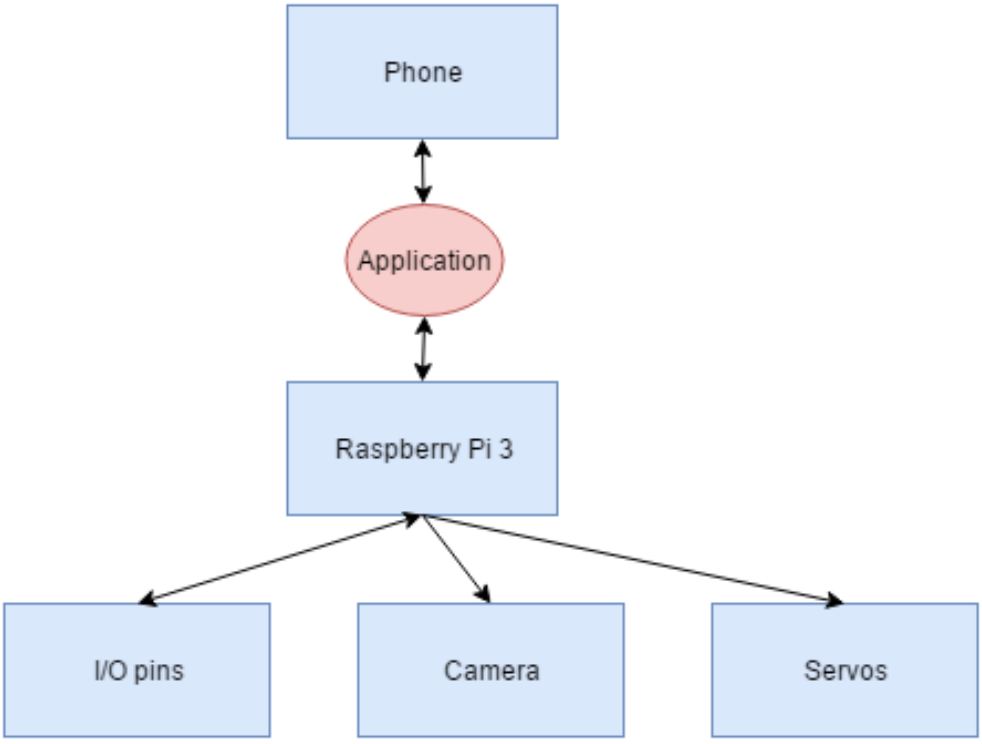


Figure 5-1: Interface Functional Diagram

5.1.2 GPIO

The General Purpose Input Output lines operate on a 3.3v maximum for the Raspberry PI. These are used to interface with the various components and sensors on the device. These will be used in a LOW and HIGH based communication corresponding to typical 0 and 1 bit based logic. If the voltage input to the device is below designed threshold the port will be considered to be LOW. If low the software will read it as a 0. If the pin is above the threshold it will be read as 1 or HIGH. Ideally we want our GPIO lines to either be reading 0 volts or close to it and 3.3 volts or close to it. We can accomplish this with voltage dividers in order to deal with any of the peripherals that may output higher than 3.3 volts.

5.1.3 i2c Driver system

When looking into the i2c subsystem it's important to consider how the i2c communication protocol works. As depicted in figure 5-2, the most recent Rev 6 protocol uses two lines Serial Data (SDA) and Serial Clock (SCL) [1]. The protocol I designed for general purpose circuits with LCD's, servos, I/O ports, EEPROM and A/D converters as well as other sensor inputs. The ultra-fast-mode is capable of data transfers of up to 5 Mbit/s and is unidirectional in nature. If utilizing lower speeds, the bus can handle bidirectional data transfers. Below is an example configuration for a typical i2c bus line.

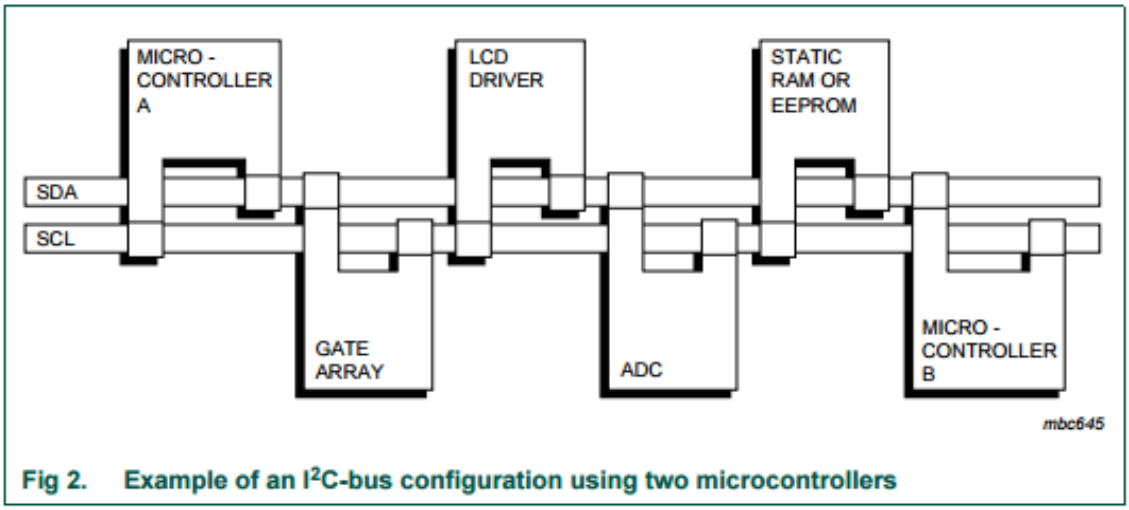


Figure 5-2: I2C Communication Diagram [10] (Permission Pending)

One of the primary reasons to use the i2c communication protocol is to reduce the number of busses traveling to the MCU. The accelerometer and magnetometer use the i2c protocol as well as the servo driver.

5.1.4 Power system across board layout

The power system across the board will be dependent on the power required by each of the sensor. The highest consumer of power will be the MCU and the Servo motor for the RSV. They position of the power supply and these components will take priority when determining the ideal placement. The other sensors will use 5 Volts or less. Based on the positioning of the sensors we will add power lines from the main power source or from the Raspberry Pi's GPIO lines. For the i2c lines we utilize a pull-up resistor to the bus line. For the ultrasonic sensors we may need to add a transistor to increase the voltage depending on the tolerances of the component. We will need to have the physical hardware to test the full functionality of the component to determine the viability and placement on the PCB board.

5.2 Raspberry Pi system

5.2.1 User interface

The Raspberry Pi can be booted using many different operating systems, including a Linux OS, RISC OS, and Windows 10 IOT OS. This allows for a programmer to pick an environment that they are comfortable with. The advantage of this flexibility of having a full computer is that the programmer can pick whatever is easiest for the application. Any language can be selected as long as the proper libraries can be obtained to interface with the PI. Using these libraries and the addresses of the GPIO pins and i2c connections, each component can be monitored and processed. The software will then take over and execute the proper instructions to complete the task. Depending on the status of the various sensors, the software will cause the PI to generate an output to the connected devices such as moving the servos in a specific direction. The code example below shows how the GPIO pins can be accessed in C using the Raspberry Pi.

```
	wiringPiSetupGpio()
pinMode(17,                                                    INPUT);
pinMode(23,                                                   OUTPUT);
pinMode(18, PWM_OUTPUT);
	digitalWrite(23, HIGH);
```

Each pin can simply be assessed by its number and set to whatever I/O setting is desired. These pins can be manipulated just as any other variable. All other parts of the software can be done just as any other program, using the GPIO pins as variable for calculations or input/outputs

5.3 Sensor system for collision avoidance and relative positioning

The goal of the main system is to interface effectively and efficiently with all of the subsystems so that the vehicle works in a manner that is capable of completing any designed directives of the software components. When looking at the layout the system needs to take into some major and minor components.

The first major component to consider is the chassis of the RSV. The chassis typically includes the wheels/treads to allow the vehicle to move. It often doesn't include any of the servos which will allow the user to customize the components to tailor the system. The design we have chosen is to use tank based treads to allow the vehicle to pivot at the center of the vehicle. This will allow for more precise movements and easier control for the MCU.

The next major component is the MCU. The MCU will have a significant footprint in comparison to the other sensors, servos and optical components. Understanding this we may choose to build a separate compartment to house the MCU to keep it elevated away from the other components as many of the components will need to be connected directly to the PCB.

5.3.1 PWM for sensor detection for controller interface

The sensors use the i2c communication protocol which allows all of the components to be connected in parallel which respective lines and pull-up resistors. Some of the components will not use this and will need direct access to the GPIO lines and voltage dividers for PWM. The servos typically communicate with either PWM or Analog. With all of that in mind we need to carefully consider the PCB size and placement relative to the size of the chassis. Fortunately, we only have the Ultrasonic sensors that require dedicated GPIO ports. While the other components have some flexibility in placement.

The PCB board will interface primarily with the i2c sensors and secondarily with any sensors that need some sort of signal modulation, or voltage difference. The ultrasonic sensor will send a signal directly to the MCU however the voltage will need to be have reduced to meet the requirements of the General Purpose Input Output lines. Since we will be utilizing three ultrasonic sensors we will utilize jumper wires to get the correct placement.

In order to determine the layout for the PCB board we will need to look at the Chassis of the vehicle. We will be utilizing a track based wheel design. The chassis has a prebuilt spot to attach the Raspberry PI MCU. It also has a spot for the power storage. Either 4 AA's or a Lithium Polymer battery. The ability to use

the AA’s allow for a quick replacement of the components and will help in the hardware testing phase.

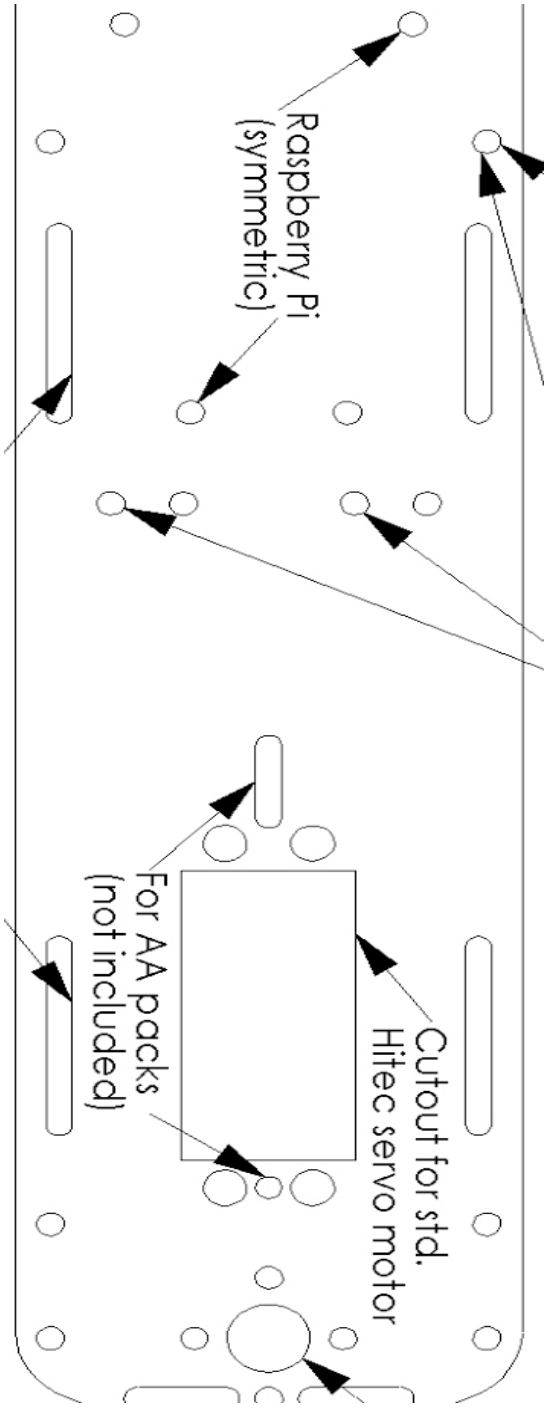

Raspberry Pi
(symmetric)
For AA packs
(not included)
Cutout for std.
Hitec servo motor

Figure 5-3: Robot Chassis Diagram [5] (Permission Granted)

Figure 5-3 shows the top layout of the chassis the left hand side has fitted mounting for the MCU while the right side has the fitted mounting for the servo mother and possible battery pack. Considering this we will need to build a platform above these components to install the PCB. We will also need a separate mount for the Camera Module so that the device can see forward.

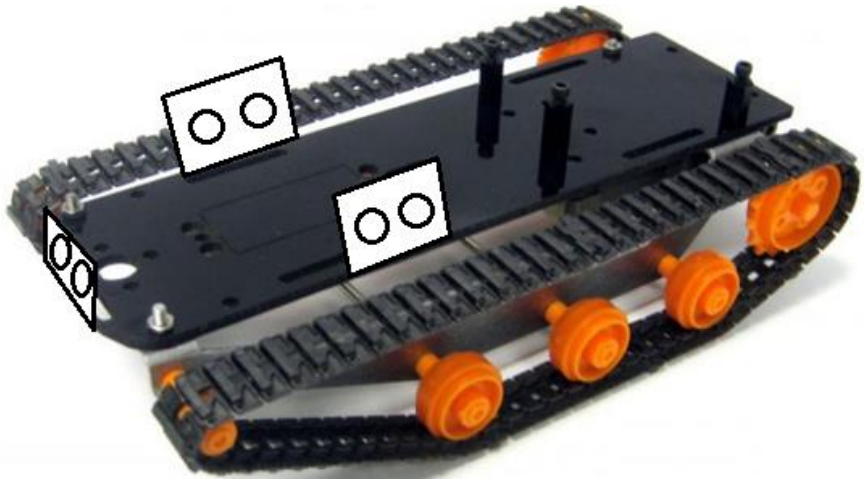


Figure 5-4: Ultrasonic Sensor Location Diagram [6] (Permission Granted)

The above figure shows the physical location needed for the ultrasonic sensors this will allow for the vehicle to get enough feedback to allow feedback in the collision avoidance system. The portion of the PCB needed for sensors is shown in figure 5-4 below. We need each of the sensors to be located physically as shown above but to connect to the MCU we will need to run the output PWM through the divider to correctly communicate with MCU. Since these components are needed for the major portion of the vehicle movement these will take priority. This will allow the vehicle to make auto corrections based off its distance from other objects.

The benefit of using i2c protocol is that in our design we are utilizing one MCU and we are expecting it to process the data for all the sensor arrays. If we utilized the various GPIO of the MCU we would have numerous jumper cables and wires running all over our circuit. It's much easier to configure the design to utilize the i2c bus and allow the MCU to call each of the components via their serial based address.

5.3.2 Sensor array

The Vehicle will utilize four primary sensors. It will utilize the Rotary Encoder, the

Ultrasonic Sensors, the Magnetometer and the Camera. The information below will focus on the layout of the sensor array as it pertains to collision avoidance. The rotary encoder will be covered more in the servo section. The sensor array will exclusively utilize the ultrasonic sensors. This will allow the vehicle to avoid any obstacles in the testing environment. Below are the schematics for the design of these sensors as well as the Magnetometer/Accelerometer. This component will not directly be involved with the collision avoidance, but it will play a vital role in determining how to move in relation to an object detected. The device will sample data points from the Magnetometer to determine what the neutral position will be. Once the device knows what the neutral position is it can then turn itself at a tangential angle to avoid the object. Sensors will be located on the sides and front of the vehicle so the device will know once it has moved past the object.

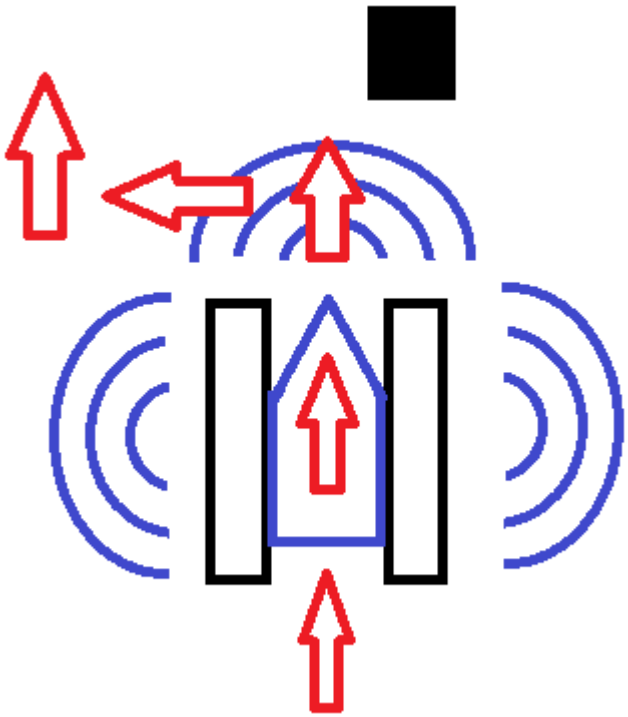


Figure 5-5: Sensor Operation

Figure 5-5 above shows how the sensor array will work during operation. As the device is moving along a path based off of its operation mode it will begin to detect an object. The vehicle will begin to slowly decelerate until it reaches a point in which it needs to move a different direction. At this point the vehicle will engage the magnetometer.

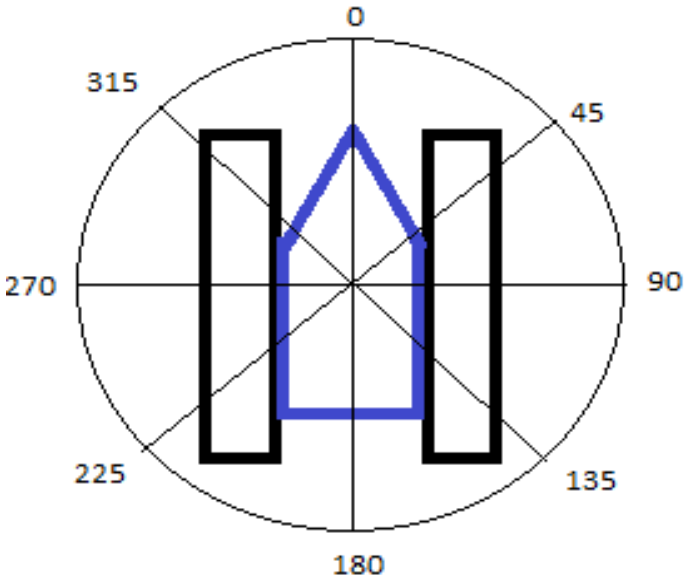


Figure 5-6: magnetometer depiction

As shown in Figure 5-6 when the device engages the magnetometer the device will make a decision on how to move away from the object. In the example with the object above the device can orient itself to the 270-degree position then move forward till the object is sufficient distance or disappears off the radar. When the device set itself to a neutral position the 0 degree because whatever the reading of that magnetometer receives in the X-Y plane. For the purposes of navigation this degree will be stored in memory so the device understands its current heading relative to its last.

Figure 5-7 shows the sensor array for the ultrasonic sensors. It is an electrical schematic so the physical location will vary. The components need a 5-volt power source to power the instrument on. Each of the components will be connected to a GPIO to send signals and receive signals. The pulse coming in will initiate a trigger sequence that is shown in the prior section about the ultrasonic sensors. The device well then initiate a sequence of high frequency pulses at around 40 KHz. The pulses will then bounce off an object and return to the device. The pulse returning to the GPIO will natively be at 5 volts. This will be an issue for most MCU as typically the required voltage should not exceed 3.3 Volts. The Raspberry PI will need a voltage divider placed in between the returning signal and the sensor.

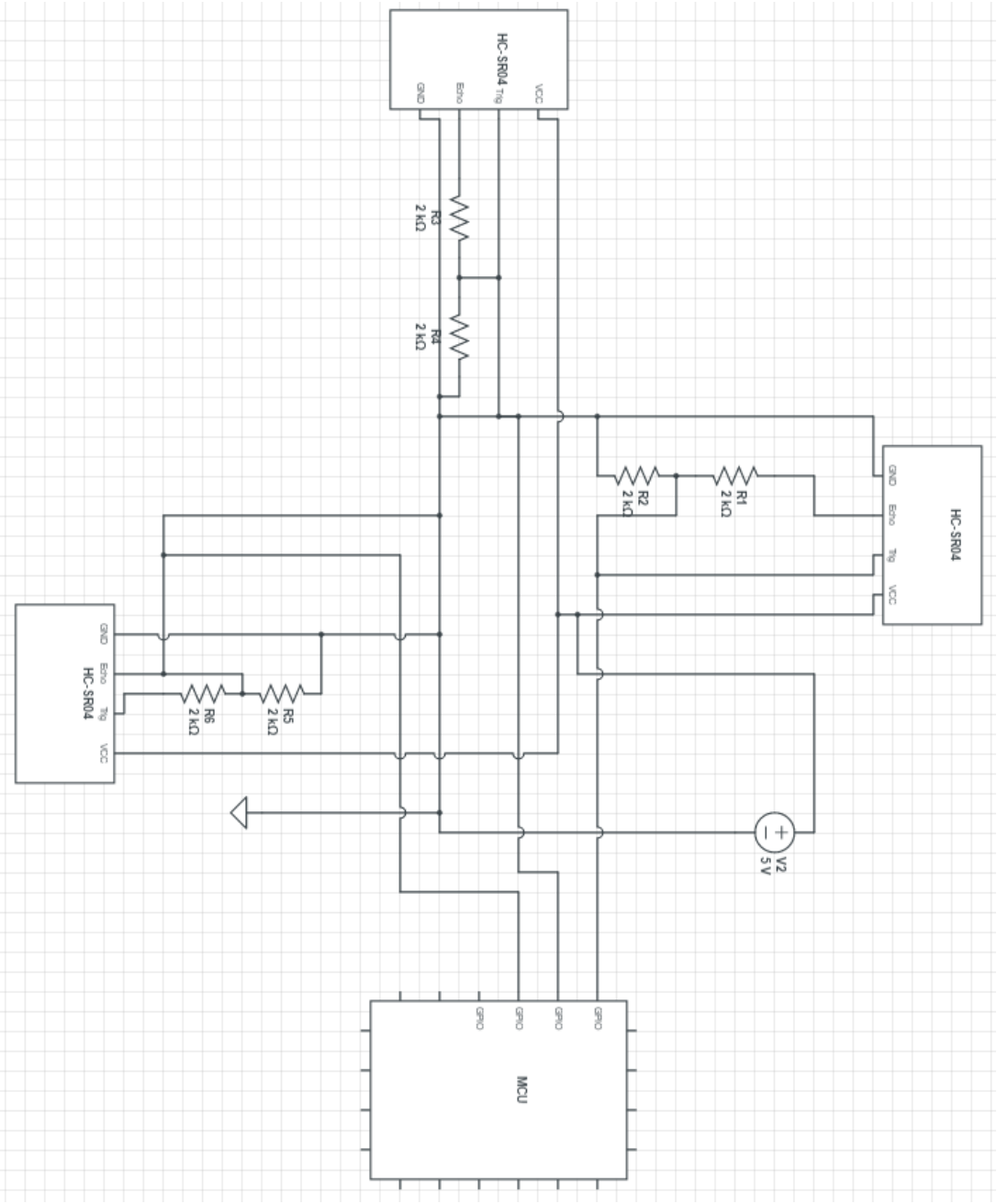


Figure 5-7: Ultrasonic Sensor Schematic Diagram

This is ideal for our project as we will want the ability to constantly send and

receive data from the sensor arrays while minimizing the foot space so the vehicle can be more compact. One of the important consideration in testing these components out is to determine if all of the devices have different serial address. Assuming they do we will be able to link them on the PCB board as well and run the output lines to the i2c communication lines on the MCU. If we feel that we have extra GPIO lines, we may separate the signals to have a non-interrupted continuous flow of readings.

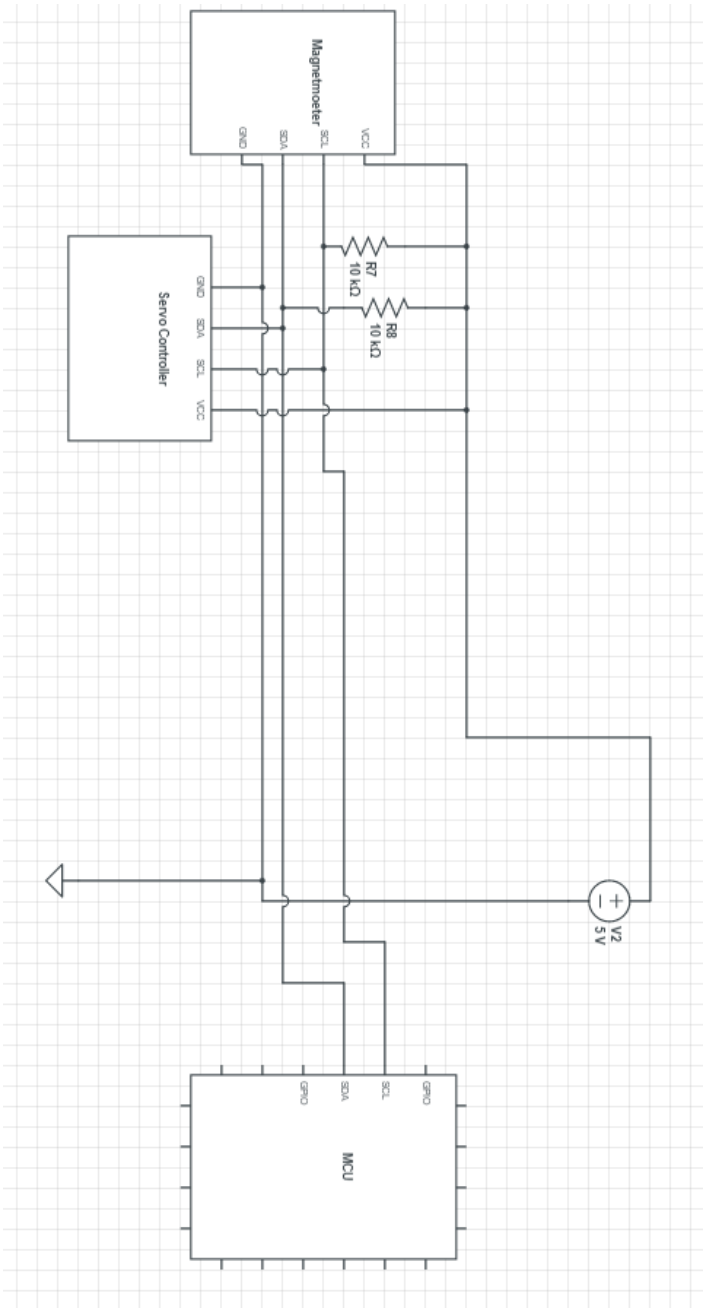

Magnetmoeter
GND
SDA
SCL
VCC
R7
10 kΩ
R8
10 kΩ
Servo Controller
GND
SDA
SCL
VCC
V2
5 V
GPIO
SDA
SCL
GPIO
MCU

Figure 5-8: Magnetometer Schematic Diagram

Figure 5-8 shows a sample of the PCB board that will recognize the i2c sensors and the interface back to the MCU. The placements of these components will vary on the physical placement of the RSV as the Magnetometer will want to be further from the electro motor to avoid any interference from magnetic fields. The Servo controller will be placed on the device physically based on the needs of the other components and will fill gaps in the area to fill out the device layout on the chassis.

5.4 Vehicle movement

The robot has a tracked chassis and is powered by dual engines, with each powering one set of the rubber treads. The motors are set in a gearbox that allows for independent articulation of each side of the vehicle. The electric brushed DC motors are a relatively simple design, often used in small robot construction. More intricate servos were considered but ultimately the extra functionality did not justify the increased complexity. Specifically, the alternate motors had the ability to rotate to specific angle rotations, rather than being able to be set on or off. As the robot is intended to follow simple movement commands, with realistic tolerances, the brushed DC motors were considered acceptable.

The motors are controlled by a servo driver with an internal clock independent of the main computer. Movement occurs by the central computer sending a command which is independently managed by the servo controller. In this system, the movement commands are sent as they change rather than the central processor continuously managing the instructions. The schematic diagram using the servo driver to control the two servo motors is shown in the following figure.

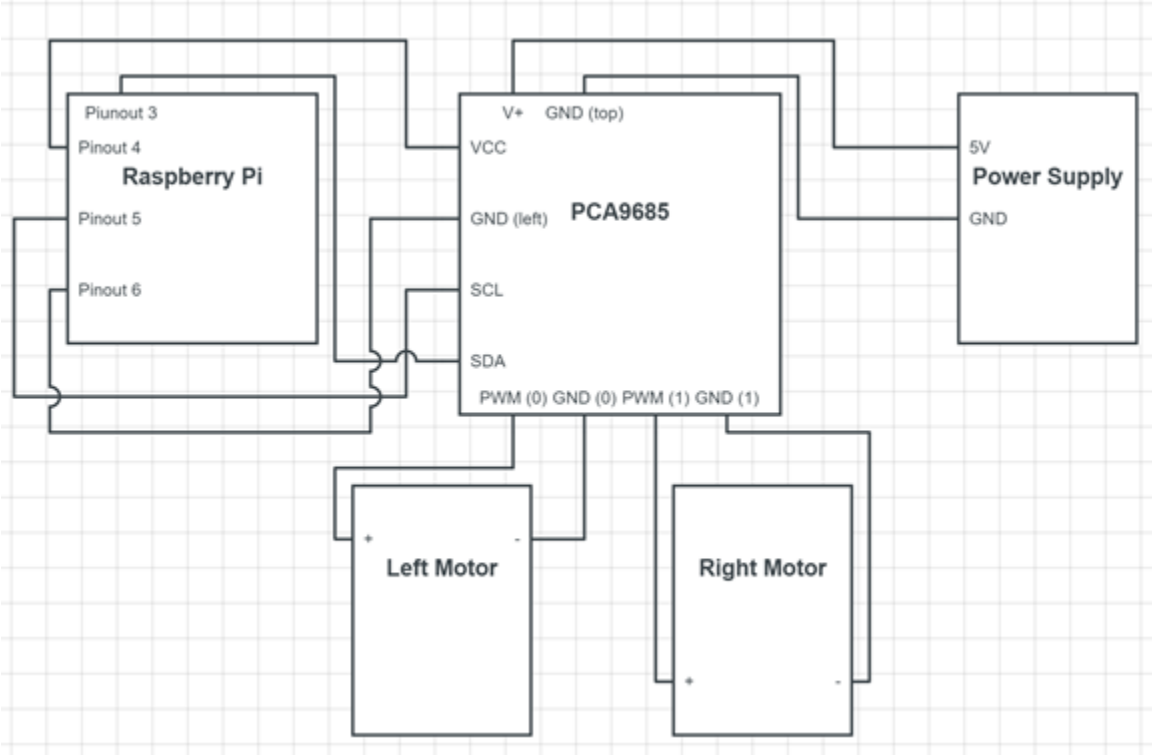


Figure 5-9: Schematic Diagram of the Servo Driver Assembly

Turning is accomplished by activating one motor in reverse and one in forward motion. Forward motion occurs when both motors are operating in forward mode. Reverse movement has been excluded from this design to simplify the robot. The differential turning allows for a change in direction without moving the robot. Since the vehicle can achieve full mobility without reverse motion, it is superfluous to include the capability. If reverse motion is to be included, it would require a rear facing ultrasonic sensor and a whole new library of commands. By omitting reverse motion, the design is simplified without sacrificing the robot's utility.

Movement data to the Raspberry Pi will be communicated by two rotary encoders, located on each axle of the robot. The encoders tell the computer how much the axles have rotated and provide 16 analog pulses per full rotation. The confounding issue is that the raspberry pi does not have any analog input pins. To solve this, issue an analog to digital converter, the Adafruit, MCP3008, will be used to convert the input into information usable by the Raspberry Pi. The converter can communicate with the Raspberry Pi in two different configurations. The first is a bit banging configuration, which can interface with any of the GPIO pins on the Raspberry Pi and the schematic diagram is displayed in figure 5-10.

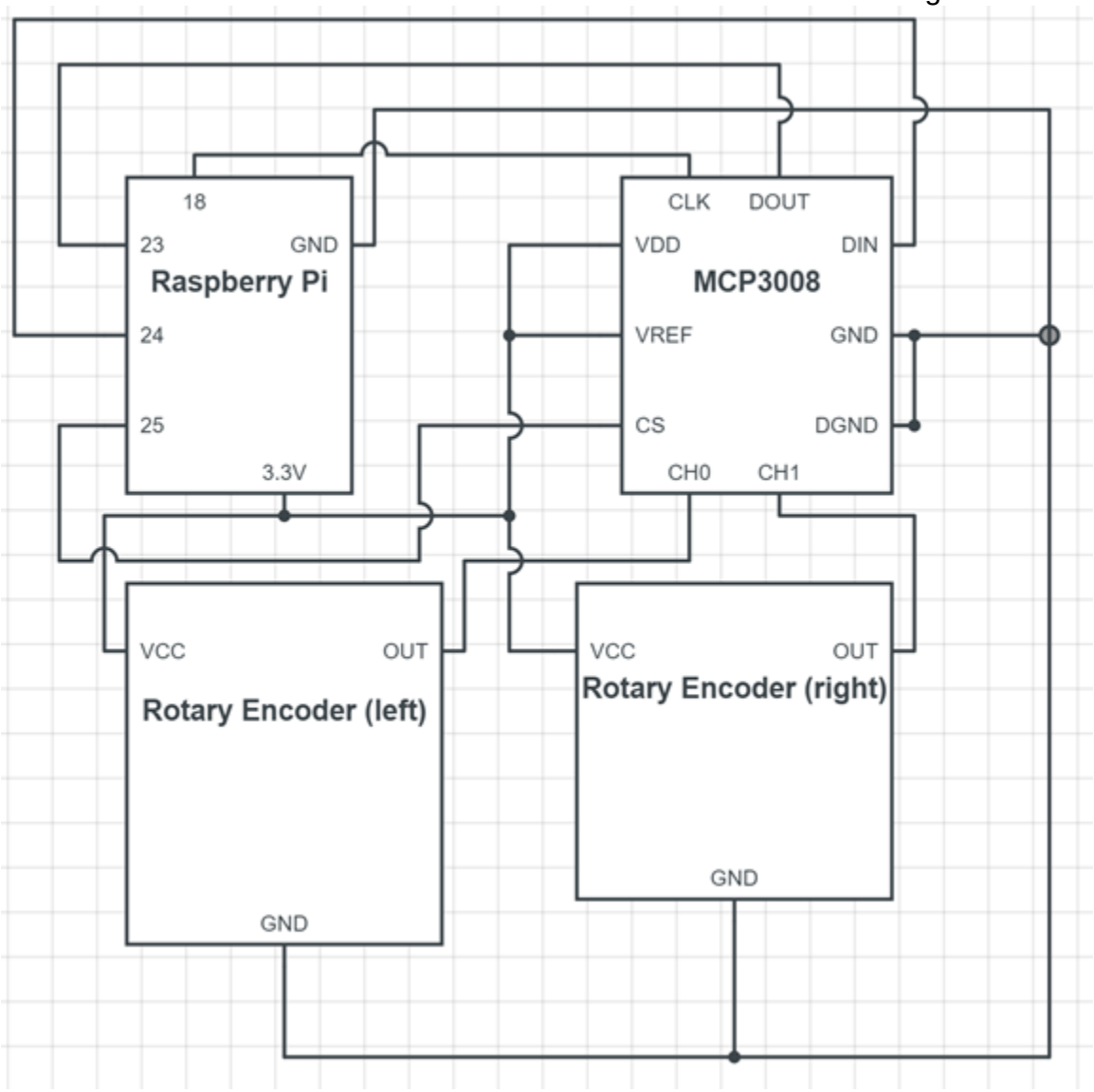


Figure 5-10: Schematic Diagram for Bit Banging Implementation

The principal drawback of this configuration, is that it communicates slowly, relative to the alternate design. The second option is a serial peripheral interface (SPI) configuration which communicates quickly with the Raspberry Pi. The drawback is that this design requires specific pins and this could be an issue if they are needed for a different peripheral. Essentially, the tradeoff is a decision between prioritizing speed and flexibility. The design is a moving vehicle and collision avoidance will be an important concern. If the sensor data is slow in reaching the Raspberry Pi, this could conceivably lead to collisions which could damage the robot. Additionally, a slow unresponsive robot would not be effective in tracking a moving target. Currently the pins for the serial peripheral system are not utilized in the design because communication speed is prioritized above

the potential flexibility. Unless the pins are needed after design changes, the SPI configuration shown in Figure 5-11 will be implemented.

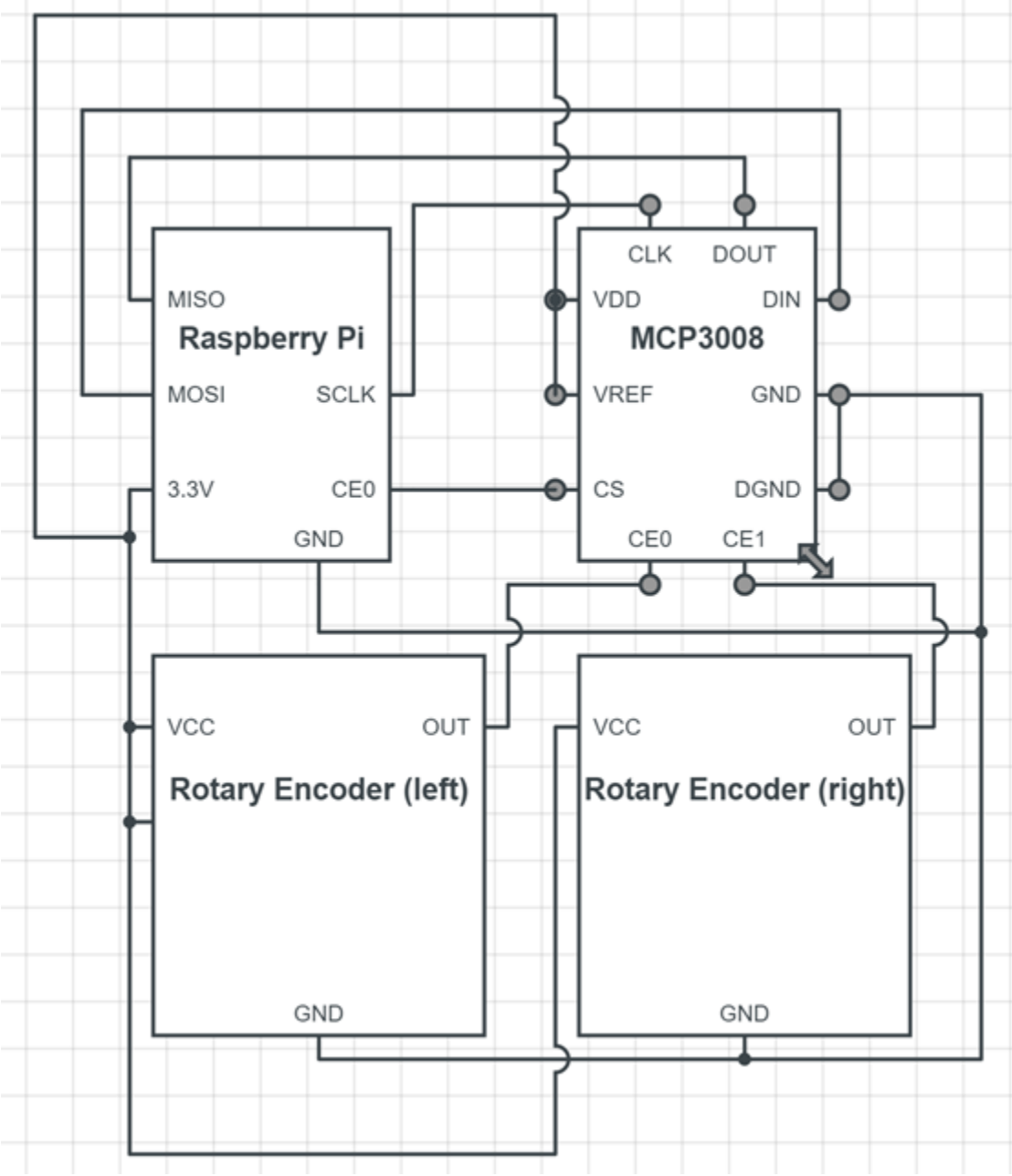


Figure 5-11: Schematic Diagram for SPI Implementation

5.4.2 Testing of Servo functionality

The motors are expected to power the robotic vehicle forwards at a maximum speed of 15 cm/second. This is a rough estimate and needs to be verified through testing. A proposed methodology would involve setting the servo to full

capacity and measuring the time required to travel one meter.

The motor can further be set at various power thresholds and the time to travel a set distance can be recorded. From this data a reliable speed database can be determined. Such a database would allow for more efficient code that would allow for the design to account for the movement parameters. A database that accounts for stop time and distance should be recorded.

The testing paradigm detailed above would also be used to calibrate the rotary encoders. Collecting the number of pulses sent from the encoders and dividing the distance traveled by this quantity would allow the distance per pulse to be determined. An accurate accounting of distance allows the robot to move precisely and ensures the equipment is functioning as intended.

The motors can be set to act as breaks and will slow down the robot in this operating mode. Understanding the distance and time required to bring the vehicle to a stop will aid in designing the collision avoidance code. A further concern is the rate of turning. Turning is likely the area with the most uncertainty, with regard to robot movement.

5.5 Summary of Design

The purpose of the device is to function autonomously. In order to do so the device will need accurate measurements from the sensor array. These measurements will help it navigate down corridors and move around objects that arise in its path. The Video feed will help the robot in the form of image detection. When the robot is in a search based mode or follow mode the robot will actively seek out a target using the camera.

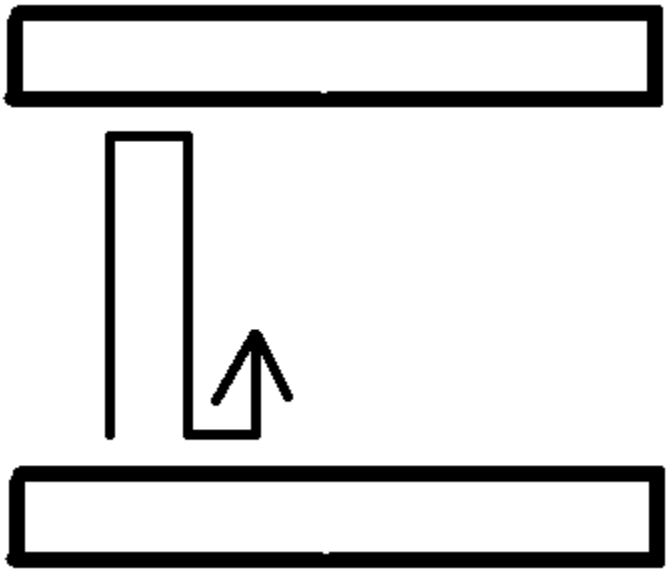


Figure 5-12: Sample navigational path of the vehicle in search mode

The device will utilize the sensor array for navigation and the magnetometer to help form a sweep based pattern. This will allow the device to understand where it has been so that will not cross over an area that it has recently explored. Ideally the device will move in a pattern as shown below. Figure 5-12 shows a sample movement path of the device. If engage the device to enter search mode and seek out a target the device will have to accurately move around its surroundings without crashing into an object. On top of that the sweeping pattern must be accurate. At each of the turning points above the device will engage the sensor array and the rotary encoder to both slow down the vehicle, determine position, turn to the desired position and move forward.

The data collected and distance traveled will be stored in the memory of the device to create a 2D map of where the robot has been. This will help the device perform a sweep pattern when actively seeking an object. With these different sensors the device will be able to perform multiple modes of operation to autonomously perform the required task. When the device has completed its required task the device will be programed to delete its cache memory or store it on the SD card space permitting. This will ensure the device has usable memory for the specific task and if a user log is needed we will be able to facilitate some of the data for user purposes.

6. Software Design Details

The Robotic Surveillance Vehicle will have many layers of software design needed for it to operate to the developed requirements. The design will cover many aspects from the firmware on our microcontroller to the mobile application used to control the Robotic Surveillance Vehicle. The levels of software and communication between the levels can be seen in Figure 6-1 below The firmware level will include the logic required to interface with all sensors embedded into the Robotic Surveillance Vehicle. These sensors will be used to meet the software level requirements such as computer vision needed to spot and track a target, and collision detections will use multiple sensors including the accelerometer, and the ultrasonic sensors.

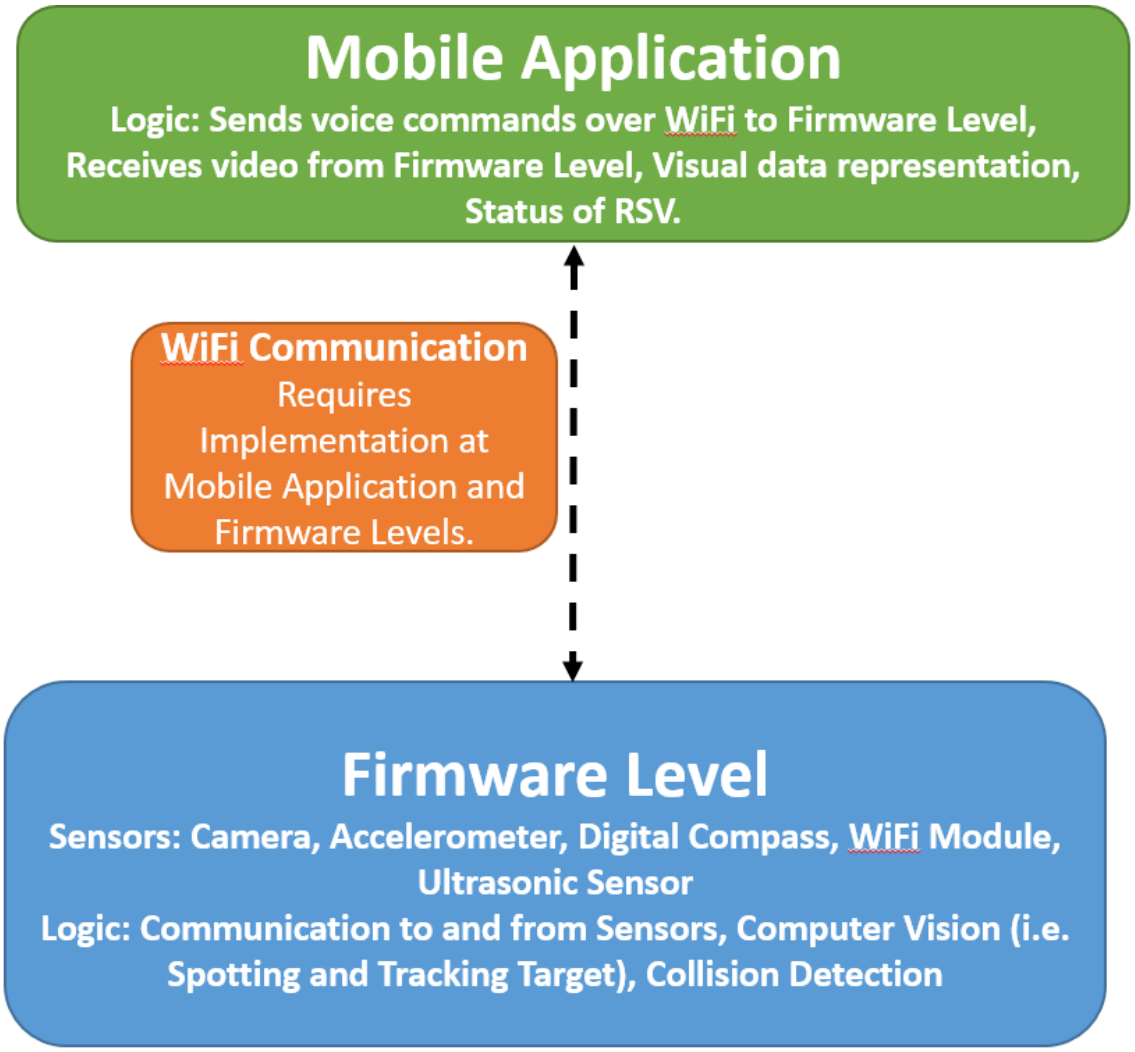


Figure 6-1: Software Design Levels

There will be a mobile application that will be used as the control module for the Robotic Surveillance Vehicle, this mobile application will include the ability to receive Robotic Surveillance Vehicle status data and video, and send voice commands to the Robotic Surveillance Vehicle to control the Robotic Surveillance Vehicle's active mode. This along with the firmware level will need to implement different aspects of the Wi-Fi communication between the mobile application controller and the Robotic Surveillance Vehicle microcontroller and sensors.

6.1 Firmware

Firmware is defined as the software used to allow communication of control, data, and data manipulation on hardware. In the case of the Robotic Surveillance Vehicle firmware will be used in the microcontroller chosen to allow interfacing and communication with the accelerometers, digital compass, camera, ultrasonic sensors, Wi-Fi module, and any other module that will be in the Robotic Surveillance Vehicle. All of these modules will be connected through our microcontroller. The microcontroller will receive data from each of these modules as well as control them (i.e. to slow down the vehicle, change directions, and when to transmit data).

Microcontrollers use an assembly language based software to develop the logic, some will allow use of intermediate level languages such as Verilog, or microcontrollers such as the Raspberry Pi will accept high level languages i.e. Java, C, and CPP. Modules will often have an open source library for various programming languages and will be supported by various microcontrollers. These will have to be verified that the library for each module is compatible with the microcontroller and language that is chosen to develop the firmware to be used in the Robotic Surveillance Vehicle.

The following sections will discuss the use of each module in the firmware implementation. Each module section will include a baseline code that will be built upon and used in the actual system to give a detailed look at the firmware logic used in each module and what role the module will play in the overall firmware logic. All libraries that are expected to be used in the final project Robotic Surveillance Vehicle will be included in Appendix B - Software Libraries section.

6.1.1 Digital Compass

A digital compass will be used to orientate the Robotic Surveillance Vehicle in addition to our accelerometer to ensure that the Robotic Surveillance Vehicle is always aware of its current orientation. This module will also be used to create a temporary memory of where the Robotic Surveillance Vehicle has been so that it can retrace its steps to return to a previous location. The base functionality of the

digital compass can be seen in the flowchart figure 6-2 below.

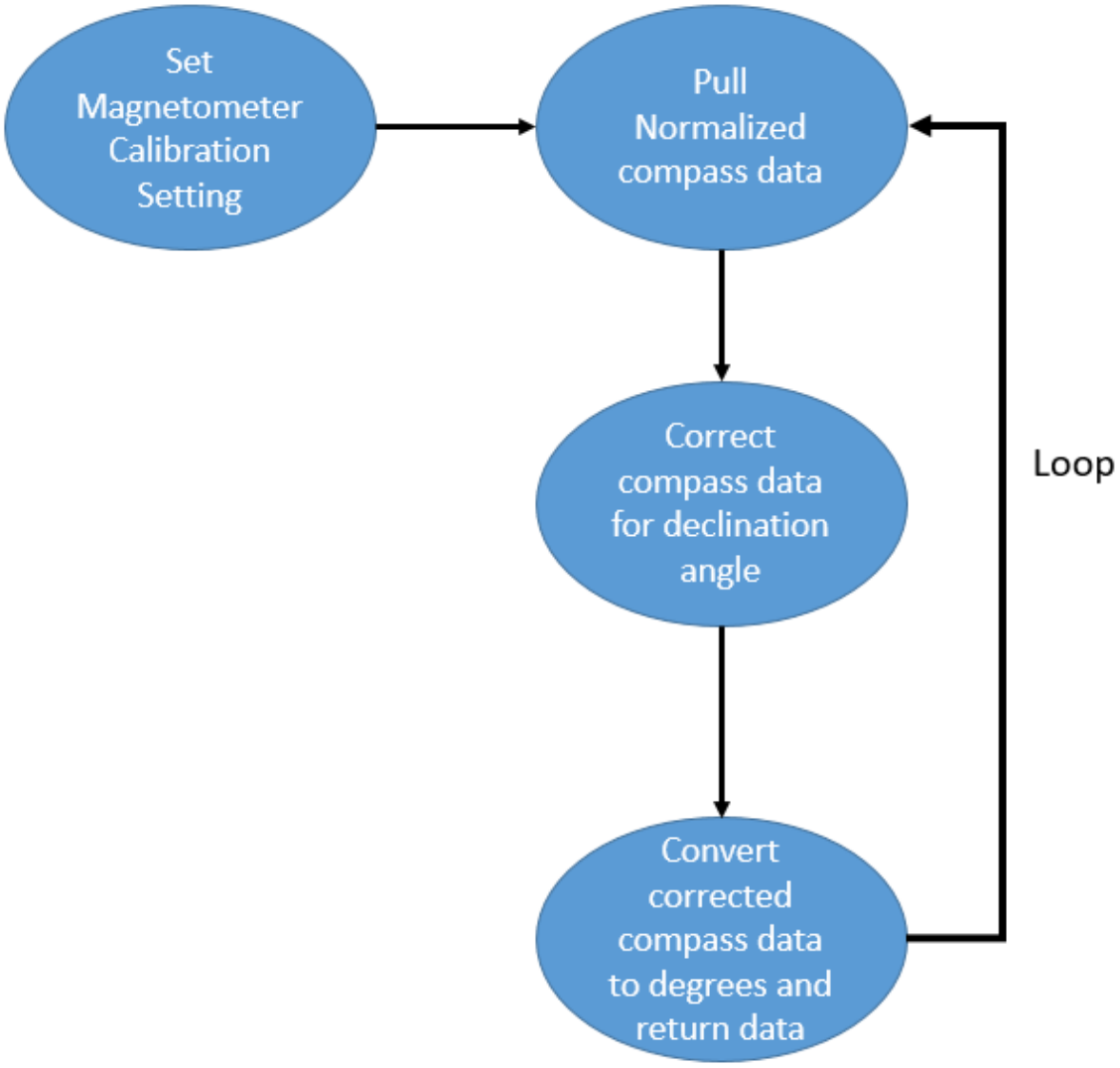


Figure 6-2: Magnetometer Logic Flowchart

The digital compass will have different frequency, gains, and measurement mode setting which will be set at the beginning of each startup of the Robotic Surveillance Vehicle. The default gain for these devices will be a gain of 1.3 dB, a default frequency of 15Hz, and a measurement mode set to continuous. The default settings will be used initially to determine if this gives us a precise enough measurement of the orientation of the Robotic Surveillance Vehicle. We may decrease the frequency of which we measure by using the single measurement mode or reducing the frequency setting and keeping the continuous measurement mode. If it is determined to save power and keep accuracy. Increased frequency can be used if we are not meeting our requirements due to lack of accuracy of this module at the default frequency. We would do this by

keeping continuous measurement mode and increasing the frequency setting.

Below is example code showing how to initialize the default setting and produce a heading in radians and degrees. This will be incorporated to ensure we gather data from the digital compass module to orientate the Robotic Surveillance Vehicle.

6.1.2 Ultrasonic Sensors

This section contains the software functions and needs of the ultrasonic sensors that will be placed on the Robotic Surveillance Vehicle to help in collision detection. Ultrasonic sensors use pulses of sound to determine how far from something the sensor is. The time to receive an echo similar to echolocation is used to compute the distance the sensor is from an object. This sensor will convert this echo into a low high voltage pulse logic that will be used by software in the form of a distance in either centimeter or inches, centimeters will be used as the default unit of measure. Below you can see a flowchart figure for how the ultrasonic sensor will collect data.

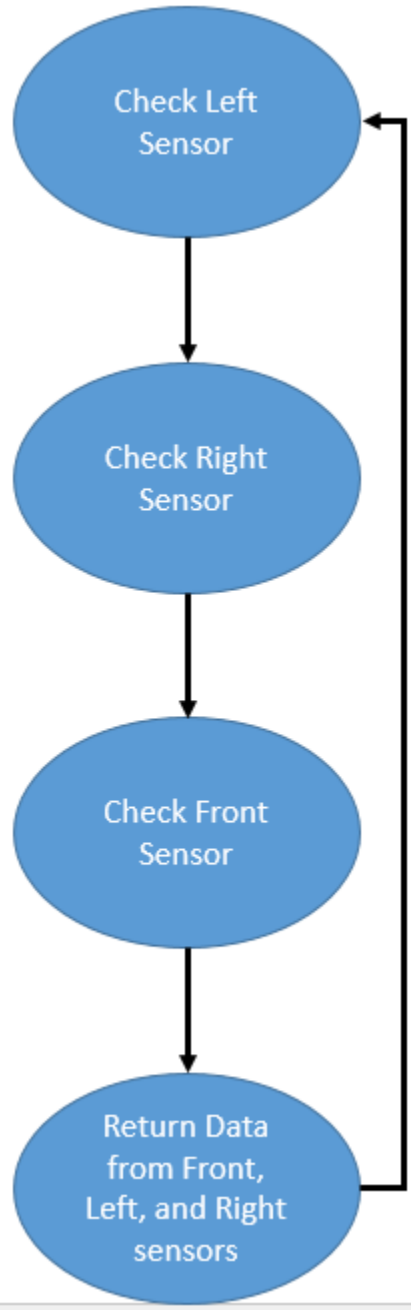


Figure 6-3: Ultrasonic Sensor Logic Flowchart

Multiple ultrasonic sensors are able to and will be implemented on the same board allowing us to detect possible collisions from multiple direction to ensure we can avoid unwanted collision, this will be discussed in more detail in the collision detection section as well as an example algorithm that will be used to perform the collision detection and vehicle correction. The Robotic Surveillance Vehicle will have three ultrasonic sensors implemented one in the front, left, and right of the vehicle.

Below is example code on how to pull the distance in centimeters from each of the three ultrasonic sensors.

6.1.3 Servo Controller

This section will contain an overview of the servo controls that will be used for the Robotic Surveillance Vehicle and what firmware implementations will be required for this controller. The servo controller will operate two servos that will provide movement to the Robotic Surveillance Vehicle. The servo controller is serially commanded accepting 8 Bit commands from the firmware. The various commands will be defined in the software libraries used to program the servo controller. This software will be crucial in the movement of the Robotic Surveillance Vehicle, allowing for manipulation of the servos during tracking, and collision detection and avoidance. This will be discussed in further detail in section 6.1.5 Collision Detection and Avoidance as the logic from this will directly affect how the servos will be adjusted to keep tracking and prevent collision.

The servo controller allows for up to two motors to be controlled in both forward and reverse directions. The servo controller also has access to a coast command that allows the servos to be put in a neutral setting where they will turn naturally. The speed being set to 0 is essentially a brake command providing no forward or backward movement and holding the servo.

Configuration controls are available through the serial commands as well as able to retrieve current and speed values from the servo controller. Error byte can be pulled and will be during communication testing as well as other possible debugging, lastly firmware version can be pulled from the servo controller which will only be used if update is necessary.

The servo code in Appendix C is example code for how to use the servo controller to run the servos.

6.1.4 Camera

This section will contain an overview on the camera’s firmware configuration since it will be implemented with the microcontroller. The camera’s main function will be to capture video that will be processed through a computer vision algorithm. The algorithm will be implemented in the firmware in conjunction with the collision detection and avoidance algorithm. These algorithms will need to be coordinated as not to break the other algorithm in running this one. The Robotic Surveillance Vehicle must also ensure it does not collide to cause a possible tracking loss. The Raspberry Pi microcontroller allows for direct connection to the Graphics Processing Unit (GPU) by the camera bypassing the Computer Processing Unit (CPU). This should allow for better video processing than other microcontrollers might offer, and this frees processing space from the CPU to be

used for our other sensors and algorithms.

Appendix C - Datasheets has some Python example code form how to capture video for some period of time using the Raspberry Pi camera. SimpleCV which is an open source computer vision library based in python will be used with this camera to develop the computer vision algorithm needed for the Robotic Surveillance Vehicle.

6.1.5 Collision Detection/Avoidance

This section will contain an in-depth look on how collision detection and avoidance will be implemented in the firmware of the microcontroller and how the other sensors mentioned above will be used in this process. This will not include full integration with the computer vision algorithm and the various modes that the Robotic Surveillance Vehicle will be able to be commanded into. This section is to look at the baseline collision detection and avoidance algorithm that will be used in all modes and modified based on the needs of those commanded modes.

The ultrasonic sensors are crucial to the logic on the Robotic Surveillance Vehicle since they will be providing distance data to the Robotic Surveillance Vehicle on where the Robotic Surveillance Vehicle might collide with an object. The ultrasonic sensors data will be a significant factor that is affecting the servos and changing the direction and speed of the Robotic Surveillance Vehicle if necessary.

To ensure the Robotic Surveillance Vehicle will not collide with various objects a threshold distance of 5cm will be maintained at all times from any object. After this there will be layers of reactions based on how close to an object the Robotic Surveillance Vehicle is. The range of distance that can be tracked in most ultrasonic modules is 5cm-400cm. Specific ultrasonic modules can go as low as 1cm and as high as 500cm away from the sensor itself. After the 5cm threshold distance Robotic Surveillance Vehicle responses will depend on the distance data obtained from the ultrasonic sensors. The following ranges will call for different responses from the Robotic Surveillance Vehicle: 6cm-20cm, 21cm-50cm, 51cm-100cm, and 100cm-400cm. These ranges and their corresponding Robotic Surveillance Vehicle responses are subject to change based on performance testing. The Robotic Surveillance Vehicle will have a front, left, and right ultrasonic sensor as discussed in section 6.1.2 Ultrasonic Sensors, the ranges discussed above will be implemented for each of these sensors with varying responses from the Robotic Surveillance Vehicle based on the sensor location, and other data from varying sensors on the Robotic Surveillance Vehicle.

The front ultrasonic sensor will be used to maintain speed and prevent direct

collision with an object in front of the Robotic Surveillance Vehicle. At and below the 5cm threshold distance the servos will be stopped to stop the Robotic Surveillance Vehicle. After this threshold distance the ranges discussed above will implement a faster speed as distance from an object increases. The exact speeds and their increases from one range to the next has yet to be determined. This is to ensure the Robotic Surveillance Vehicle cannot go from its top speed to stopping there will always be an acceleration or deceleration based on if the Robotic Surveillance Vehicle is approaching an object.

The left and right ultrasonic sensors will have a much more intricate position in the collision detection and avoidance algorithm. A sampling of distances will be taken from both the left and right sensors simultaneously. This sampling will be temporarily stored and used to determine one of the example cases described below:
Cases:

- The sensor sampling is relatively steady in value, there is no movement towards or away from the object on that side.
- The sensor sampling is decreasing in value over time, the extent to which the value is decreasing will be used to determine how quickly the Robotic Surveillance Vehicle is approaching the object.
- The sensor sampling is increasing in value over time, the extent to which the value is increasing over the sampling will be used to determine how quickly the Robotic Surveillance Vehicle is receding from the object.

These cases will be used in conjunction to determine what is happening in the environment around the Robotic Surveillance Vehicle:
Environment Cases:

- Both side’s sampling is steady, there is no change in the walls near the Robotic Surveillance Vehicle.
- One side’s sampling is changing in value while the other’s remains steady. One wall is becoming further away or closer.
- Both side samplings are changing in value, meaning the walls are opening up, closing in on the Robotic Surveillance Vehicle, or one is becoming closer while the other is further away.

These cases will be used in determining the appropriate Robotic Surveillance Vehicle response, along with a ranging system similar to that used by the front sensor. An overview of the collision detection and avoidance algorithm as described in detail above can be seen in Figure 6-4 below.

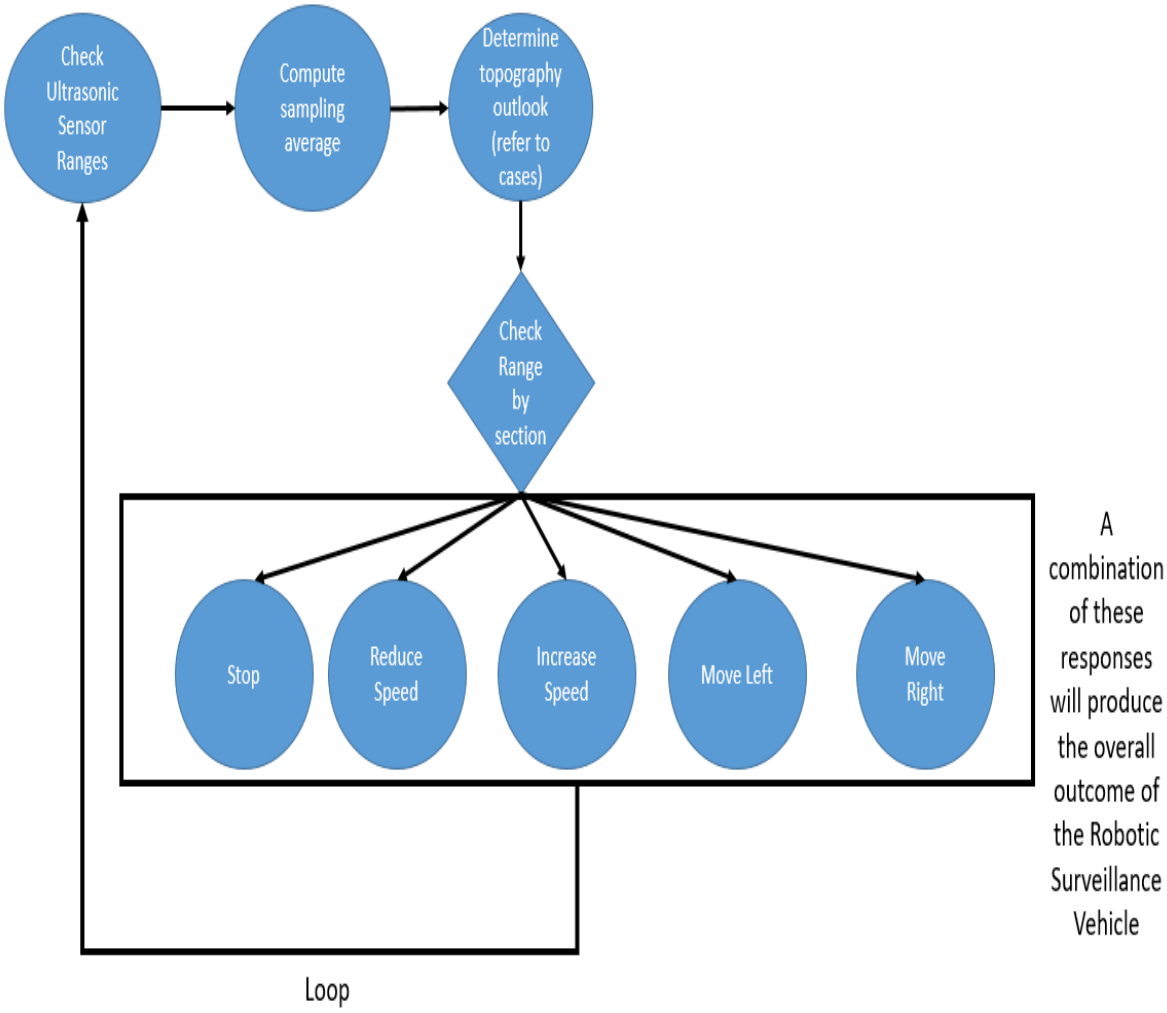


Figure 6-4: Collision Detection and Avoidance Flowchart

6.1.6 Wi-Fi Module

This section contains an overview of the Wi-Fi module that will be used on the Robotic Surveillance Vehicle and how it will be implemented in the software of the Robotic Surveillance Vehicle.

The Wi-Fi module is already implemented into the Raspberry Pi microcontroller and has python implementation through the Raspberry Pi interface. This module will be used to implement communication between the Robotic Surveillance Vehicle and the smart device via the mobile application. This will be done by implementing sending and receiving protocols for both the Robotic Surveillance Vehicle and the mobile application. This will be used by the mobile application to send voice commands to the Robotic Surveillance Vehicle as well as receive video data and status reports from the various sensors.

A User Datagram Protocol known as UDP will be used as the specific protocol implemented for the Robotic Surveillance Vehicle as it most efficiently fits the needs of our system. This is described in more detail in the Data Transmission over Wi-Fi section of this document.

6.2 Data Transmission over Wi-Fi

Wi-Fi will be used as the form of data transfer for both the audio voice commands that will be used to control the Robotic Surveillance Vehicle as well as the method for which to stream the video from the Robotic Surveillance Vehicle back to the user through the mobile application. This was chosen in lieu of Bluetooth due to the range limitations of Bluetooth, as well as the more in depth data transfer functions that are already implemented in the Wi-Fi communication system. These Wi-Fi protocols are more reliable and quicker than a Bluetooth connection would be. An overview of the transmissions used for each situation and page can be seen in Figure 6-5.

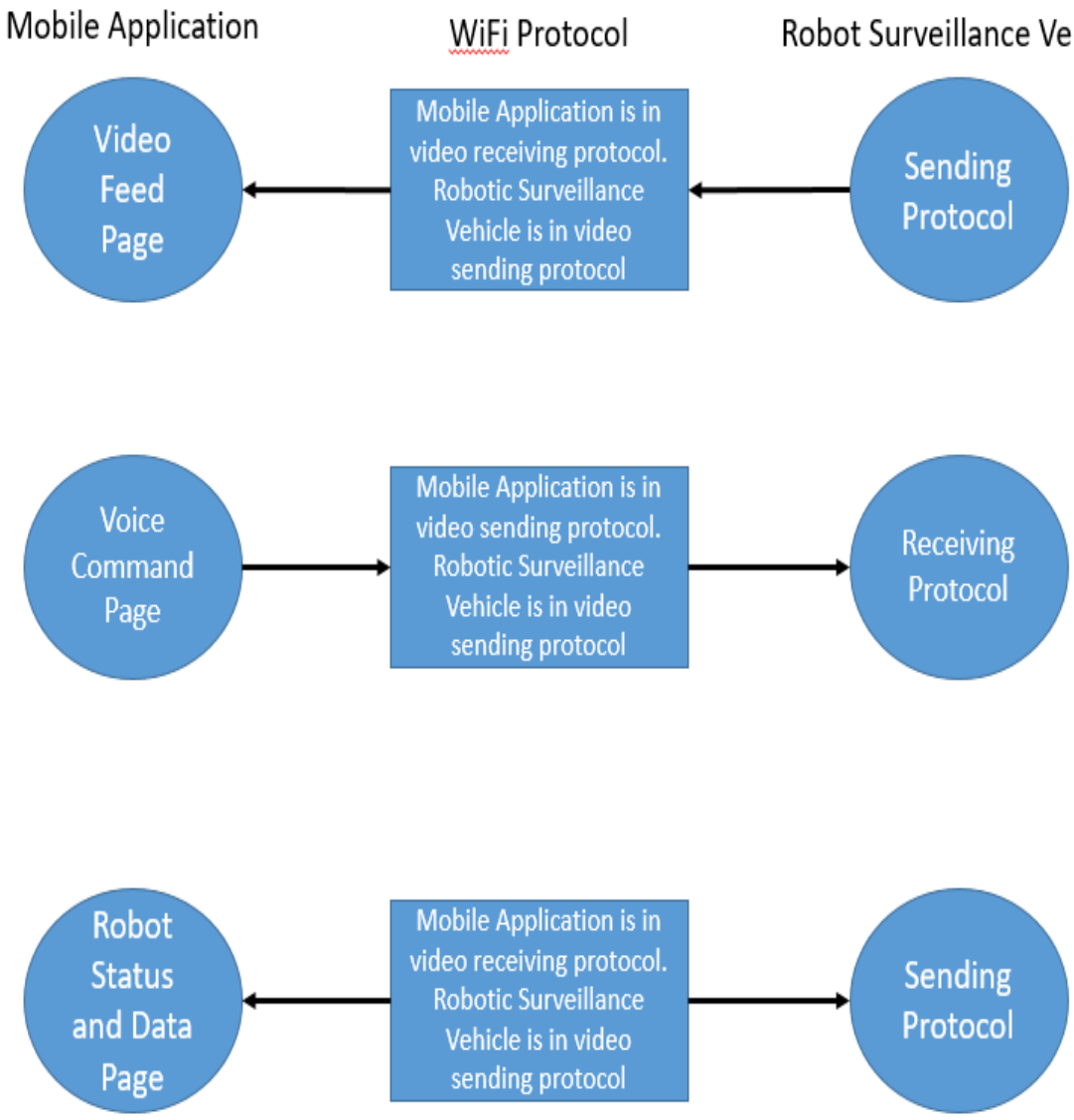


Figure 6-5: Wi-Fi Data Transmission Overview

A UDP connection will be the preferred protocol used for our system due to how data transfer occurs on the UPD protocol. The UPD protocol is quick and efficient when transferring data between stations. Since the UDP connection will only be used between the mobile device and the Robotic Surveillance Vehicle there is little concern of data packets being dropped due other connections. UDP does not have an acknowledgement system and does not have to receive the data packets in order. This means the sender of the data packets will never know what data has been properly received and in what order it is received.

The UPD protocol can be implemented in multiple different ways through different programming languages. The mobile application will need to implement a receiver when in streaming mode to receive video from the Robotic

Surveillance Vehicle. The mobile application can be put into control mode to become a sender across the Wi-Fi for the voice commands to be sent to the Robotic Surveillance Vehicle. On the Robotic Surveillance Vehicle side of the Wi-Fi implementation if possible Java will be used in conjunction with the mobile application to implement the sending and receiving of the camera video and voice commands. This will have to be integrated with all other sensors discussed in the previous sections.

6.3 Mobile Application

This section contains the details of the logic and functionality the mobile application will have in the development of the Robotic Surveillance Vehicle. The objective of the mobile application is to send commands to the Robotic Surveillance Vehicle, receive video feed from the Robotic Surveillance Vehicle and to check the status of the sensors and relevant data returns. The sections below will include a description as well as an example interface for each of the three pages our mobile application will have. This will include each page's primary function in the mobile application and the Robotic Surveillance Vehicle system as a whole.

6.2.1 Voice Command Page

This section is to detail the page of the mobile application that will allow for voice commands to be inputted to control the Robot Surveillance Vehicle. This page when selected will display a buttons of voice commands that will be accepted by the Robotic Surveillance Vehicle. These buttons are selectable, but the primary command method would be through voice commands allowing for commanding of the different modes of the Robotic Surveillance Vehicle. A template of the user interface as described above is shown in Figure 6-6 below.

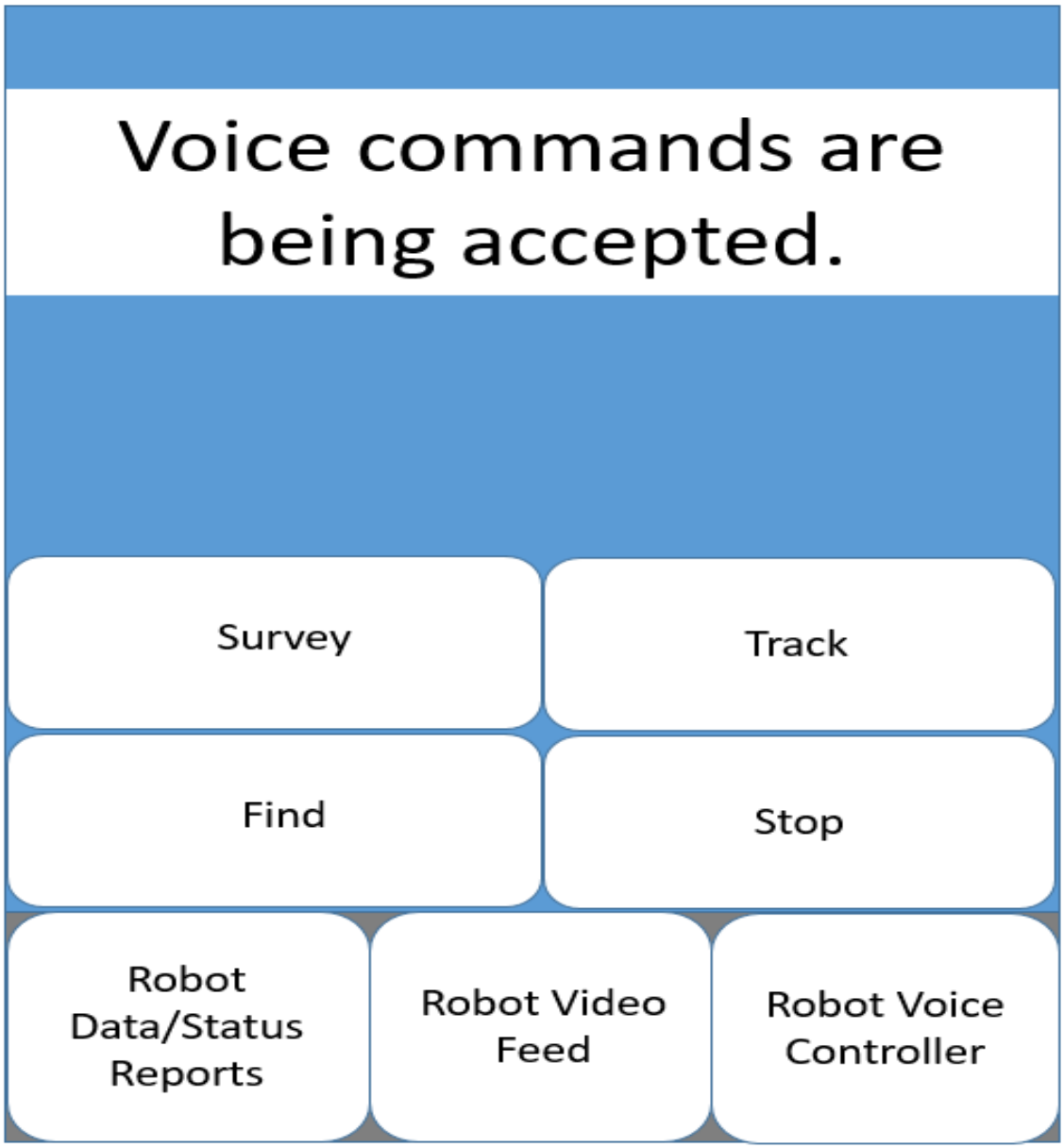


Figure 6-: Robot Voice Controller User Interface

This page will function by implementing speech recognition through the Java speech API. Audio coming through the smart device and mobile application will be filtered through the speech recognition software. Any filtered audio that is recognized as one of the four voice commands will be transmitted through the Wi-Fi connection between the smart device and the Robotic Surveillance Vehicle. The UDP protocol will be the Wi-Fi protocol used to send data from the smart device to the Robotic Surveillance Vehicle. Once the command is received from the Robotic Surveillance Vehicle the firmware will take over and be set in a logic command based on the voice command given. Below is the logic and data flow diagram of the software as described in this section.

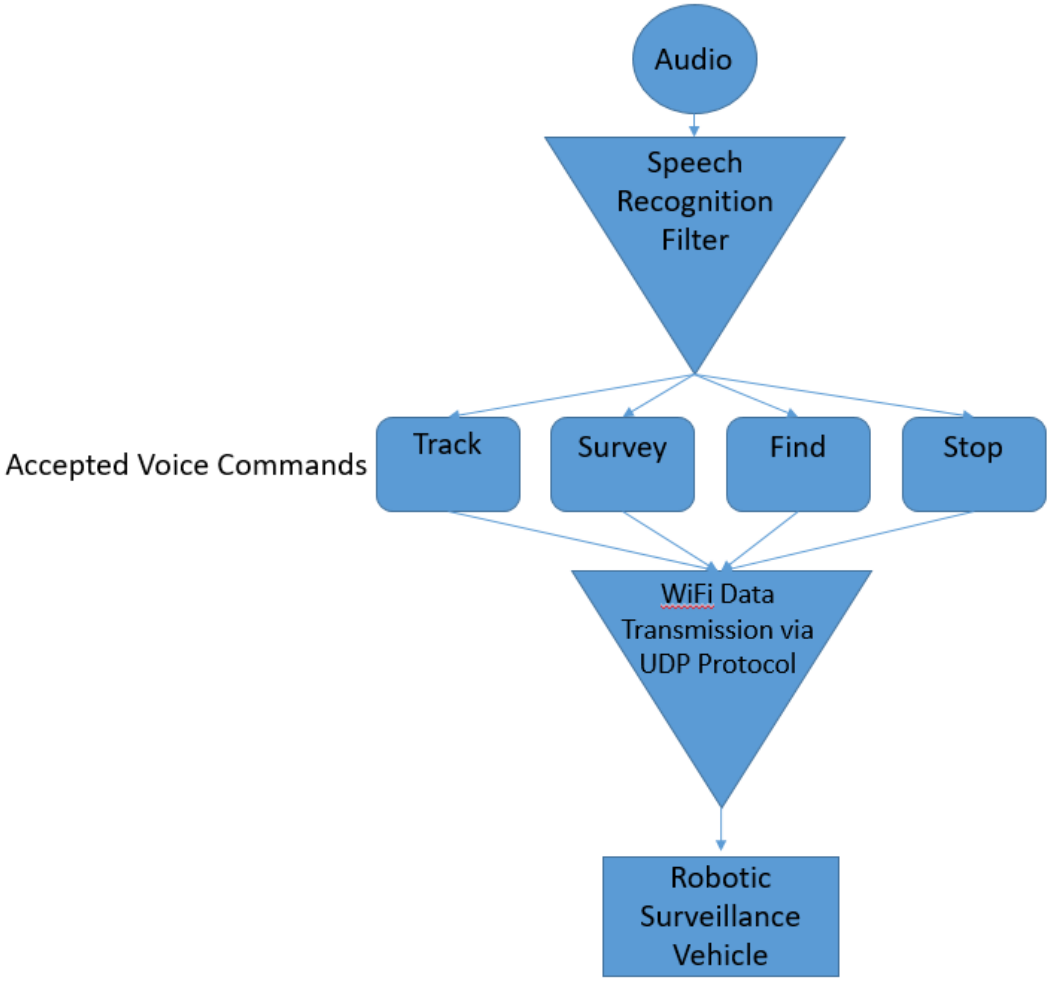


Figure 6-7: Voice Command Flow Chart

6.2.2 Video Feed Page

This section is to detail the page of the mobile application that will allow for video feed to be viewed from the Robotic Surveillance Vehicle. This page when selected will display a video stream from the Robotic Surveillance Vehicle. This is the simplest section as the video stream is the only functionality other than the ability to switch between the pages. Shown in Figure 6-8 below is the video feed page user interface as described in this section.

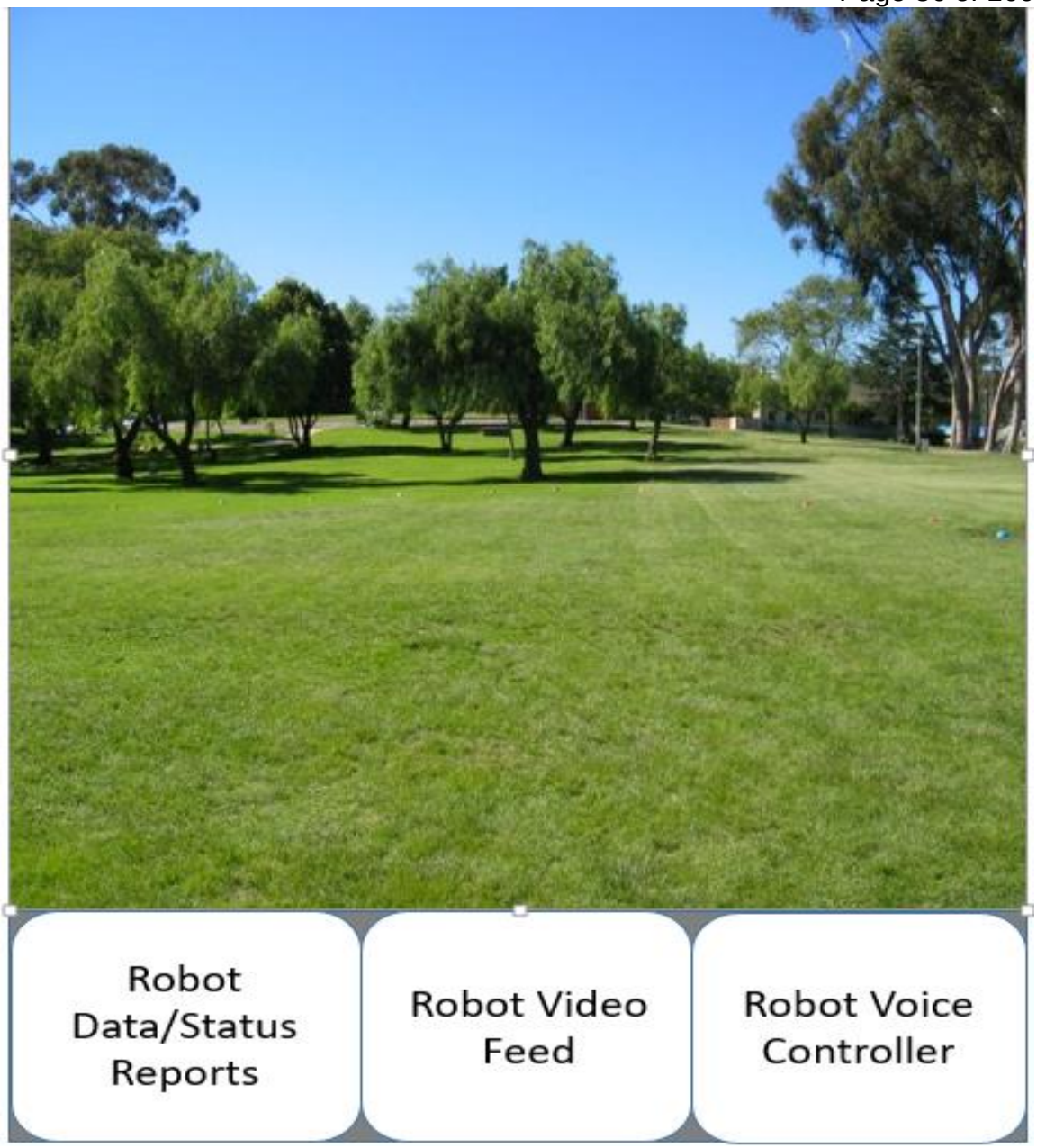


Figure 6-8: User Interface of Robot Video Feed Page

As discussed in the Wi-Fi data transmission sections a UDP connection will be used for fast transfer of video to the mobile application from the Robotic Surveillance Vehicle. The video feed will be run through the computer vision/tracking algorithm for points of interest during the surveying and tracking modes of the Robotic Surveillance Vehicle. This will work hand and hand with the collision detection and avoidance algorithm to direct the Robotic Surveillance Vehicle both of these algorithms are discussed in more depth in their respective sections.

6.2.3 Robot Data/Status Reports Page

This section contains the details on the Robot Data and Status Reports page. This page will contain reports from the various sensors housed on the Robotic Surveillance Vehicle. This will contain anything from the status of these sensors to ensure they are working correctly to the data they are relaying to be used in the processor. The setup of this data and its user interface can be seen in Figure 6-9 below.

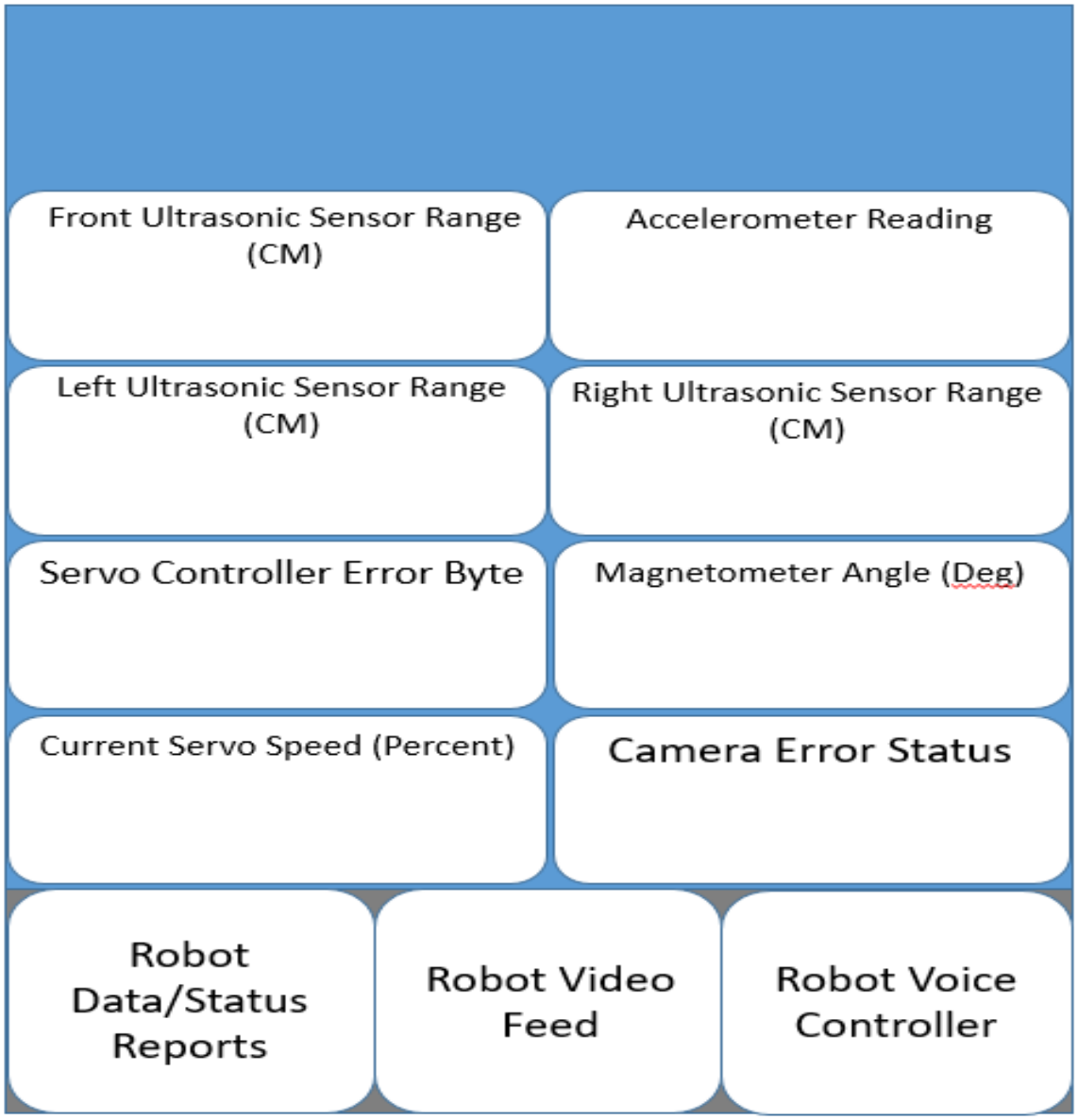


Figure 6-9: Robot Data / Status Reports User Interface

6.4 Computer Vision

This section contains an in-depth look at the computer vision portion of the Robotic

Surveillance Vehicle that will be used in the survey and tracking modes of the Robotic Surveillance Vehicle. The algorithm used for the computer vision will need to interface and influence other portions of the firmware that have been previously discussed such as the movement of the Robotic Surveillance Vehicle and the collision detection and avoidance algorithm it uses. Figure 6-10 below shows a flowchart of the computer vision algorithm used by the Robotic Surveillance Vehicle.

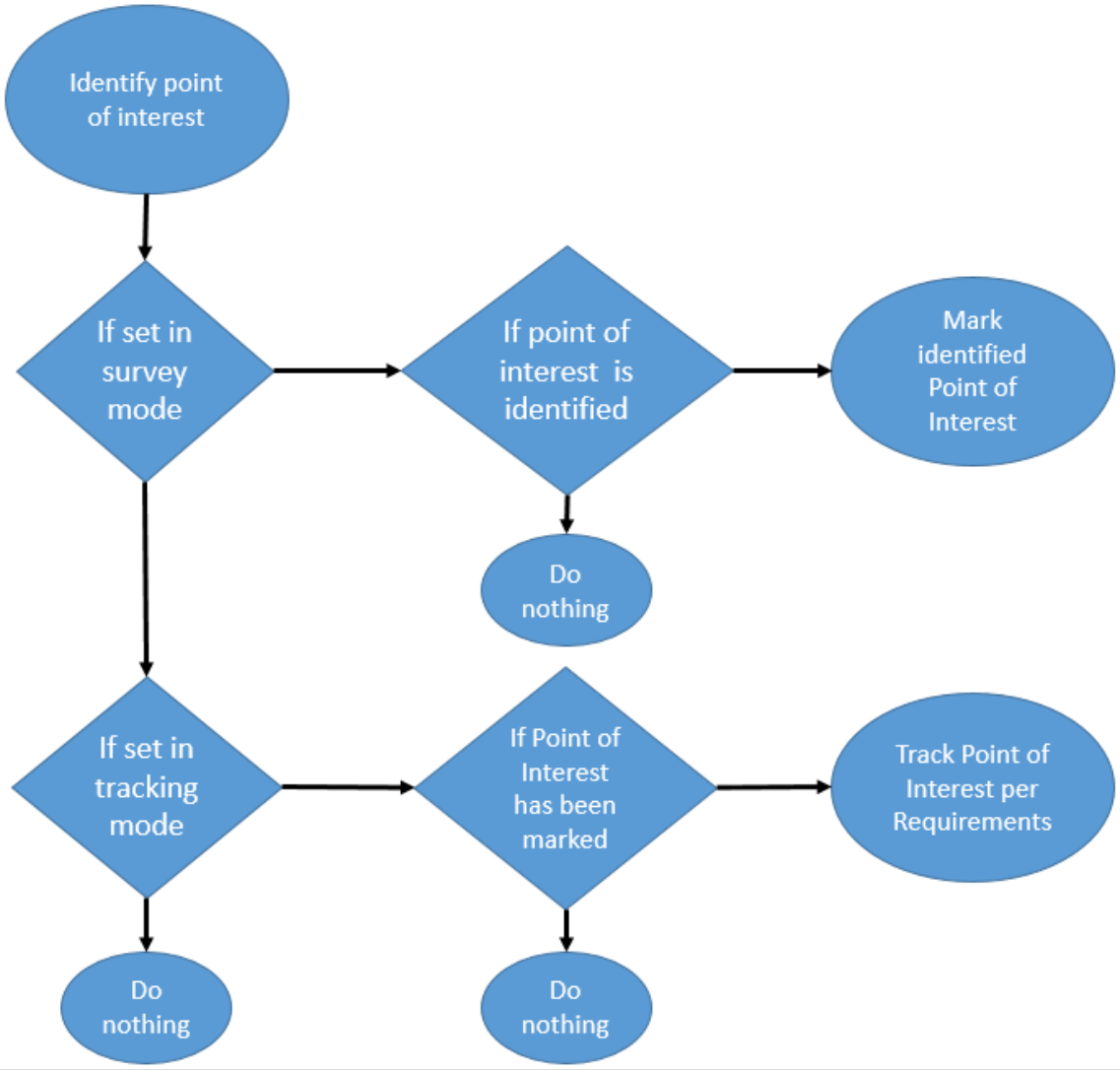


Figure 6-10: Computer Vision Algorithm Flow Chart

The computer vision portion of the firmware needs to be able to identify a shape of a particular color. The key to implementing this is to define and mark the edges of the shape easiest of shapes to do this with will be a square or rectangular shape. Then using a specific color that is easily outlined in contrast to its surrounding environments such as a bold red. These parameters will make identifying the pre-determined point of interest easier to allow for focus on the portion of the algorithm that will affect servo and motor performance based on

the visual feed and logic from the camera. SimpleCV, CV standing for computer vision is a program that provides an easy library for implementing computer vision on microcontrollers for project such as this one. The flowchart below shows the logic mentioned above before the data is used to control the Robotic Surveillance Vehicle.

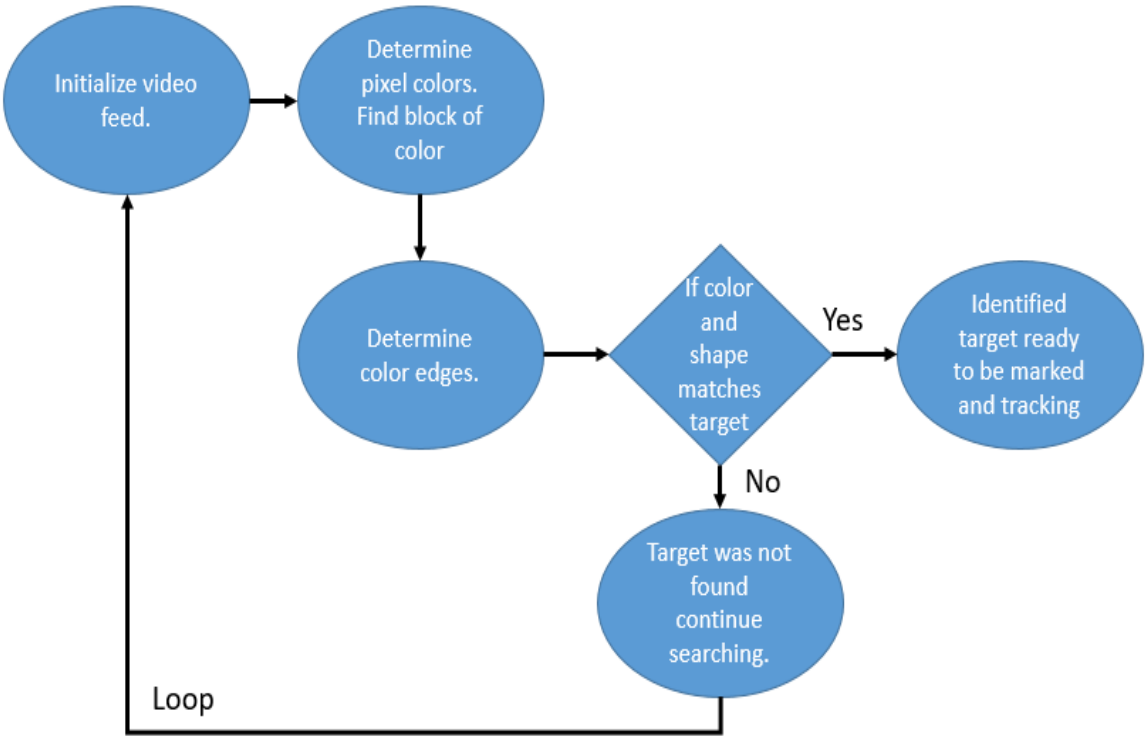


Figure 6-11: Computer Vision Algorithm to Determine a Target

Since the parameters for the point of interest has been defined for the benefit of the system now a look into how this will be implemented into the whole algorithm. During the survey mode the Robotic Surveillance Vehicle will scan an area of at least 120 degrees in front of the Robotic Surveillance Vehicle. During this time the Robotic Surveillance Vehicle will be attempting to identify a point of interest per the parameters mentioned above. Once the point of interest is identified it will be marked. Marking is defined as holding position on the point of interest keeping it in front of the Robotic Surveillance Vehicle and centered in the camera view to allow for tracking if initiated by the user via the smart device.

Once marked as a point of interest via the survey mode of the Robotic Surveillance Vehicle tracking mode can be initiated. Tracking is defined as the Robotic Surveillance Vehicle following the point of interest at a maximum distance as well as keeping the point of interest centered in the camera view of the vehicle. This is where the algorithm interfaces with the servo controller to direct the Robotic Surveillance Vehicle to track the point of interest.

The major controls that will be implemented during the computer vision portion of the firmware will be to speed up, slow down, and direct the Robotic Surveillance Vehicle to the left or right.
Cases:

- The point of interest is enlarged compared to the defined center tracking target from the camera of the Robotic Surveillance Vehicle. The Robotic Surveillance Vehicle is too close to the point of interest and speed will need to be decreased to center the point of interest in the track.
- The point of interest is shrunk in the center of the track. The Robotic Surveillance Vehicle will need to get closer to the point of interest by increasing its speed to center the point of interest in the track.
- The point of interest is not center by being shifted to the left or the right. To follow this, shift the Robotic Surveillance Vehicle will need to increase or slow speed of one motor in order to turn the Robotic Surveillance Vehicle and center the track.
- Lastly and most commonly will be some combination of these 3 above cases that will need to be implemented together to provide the proper correction of the Robotic Surveillance Vehicle to continue to keep the track centered.

6.5 Software Summary

The Robotic Surveillance Vehicle will have multiple software components including the firmware onboard the Raspberry Pi microcontroller, and the mobile application software.

The firmware onboard the Raspberry Pi microcontroller will consist of the various sensors used to control the Robotic Surveillance Vehicle such as, the servo controller, ultrasonic sensors, camera, and digital compass. These will be used to implement the collision detection and avoidance as well as, computer vision algorithms to guide the Robotic Surveillance Vehicle as it tracks. Surveying and tracking points of interest will be the Robotic Surveillance Vehicle's primary function, the software algorithms mentioned above will be integral to these functions. These functions will be implemented on the Robotic Surveillance Vehicle, but will need to be integrated and communicate with the smart device via the mobile application used on it.

The software to be used outside of the Robotic Surveillance Vehicle will be handled through a mobile application on a smart device. The Robotic Surveillance Vehicle will be able to communicate with the mobile application via a UDP Wi-Fi connection. This will allow for voice commands to be transmitted from the smart device to the Robotic Surveillance Vehicle as well as, transmitting video and status updates of the sensors from the Robotic Surveillance Vehicle to the smart device.

7. Project Prototype Construction and Coding

The prototype construction will begin once the hardware testing has been completed list in the next section of the report. During the testing phase the PCB board will be designed and ordered to match the specifications discussed earlier. The PCB will integrate all of the required sensor systems so that the components are consolidated and can be placed on the same level.

7.1 Integrated Schematics

Below shows the integrated schematic for all of the system components. The schematic represents the previous schematics contained in the document. It includes the following major sections:

1. Sensor Array
    a. 3 x Ultrasonic Sensors
    b. Magnetometer
    c. Pi Camera Version 2.1
2. Rotary Motor Control
    a. Servo Driver
    b. 2 x Servos with gearbox
3. Rotary Encoder
    a. Analog to Digital Converter
    b. 2 x Rotary Encoders
4. Raspberry PI MCU
    a. A basic layout of the device as associated with the components

The integrated schematic is not to size or scale but is representative of the entire interface structure to the Vehicle. Some of the sensor arrays are summarized by larger components, the Sensor Array and Rotary Encoder Array are fully detailed in their respective schematics and the components listed below are for a high level view of the layout. These will be used in the final PCB design however the PCB schematic will be completed in Senior Design 2.

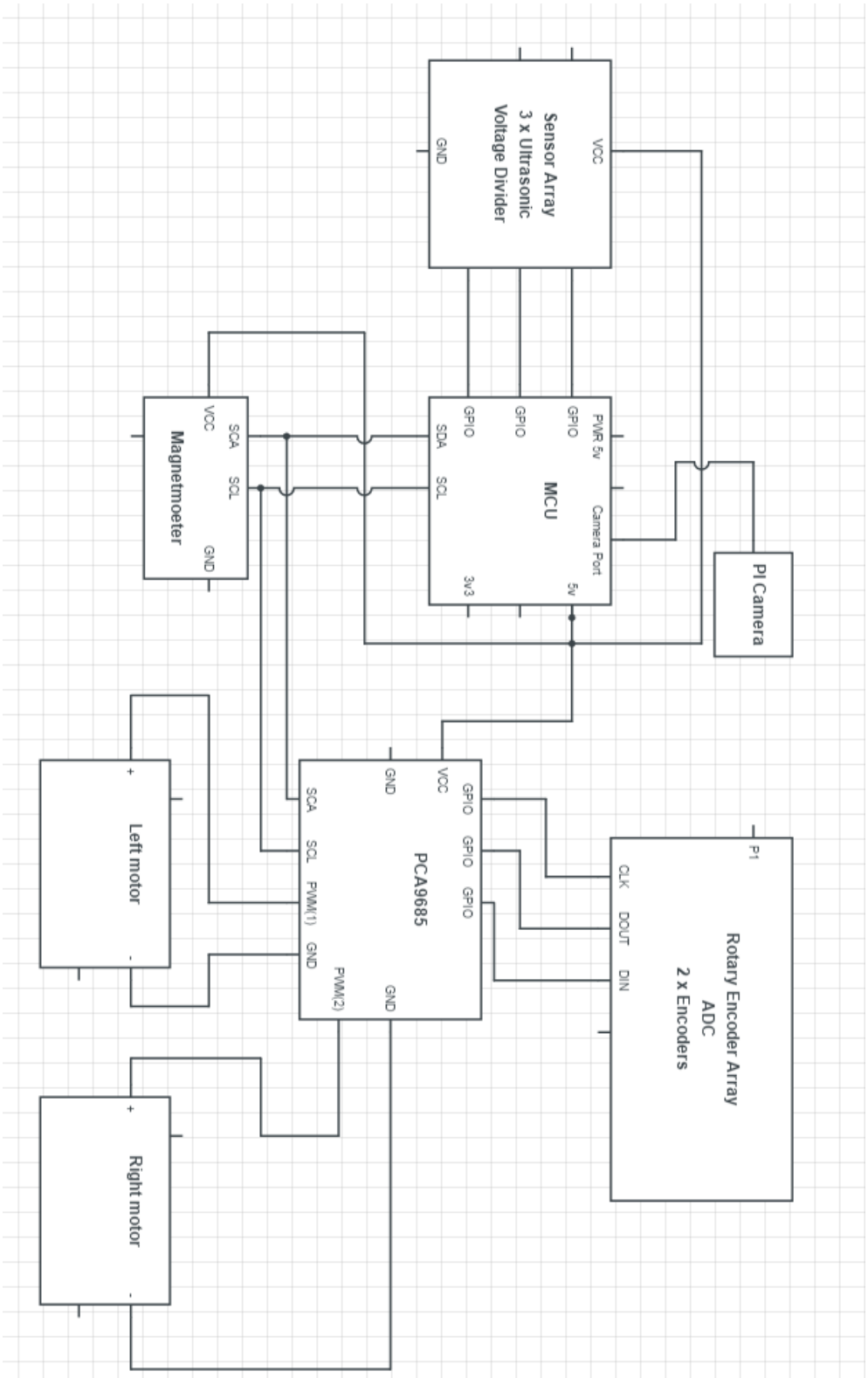

Sensor Array
3 x Ultrasonic
Voltage Divider
VCC
GND
MCU
GPIO
GPIO
GPIO
PWR 5v
SDA
SCL
Camera Port
3v3
5v
Magnetmoeter
VCC
SCA
SCL
GND
PI Camera
PCA9685
GND
VCC
GPIO
GPIO
GPIO
SCA
SCL
PWM(1)
GND
PWM(2)
GND
Left motor
+
-
Right motor
+
-
P1
Rotary Encoder Array
ADC
2 x Encoders
CLK
DOUT
DIN

Figure 7-1: Integrated Schematic

7.2 PCB Vendor and Assembly

Eagle Cad will be the primary software used to make the final PCB assembly. It will make use of the integrated schematic for a top level layout design and all of the other schematic designs. There are some primary concerns and precautions to consider when designing the PCB and those are the specific currents and voltages on each line, the line layout, the device power feeds and the physical size of the PCB.

The currents and voltages will need to be verified when we are in the testing phase of the project. Each of the peripherals will have different tolerances for voltage and current draw. This will be important so that we don't overload the raspberry pi GPIO lines as well as ensure that there are no disconnects when one device may require a different voltage than another. The i2c lines generally recommend a 10kΩ pull-up resistor for each line. However, others recommend calculating the specific load capacitances. Before assembling the PCB, the exact load capacitance will need to be determined both in practice and calculation. Prior to determining the PCB final layout, we will test the exact pull-up resistors needed.

The line layout for the device will be covered in the Eagle Cad software. The benefit to using this software is that it is an application that takes schematic drawings and directly exports them to PCB layout manufacturing. We will use a PCB manufacturer that will suit our time constraint needs and is of market level cost. Currently there has been recommendation to use the Elecrow company for their quick turnaround time. Once we produce the PCB drawing if their prices are within the average market range with no more than a 10-15% variation then we will purchase the PCB from them.

The device power feed lines will be important. It will be important for us to test the maximum power draw through the battery and the MCU to ensure that we remain within operation tolerance. The power supply is limited in the supply current and the MCU is limited in the amount of power that can be drawn from the device. Since the components offer a range of values and do not accurately document their max. current draw we will need data from our own testing to verify the size of the power supply needed and the ability for the MCU to power the components.

The final important layout aspect to consider is the physical dimensions of the PCB. The ultrasonic sensors will need to be located on the outside edges to they don't pick up and interference from the device. We also need them in specific locations relative to the size of the device. For that reason, the PCB size may be larger than needed for the electrical components in order to satisfy the size

requirements on the device.

Finally, when we move on to the prototype construction in Senior Design 2 we will utilize the data collected from the tests. We will construct two different prototypes to help smooth the process and troubleshoot any issues. The first prototype will consist of a breadboard based layout with the sensors attached and the servos in place. We will fix temporarily attach all of the components to the device so that we can run through some of the testing programs and basic operations. This will allow us to troubleshoot any issues before we create the permanent PCB mount for the device. At this point we will most likely already have ordered the PCB so pending any design changes we will attempt to troubleshoot the problems with the PCB created based of the single component testing. If needed, we will rush a second PCB but we feel confident from component and system level testing that we can mitigate any potential errors.

The second prototype will be the test vehicle. This version won't have jumper cables unless absolutely needed and most of the components will be soldered onto the PCB board. At this point of the testing process we expect everything to be operating correctly and we will be optimizing the coding to ensure we meet all of our design objectives. If we run into electrical component issues, we will replace them with the extras we have ordered. If design changes are needed, we will attempt to modify the PCB board or order a second board to alleviate this. If absolutely necessary, we will redesign the original PCB of the vehicle. If it comes to this, we will consult with the program director on the best method moving forward in order to complete the project within all of our time constraints.

7.3 Senior Design 2 Alternate design concept

While researching the design of the device we've realized that our vehicle uses multiple components that require Pulse Width Modulation. This is used in the Ultrasonic Sensors, Rotary encoders and the Servo Driver. Due to this requirement we've implemented and Analog to Digital converter, a voltage divider and the General Purpose Input and Output pins of the MCU. After much research we've come across another solution that varies from the processes described above. In this solution we will utilize either a small MCU that can be programmed with the primary MCU. This device will be small in size, around the same number of pins as a typical Op-Amp but will be able to perform basic logic commands. The benefit to using the device is we will be able to combine some of the functionality of the Analog Digital Converter, the Servo Driver and the GPIO pins used in the sensor Array. The Servo driver is meant to be used with up to 16 different servos while the analog to digital converter is capable of 8 channels. Since this is far exceeding the needs of our device we have determine that we can replace these components plus the logic inherent in the GPIO to Ultrasonic Sensor Array with a Slave MCU.

The Slave and Master Concept isn't a new concept and has been utilized for some time primarily being credited to the Motorola Company. [17] This concept arose as a way to communicate with a Master controller and Slave controllers. The primary method of communication is the MOSI and MISO lines described below. Each of the slaves can be wired up in parallel with a Slave Select line enabling the device the Master MCU chooses to communicate with. The protocol utilizes Serial Synchronous Interface which means that the device will send data over sequentially one bit at a time. The advantage is that this means the devices can use less lines to communicate with. To ensure that the two devices are communicating properly and data is not lost the Master MCU will set a serial clock. The Slave MCU will communicate synchronously with the Master MCU via the two serial lines, MOST and MISO.

The Slave MCU we researched is the ATtiny25/85 produced by Atmel. They have the following characteristics shown in table 7-1. We also included a larger MCU that had extra pins as the ATtiny25/85 do not have enough pins to accommodate the number of devices for our project.

	ATtiny25	ATtiny85	ATtiny2313A
Flash(KiB)	2	8	2
RAM (bytes)	128	512	128
ADC Channels	4	4	-
PWM	2x2 sharing 3 pins	2x2 sharing 3 pins	1x4
Operating Freq (MHz)	20	20	20
Number of Pins	8	8	20

Table 7-1: MCU comparisons

We will need to research the design of the master and slave concept further to determine the exact specifications that we can replicate to help remove the Servo Driver and the ADC. Depending on the desired layout we may utilize multiple smaller MCU's or a larger MCU to control all of the devices.

There are major benefits to using an MCU slave master design concept over the use of the traditional single MCU. One of the biggest benefits is the Voltage tolerance of the smaller MCU's. The voltage range for the I/O ports are typically 1.8 to 5.5 volts. This is of particular importance to the ultrasonic sensors as they will output in 5 volts and our Raspberry Pi can only handle a 3.3 volt input to the GPIO. The next advantage is some come with built in analog to digital

converters, this can help us with our rotary encoders as we won't need a larger fully dedicated ADC. Since these slave type MCU's typically perform multiple functions it will allow us to utilize the ADC while still operating the devices that require PWM. The last beneficial ability is that the device can communicate i2c. The reason why this is important is that will reduce the need of the GPIO lines. It will allow the Raspberry pi communicate with all of our components via two lines. The ATtiny can be programmed via the large microcontrollers including our choice of the raspberry pi. We will be able to specify any of the coding inside the smaller device so that it can act as an interface to the Sensor Array and Servo Controller. We can specify the i2c address so we won't have any conflicts with the magnetometer. This will also allow us to run multiple ATtiny's in parallel so that each can perform a different function.

Based off our preliminary research we know that the smaller MCU can drive up to 5 servos, but we may be able to add more components to the MCU depending on the coding we choose. I have seen examples of running more than 5 servos but that will be dependent on the demands of the code and how often we need to communicate with our devices. We will be able to determine this in the testing phase and will need additional research in SD2 of the limits of the slave MCU.

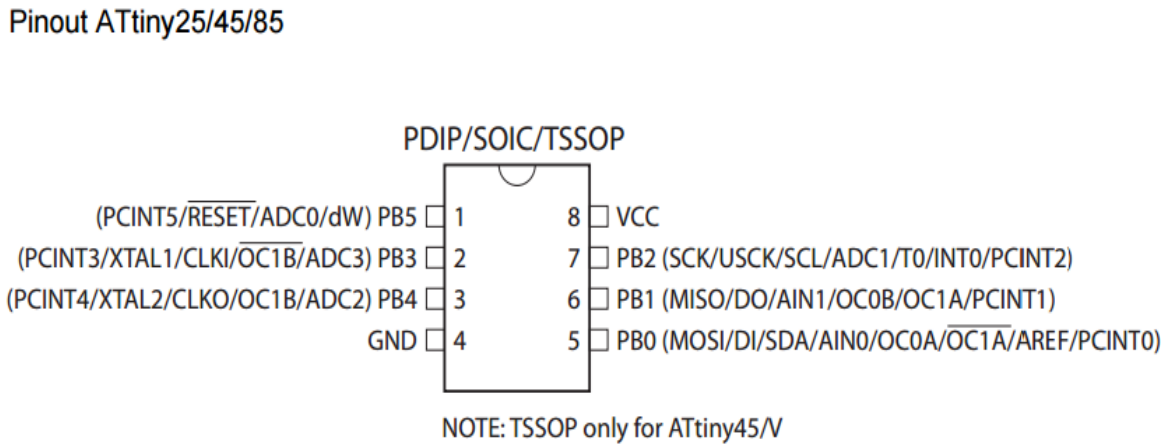


Figure 7-2: ATtiny pin diagram [7]

Above shows the pin diagram for our suggested slave MCU. Pin 7 and Pin 5 will need to be utilized for the communication line to the master MCU. That will leave us with Pin 1,2,3 and 6.

In the use of the Ultrasonic Sensor array that will gives us one spare pin as each array can connect the Echo and Trigger Pin to one of the pins in the slave MCU. When programming the device, we can utilize the pin to trigger the device for a reading and then use that same pin to receive the echo. Depending on the speed and memory specification we may have the ATtiny process the data or send the raw data back to the master MCU for processing.

When using the servo controller, we can also utilize those same pins as they operate via PWM similar to the ultrasonic sensors. The rotary encoder utilizes an analog to digital converter so we would use the same pins for this section as well. We would need to research and test the capabilities of the MCU running multiple devices to ensure it is capable of simultaneous operation of all the devices.

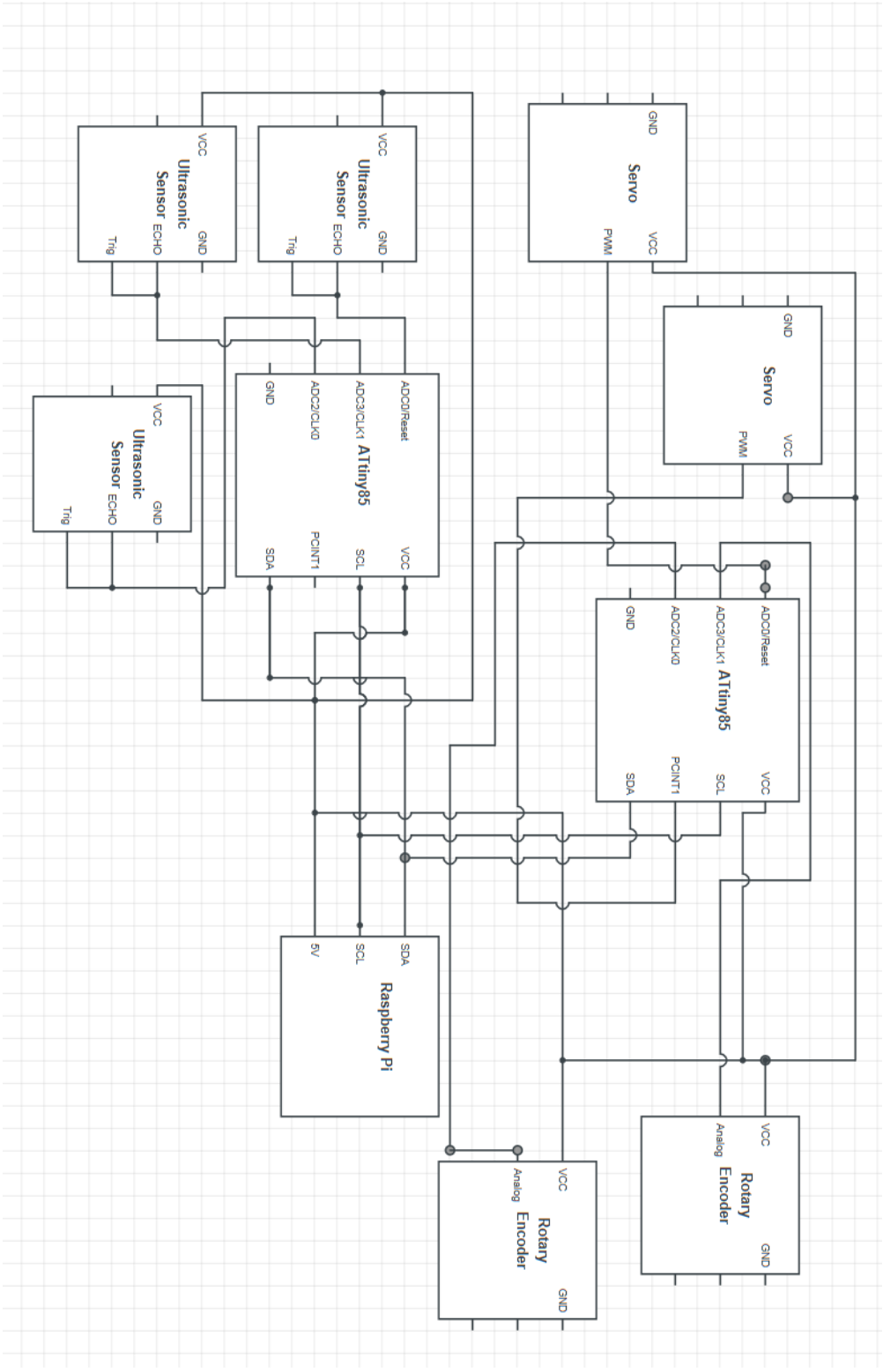

Servo
GND
PWM
VCC
Servo
GND
PWM
VCC
Ultrasonic Sensor
VCC
Trig
ECHO
GND
Ultrasonic Sensor
VCC
Trig
ECHO
GND
Ultrasonic Sensor
VCC
Trig
ECHO
GND
ATtiny85
GND
ADC2/CLKO
ADC3/CLK1
ADC0/Reset
PCINT1
SDA
SCL
VCC
ATtiny85
GND
ADC2/CLKO
ADC3/CLK1
ADC0/Reset
PCINT1
SDA
SCL
VCC
Raspberry Pi
5V
SCL
SDA
Rotary Encoder
Analog
VCC
GND
Rotary Encoder
Analog
VCC
GND

Figure 7-3: Slave integrated schematic

Figure 7-3 is an integrated schematic utilizing the slave and master MCU design. The primary benefit is reduced lines with only 2 lines of data running to the Raspberry Pi, SDA and SCL. These are the i2c lines that will be able to communicate the information from our sensors. We will also power the components via the Raspberry Pi's power pin which is a direct connect to its 5-volt power source. If additional current is needed there are options on using the battery source as the primary supply to the components. We will determine this need via testing.

The final part to consider when implementing the ATtiny microcontroller is the programming steps required. There are publicly available libraries that contain some of the code required to have the device control Servos, to function as an Analog to Digital converter or to have the device read and use Pulse Width Modulation. The ATtiny85 contain 8 pins that are used by the device to control other peripherals. Since the ATtiny is small and only contains 8 pins the same pins are used to program the device. When comparing the pin diagrams, you will notice that the ATtiny has similar GND, VCC, MISO, MOSI pins. On the data sheet for the ATtiny the Serial Clock input is given as well as the Reset input being active low.

1. Pin 1 (ATtiny) Reset	Pin 24 (Raspberry PI)
2. Pin 2 (ATtiny)	Not Used
3. Pin 3 (ATtiny)	Not Used
4. Pin 4 (ATtiny) GND	Pin 06 (Raspberry PI)
5. Pin 5 (ATtiny) MSOI	Pin 19 (Raspberry PI)
6. Pin 6 (ATtiny) MISO	Pin 21 (Raspberry PI)
7. Pin 7 (ATtiny) SCLK	Pin 23 (Raspberry PI)
8. Pin 8 (ATtiny) VCC	Pin 01 (Raspberry PI)

The ATtiny microcontroller utilizes the Serial Peripheral Interface Bus. This interface utilizes a master slave architecture for the purposes of programming the device. The standard utilizes the Master Output, Slave Input and the Master Input, Slave Output as stated above. The Serial Clock is set by the Master MCU for synchronous serial communication. The i2c communication protocol isn't built into the device but the Universal Serial Interface is. With the manufacturer's datasheet the device can easily be programmed to communicate with the i2c protocol via the Two-wire serial interface on the ATtiny. This along with the basic function of the ship will have to be programmed prior to use. We will take this into consideration as it may cause extra work load on the programming portion of our project. If needed, we will redistribute some of the work so that the programming portion is properly dispersed among the group members.

7.4 Final Project Coding Construction

This section contains the overview of the software logic that will be implemented in the final design of the Robotic Surveillance Vehicle project. Each sensor is looked at extensively from a software logic side to be implemented in the firmware in Section 6. The collision detection and avoidance, and computer vision algorithms are also explored in-depth in section 6. Each of this can be looked at individually in section 6. However, implementation and integration of these individual logic flowcharts is needed for the final project in senior design II. All functioning steps of the firmware integration below will be proceeding concurrently with the mobile application development for the Robotic Surveillance Vehicle.

The primary logic to being implementing will be the computer vision to ensure that video feed is being inputted and an accurate target is being acquired for the Robotic Surveillance Vehicle to track. Raspberry Pi Camera integration is required to complete this development. This step will also require development of the mobile application's video feed functionality, the implementation of the Wi-Fi protocol described in section 6 will be used to communicate between the mobile application and the Robotic Surveillance Vehicle. Once this functionality is completed and verified through the performance testing outlined in section 8 phase 1 will be completed.

As you can see in the figure below all three phases are outlined to show the progression of the project as we move through implementation of the final coding.

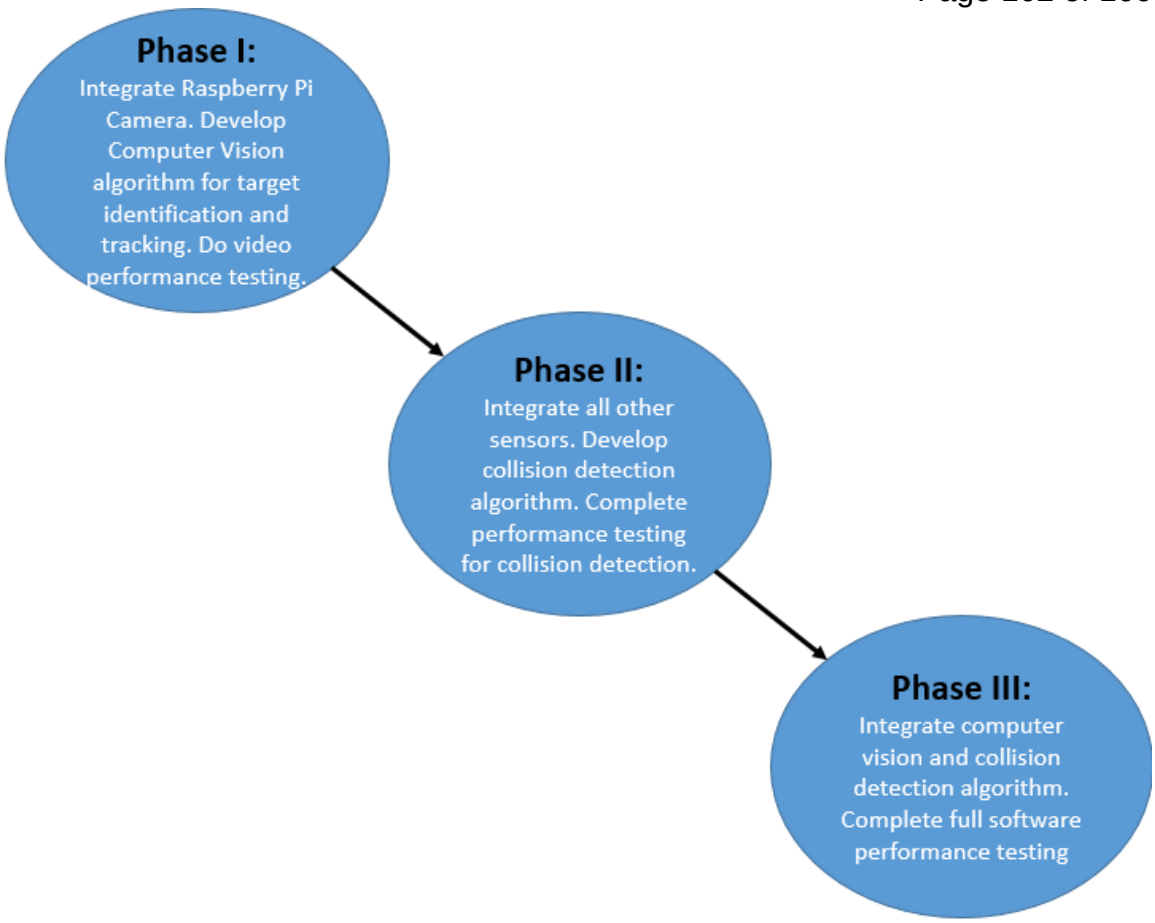


Figure 7-4: Software Phase Diagram

Phase 2 of the software integration will require the use of all other sensors on the device including the motors as well as the collision detection and avoidance algorithm. This algorithm will take the inputs from the sensors to determine a proper path forward for the Robotic Surveillance Vehicle by controlling motor functionality. This will need to work hand and hand with the phase 1 computer vision logic to help the tracking algorithm retain its track according to the requirements specified. The computer vision algorithm will also need to notify the collision detection algorithm if a track is present to begin to follow or if the Robotic Surveillance Vehicle is in an idle mode.

Once both of the phases have been complete final performance testing per section 8 of this document will need to occur verifying all requirements are met, Phase 3. If failures occur the failure management process will be used to help navigate and debug the Robotic Surveillance Vehicle in hopes to create a final product that meets all requirements. The figure below shows the phases of software integration for the final software of the Robotic Surveillance Project.

8. Project Prototype Testing Plan

The project Prototype Testing Plan will primarily focus on testing the hardware and software components of the device. It will determine if the components will meet our required specifications. The following sections will describe the testing environment and how were will proceed with the testing, the hardware specific testing, the software testing and the software testing environment.

8.1 Hardware Test Environment

The hardware testing environment will consist of three primary phases. The first phase will be breadboard and component testing. This will primarily focus on the testing of each individual sensor and their interactions with the MCU. This purpose of the first phase will be to isolate the function of the sensor and test its response. The key here is to look at any variations from expectations and to determine if the variations can be mitigated through code or other methods. We can connect each of the sensors and components to the MCU individually to determine their individual responses. We will also connect the components to an oscilloscope to determine if the pulse width modulation signals are being sent from the components to the MCU. For the i2c lines we can also test that the signals are being sent correctly through the oscilloscope. Since it would be difficult to determine what data is being sent we can at least determine if there is any attenuation in the signals.

The second phase of testing will be the integration of the PCB board testing. At this point in the project we will move to testing the PCB design. Some components will be connected via jumper cables to their individual Breakout Boards while other components will be soldered and attached to the PCB board design. The importance of this step is to determine again if each component is functioning properly. We will also look to see if there are variations in the functionality of the components when connected all together. We will use data collected in first phase of testing to determine if any additional calibrations steps will be needed to be adjusted with the additional equipment added. For example, we know that the weight of the vehicle will change so the calibrations for the servos may need to be adjusted to account for this.

The final phase of testing will be the assembly of the RSV in its final build. The importance here will be to mimic the second phase of testing in a working form. At this point the testing will be focused on troubleshooting. We will need to ensure the vehicle is operating as expected this may involve all of the steps in the other two phases to attempt to isolate any problem areas.

8.1.1 Breadboard component Testing

Below shows the raspberry pi with the GPIO ports connected to for testing. Below is a picture showing the important ports located on the raspberry pi for testing. The GPIO ports will be used to control the sensors. The HDMI will need to be plugged into a monitor and a keyboard will need to be plugged in through the USB ports. The card will be preloaded with the OS and the programming for hardware testing can begin.

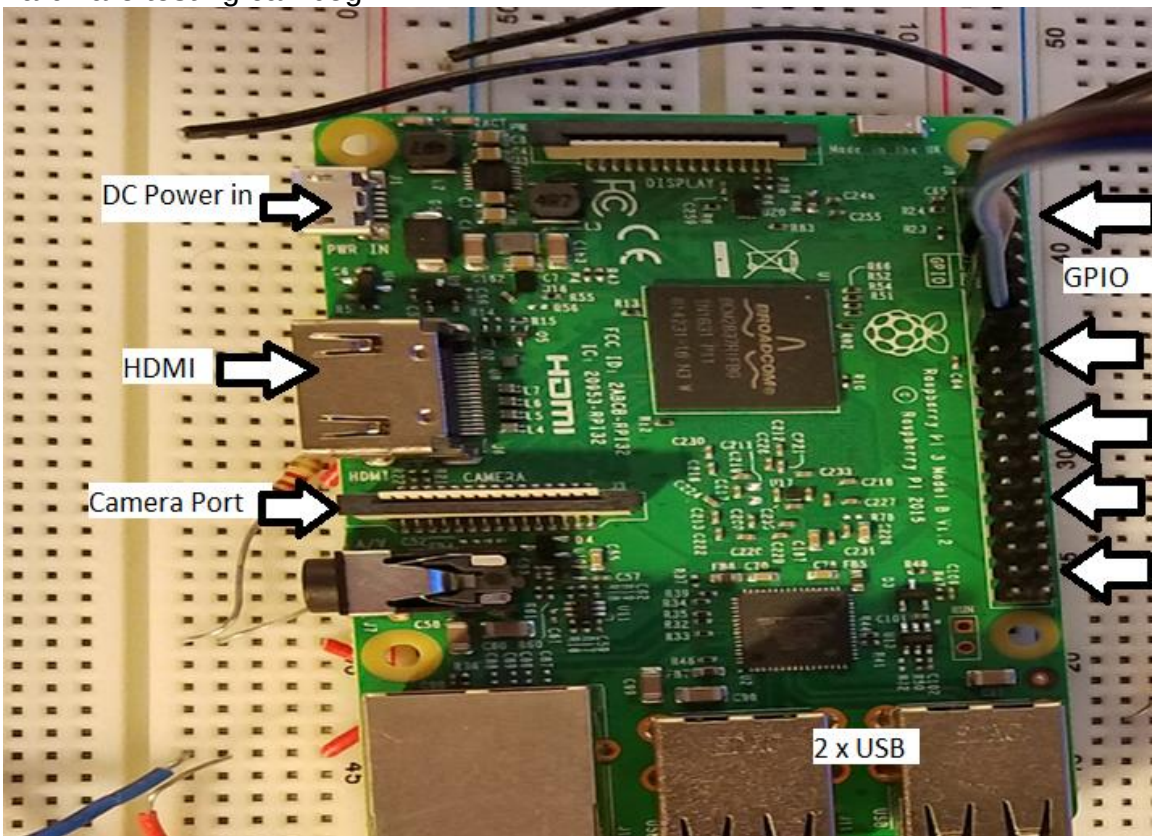


Figure 8-1: Raspberry Pi

Below shows the process to test the ultrasonic sensors. During the breadboard testing phase we will use jumper wires as shown below. By doing this we can set our voltage rails to exactly what is required from the devices. We can also hook the components to the oscilloscope to determine if the information's being across the GPIO is accurate. We will be able to see the pulses and any possible attenuation that may occur from a faulty device.

Figure 8-2: Ultrasonic Sensors

Each of the components will be different the purpose of this section is to show the basic setup used to test each individual component.

8.1.2 PCB testing in lab

In Senior Design 2 we will have completed our PCB design. Once complete we will implement the final circuit implementation in the lab. This will involve deciding which component we will commit to the PCB board and which components we may have to connect with the jumper cables as shown above. Our goal in this process is to place as many of the components as possible on the PCB board. Depending on the method we choose to build in Senior Design 2 we may be able to achieve placing all the components on the PCB board through a compact design using the master and slave MCU method mentioned earlier.

8.1.3 Full component testing in field

While the device will be operational the primary focus will be the smooth functionality of the components. While we may achieve proper functionality of the Servos, Sensor arrays, and the Camera we will still need to determine if the device is properly functioning as expected. We will need to ensure the RSV as a whole is functioning as expected. The Servo movements may need tweaking if we find the movement to be jerky we may also need to increase the power of the battery of the vehicle.

8.2 Hardware Specific Testing

First download the datasheet for the HC-SR04, LSM 30DLHC, Camera V2.1 and the Raspberry Pi. This is used for pin layout identification and comparison of testing values. These layouts with their respective pin numbers and locations are consistently used for all the following sections and is a direct correlation to the pin configuration of the physical device schematic.

8.2.1 Microcontroller

Raspberry PI 3 perform the following steps to calibrate for the testing of the components in the device.

1. Connect the raspberry pi to a 5-volt power source via the micro USB
2. Upload the Raspberry PI OS from the following site
    a. https://www.raspberrypi.org/downloads/
3. Hook up the Raspberry PI to a monitor via the HDMI cable as well as the keyboard through the USB ports
4. Follow the instructions in the software portion to setup the device.
5. Use the below schematic to ensure that that each of the GPIO ports are function properly
6. Connect each port to a breadboard via jumper cables to check the 3.3 and 5-volt power sources.
7. Use a program to turn on each of the GPIO ports to ensure the device is sending the proper voltage to the port. This can be done in conjunction with the testing of the Ultrasonic Sensors
8. Use a program to communicate through the SDA and SCL lines. This will require communication with one of the i2c compliant devices. This can be

done with the testing of the Magnetometer and the Accelerometer.

9. Hook up an oscilloscope to read the SDA and SCL lines. This will ensure that the device is receiving proper signals.

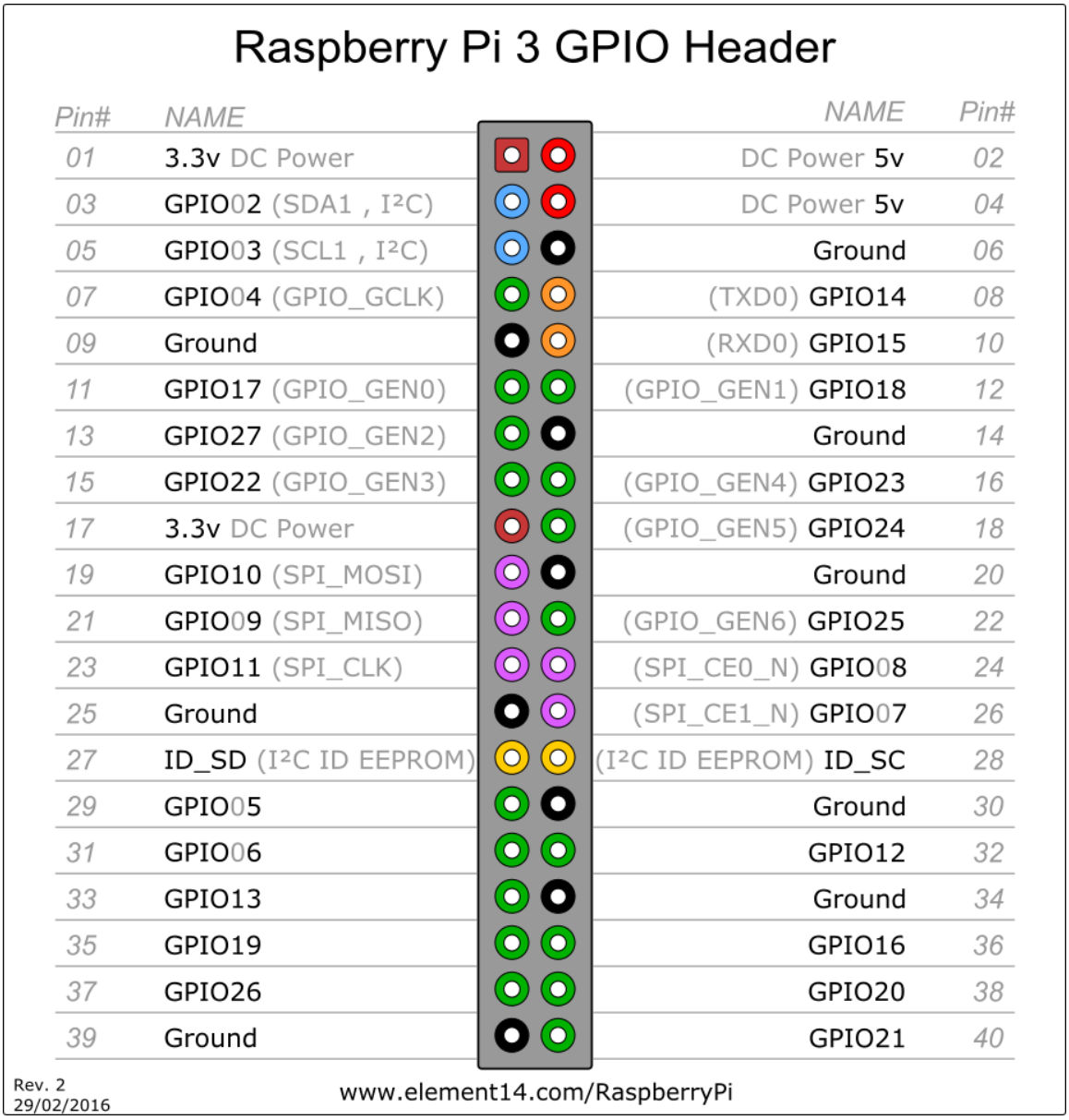


Figure 8-3: Raspberry Pi GPIO Chart [8] (Permission Granted)

8.2.2 Sensors

8.2.2.1 Ultrasonic Sensor

To test the ultrasonic sensor, perform the following steps.

1. Use the pin diagram below.
2. Set the VCC voltage to 5 volts.
3. Attach the Trig pin to a GPIO pin on the MCU
4. Set up a voltage divider to reduce the voltage of the wire leaving the Echo pin. To 3.3 volts from 5 volts
5. Have the reduce voltage output of the Echo pin wired to a different GPIO pin
6. Hook the Trig and Echo pins to the Oscilloscope
7. Load a test program on Raspberry PI and have the device detect an object from the ultrasonic sensor. Place an object within 6 feet and no more than a 15-degree deviation from center of the device.

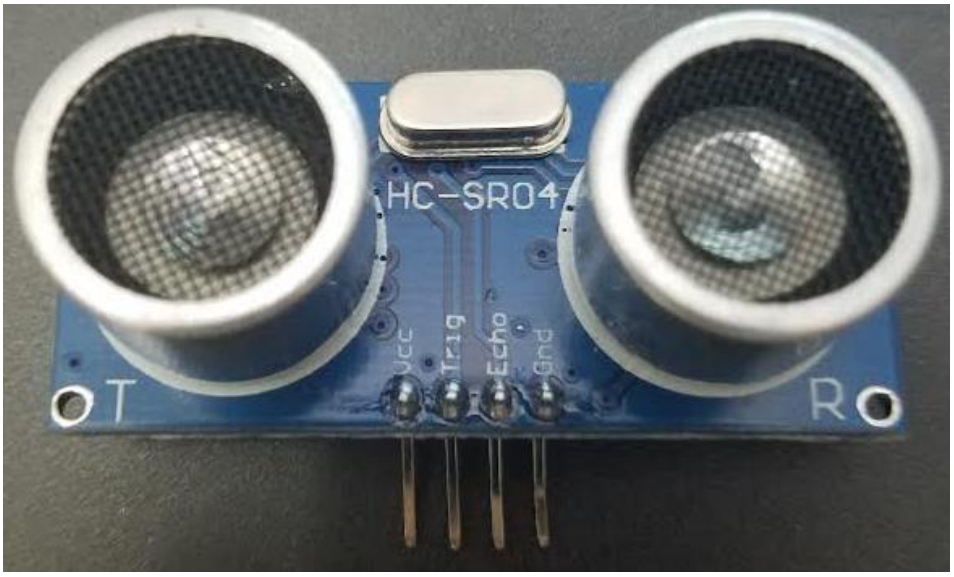


Figure 8-5: Ultrasonic Sensor

Once the functionality of the device has been tested the range in which it can operate needs to be tested. Perform the following steps and record the data that will be relevant for operation of the device. At this point we will not be testing the full functionality of the vehicle in operation we will however be testing the capability of the sensor to determine if the sensor will be adequate for collision avoidance. It is noted that this is performed in a laboratory environment under semi ideal conditions. Any obstacle we place in the path of the device will be recognizable to the ultrasonic sensor.

1. Lay down a measuring stick from 2 cm to 400 cm.
    a. This is the effective range of the device
    b. Move an object from 2 cm all the way to 400 cm and verify the results through the MCU
2. Starting at 2 cm record the effective range of the device.
    a. Record the distance of detection around the device. To do this start at the 2 cm mark and move in a semicircle from the specified

distance in 1 degree increments. This data will be used during the assembly phase of RSV. If an object is in the no detection zone we will need the code to prepare for this so that we stop the vehicle prior to this point.

b. Move out an additional 1 cm and repeat the above step A
c. Repeat step B until you've reached 15 cm.
d. Repeat step A moving starting at 1 distance of 1 foot and move out an additional foot until the device can no longer detect an object
e.

This will conclude the testing of the ultrasonic sensor. This will need to be performed with all three ultrasonic sensors. The detail of the testing doesn't need to be in depth as the results for each should be within a small margin of error if the device is functioning properly.

8.2.2.2 Magnetometer and accelerometer

For the testing of the magnetometer device perform the following steps. Use the below picture for a pin diagram of the device.

1. Connect Vin to the 3.3-volt source on the GPIO pin of the Raspberry pie
2. Load the testing software for the compass and accelerometer
3. Perform the necessary calibration steps to add any adjustments as needed
    a. Display the output in real time
    b. Use another device to determine cardinal directions
    c. Move the device a few times so that it is oriented
    d. Starting at True north turn the device clockwise in 1 degree increments
    e. Use the secondary device to determine the reference
    f. Ideally attach the device together so that as the magnetometer moves so does the testing device
    g. Once the device has been verified use the following site to program the internal calibration
        i. https://learn.adafruit.com/lsm303-accelerometer-slash-compass-breakout/calibration

Figure 8-6: Magnetometer

Once the sensors are connected and tested the device assembly platform can begin. Use a small breadboard such as the one below.

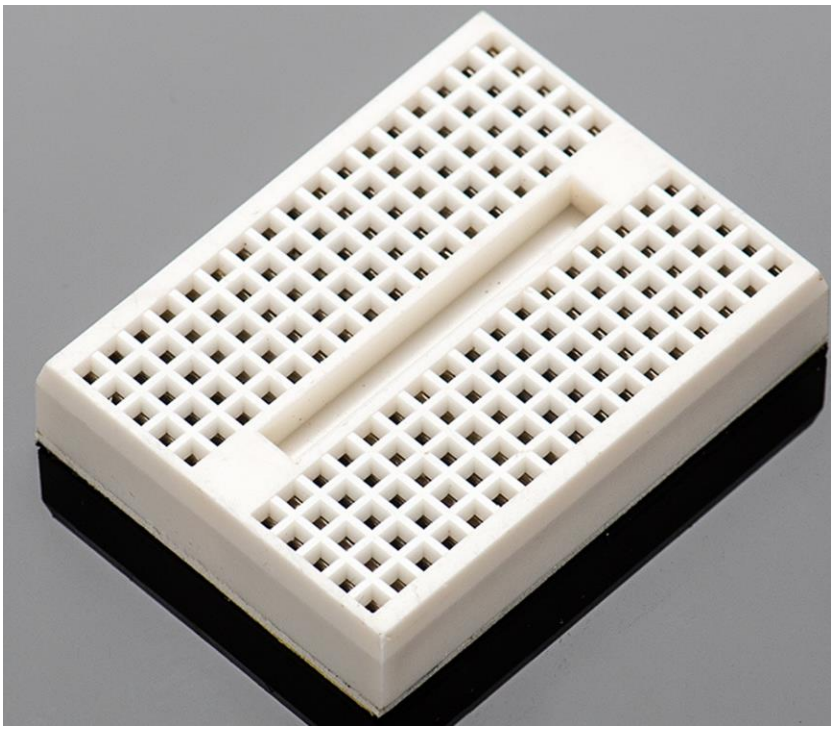


Figure 8-7: Breadboard

The purpose of semi-permanently mounting it to the device is that the sensor arrays will need to be fixed in place on a platform that will be placed above the MCU. We will need precise accelerometer and magnetometer readings to determine how the device moves when in operation. If the mounts are moving this could give false, sensor readings when determining the device's exact positioning and movement.

8.2.2.3 Pi Camera Version 2

Below shows the Pi Camera Version 2. From a hardware perspective there isn't much that can be tested without hooking up the device and running the software programs associated with the device

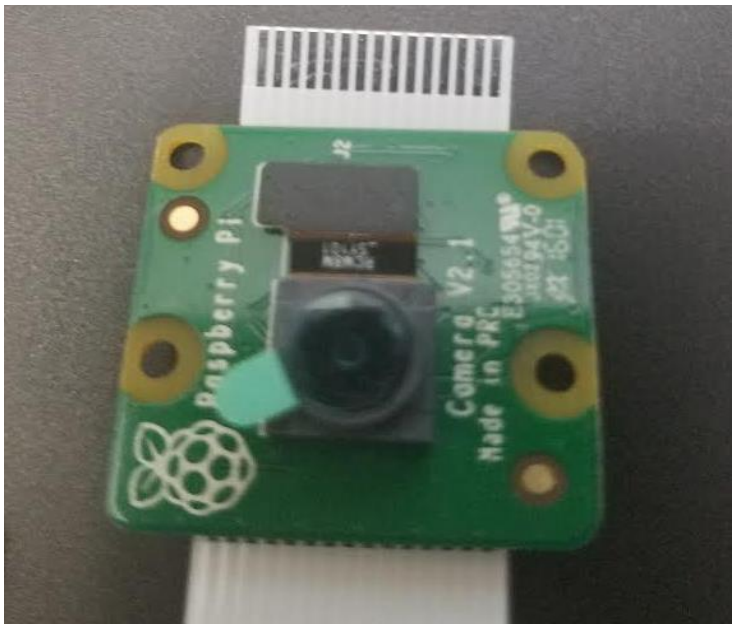


Figure 8-8: Pi Camera Version 2

As you can see in the next picture the camera has a pre-built in port to the Raspberry pi. This will be useful in the testing of the device and hardware troubleshooting. The only issue we will run into is the device mounting of the Camera. We may need to utilize a cable extender to allow for better placement of the camera.

Figure 8-9: Raspberry PI Camera Port

8.2.3 Servos

8.2.3.1Testing Procedure Summary:

To test the motors and the servo drivers the following general steps must be performed.

1. Attach the Raspberry Pi to the servo driver.
    a. Connect the power
        i. For the PCA9685, accomplished by connecting the VCC and GND (left) pins to a 5V and GND pins on the Raspberry Pi.
        ii. The motor power inputs are connected directly from a 5V source and attach to the V+ (top) and GND (top) pins
    b. I2C communication is established between the PCA9685 and Raspberry Pi.
        i. Connect the data line (SDA) to pinout 5.
        ii. Connect the clock line (SCL) to pinout 3.
2. Attach the electric motors to the servo driver.
    a. The positive motor pin of the left motor is connected to the PWM pin of column 0.
    b. The motor ground pin of the left motor is linked to the GND pin of column 0.
    c. Repeat steps 3a and 3b for the right motor and column 1.
3. Calibrate the engine alignment.
    a. Set the rover on 3 meters of two-inch-wide painters tape in a straight line and run it along the length with the PWM set to 10% duty cycle.
    b. If the robot drifts to one direction instead of straight, increase the duty cycle of the engine on that side.
    c. Repeat until the robot drives along the tape without leaving the path
    d. Repeat steps 4 a-c for 10% increments of increasing duty cycle until 90% duty cycle is completed.

8.2.3.2 Detailed Testing Procedure

Movement for the robot will be achieved with two 5V motors controlled by a servo driver. The Adafruit PCA9685 servo driver communicates with the Raspberry Pi using I2C, which is a serial communication bus. The preliminary step in testing the driver and motors involves attaching the components to the proper pins on the Raspberry Pi. The 5V pin 4 on the Raspberry pi should be connected to the Vcc pin on the right side of the PCA9685. Pin 6 which is a ground should be attached to the ground, labeled GND, pin on the right side of the servo driver. These pins provide the power that runs the servo driver, next the power to the motors must be supplied. The following table contains a summary of the pins and their connections.

	Hardware				
	PCA9685	Raspberry Pi	Power Supply	Left Motor	Right Motor
Pins	Vcc	Pinout 4	-	-	-
	V+ (top)	-	5V	-	-
	GND (left)	Pinout 6	-	-	-
	GND (top)	-	GND	-	-
	SCL	Pinout 5	-	-	-
	SDA	Pinout 3	-	-	-
	PWM (0)	-	-	Positive Terminal	-
	PWM (1)	-	-	-	Positive Terminal
	GND (bottom 0)	-	-	Negative Terminal	-
	GND (bottom 1)	-	-	-	Negative Terminal

Table 8-1: Pin Connection Chart

The servo power sources should bypass the Raspberry Pi and directly connect to the power supply. The positive terminal of the battery connects to the V+ terminal on the top side of the PCA9685 next to the red LED labeled “power.” Immediately next to this, a ground for the power supply needs to be affixed to the servo driver. The servos will draw more current, especially under sustained motion and it is beneficial to have a separate energy source for their operation. Excess current can cause the Raspberry Pi to shut off. The motors are noisy and this can adversely impact the operation of the Raspberry Pi and is an additional reason to have a separate motor power supply. Now that the power supplies are connected to the PCA9685, the communication pins are the next to be attached. The schematic diagram below shows the correct configuration for the servo driver attached to the Raspberry Pi and motors.

Information is shared in in I2C communication using two pins that correspond to SDA and SCL. I2C is a master slave protocol and the Raspberry Pi sends commands and the PCA9685 receives this data. SDA stands for the data bus

and communicates the commands and the SCL bus is the clock. On the Raspberry Pi the SCL bus uses pin 3 and the SCL pin is 5. The servo driver is more straightforward, and the SCL line is attached to a pin labeled SCL on the left side of the board. Similarly, the SDA is also directly labeled and can be found directly below the SCL pin. Once these steps have been followed the PCA9685 has been fully connected to the Raspberry Pi and it is time to connect the motors to the servo driver.

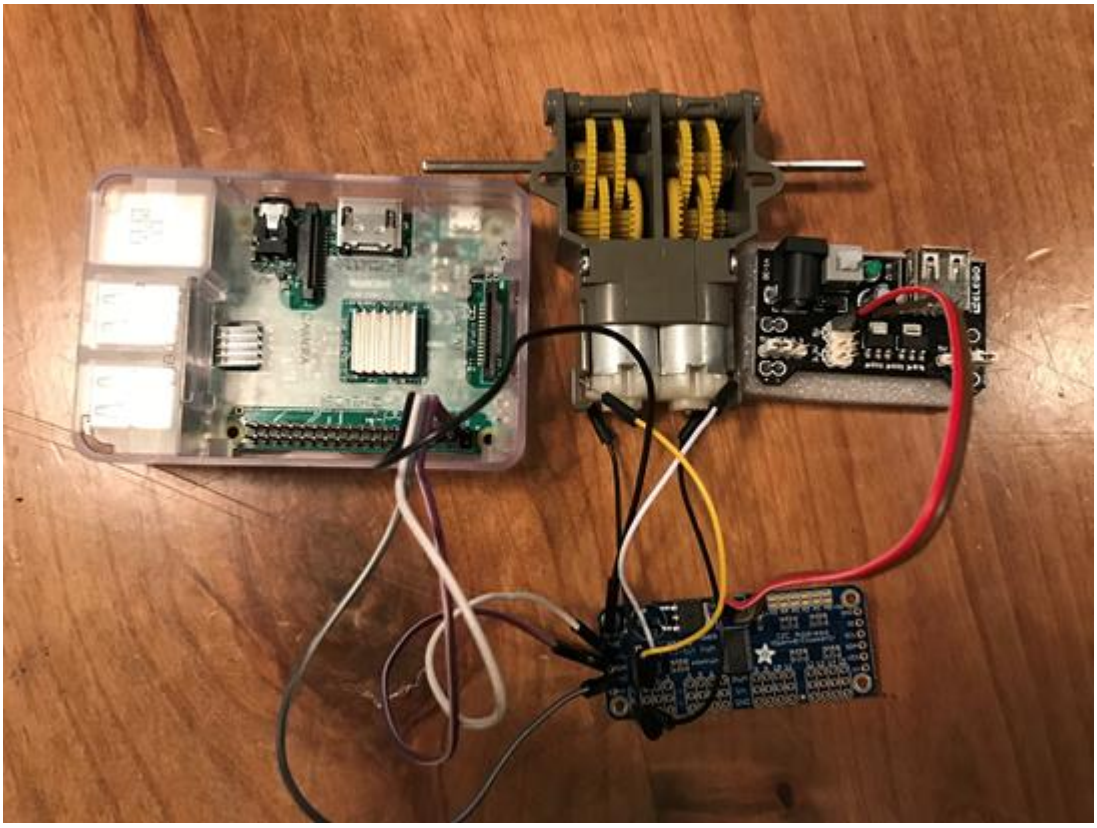


Figure 8-10: Servo Driver Connected to Raspberry Pi, Motor and Power Supply

Two pins are required for each servo for proper operation of the electric motors. The servo driver has units of three pins assembled in columns for each individual motor. The columns run along the bottom edge of the PCA9685 board and from bottom to top they are GND, V+ and PWM. The GND pin should be connected to the ground on the electric motor. The PWM pin represents the pulse wave modulation output, and this pin provides the output that controls the electric motors. The PWM pin should be attached to one of the pins on the electric motor. Setting the frequency of the pulse width modulation determines the speed of the connected motor. The modulation can be set on or off for each channel and this enables individual motors to be controlled. The left motor should be attached with its pins connected to the first column labeled “0.” The right motor in the gear assembly should attach to the column labeled “1.” The labels are

located immediately above the columns and each set of three pins in encased in a white oval. Now that the Raspberry Pi, servo driver and motors are connected, the hardware testing can begin. The following picture shows the PCA9685 attached to the raspberry pi and servo motors.

First the proper connection should be verified and the power supply activated. If the power is attached correctly, the LED should activate on the servo driver board. Once the power is verified, basic commands should be sent to the servos to verify their function. The duty cycle of the pulse width modulation will determine the speed of the motor and for this initial testing should be set to 50%. Additionally, the frequency of the pulse width modulation can be set on a range of 40Hz to 1000Hz and should be set to a value of 500 Hz for initial testing. To verify motor function each motor should be turned on at these configurations. Each should be activated individually and then the two motors should be run simultaneously. By verifying the motor operation at these basic values the initial setup of the Raspberry Pi, PCA9685, and two motors can be verified. Next the motors should be calibrated to ensure proper function.

A likely concern is a differential in motor output. In this scenario, the two motors would receive identical input but provide different rotation. To verify that the motors are functioning properly, the rover should be set to run in a straight line and the robot should be inspected to see if it turns. If the rover turns, the motors are likely providing different output. The robot should be placed on a hard surface with a straight line of tape two inch painters tape extending four meters. the rover would straddle the tape and proceed along its length and if a wheel crosses onto the tape the robot will be considered misaligned. Differential motor output could be accounted for by varying the pulse width modulation of each motor individually. If the rover crosses onto the tape in a direction, the duty cycle of the motor on the side corresponding to the direction the rover moved should be increased. Through this process the rover can be calibrated to travel in an acceptably straight line.

The rover will operate at variable speed and the exact values need to be obtained for each speed configuration. The speed settings will consist of nine values starting at 10% duty cycle, stepping up at 10% increments and ending at 90% duty cycle. At each stage the alignment will need to be verified to ensure that the robot travels in a predictably straight path. Corrections at one duty cycle may not be applicable to different pulse widths and the proper calibrations need to be determined

8.2.4 Wi-Fi Control Interface

The Raspberry PI has a built in Wi-Fi adapter. The hardware will be tested in the software section of this report. Since the module is built into the PCB of the microcontroller we won’t be able to determine if the device malfunctions through

hardware means. In the event of a device malfunction we will implement a backup plan via a Wi-Fi adapter.

8.2.5 Rotary Encoders

8.2.5.1 Testing Procedure Summary:

Installation, calibration and testing of the rotary encoders that communicate vehicle movement information requires the following steps.

1. Connect the Raspberry Pi to the analog to digital converter.
    a. Power
        i. Converter pin 16 (VDD) and pin 15 (VREF) are connected to Raspberry Pi pin 1 (3.3V).
        ii. MCP3008 pin 14 (ground) is attached to GPIO pin 14 (ground)
    b. Communication
        i. Bit Banging Configuration
            1. Converter CLK pin (13) is connected to Raspberry Pi pin 18.
            2. DOUT on the converter (12) is linked to GPIO pin 23.
            3. DIN (11) is connected to the GPIO pin 24.
            4. MCP3008 pin 10 (CS) is connected to the Raspberry Pi pin 25.
            5. The DGND pin (9) is connected to the same ground as the power (14).
        ii. SPI Configuration
            1. Converter CLK pin (13) is connected to Raspberry Pi pin 40 (SCLK).
            2. DOUT on the converter (12) is linked to GPIO pin 35 (MISO).
            3. DIN (11) is connected to the GPIO pin 38 (MOSI).
            4. MCP3008 pin 10 (CS) is connected to the Raspberry Pi pin 24 (CE0).
            5. The DGND pin (9) is connected to the same ground as the power (14).
        iii. Use the SPI configuration if the GPIO pins are available.
2. Connect the rotary encoder to the analog to digital converter.
    a. The VCC pin for the encoders are connected to the VDD (15) of the analog to digital converter.
    b. The grounds of both rotary encoders are attached to the ground pin (14) of the converter.
    c. The left encoder communication pin is attached to analog to digital converter channel 0 (pin 1).
    d. The right encoder communication pin is attached to analog to digital converter channel 1 (pin 2).

3. Rotary encoder calibration.
    a. Let the robot move a known distance and count the output pulses from the rotary encoders.
    b. Divide the distance traveled by the number of pulses to determine the distance per output.
    c. Repeat 10 time and average results to determine calibrated distance per encoder output pulse.

8.2.5.2 Detailed Testing Procedure

The implementation of the two rotary encoders is complicated because they only produce analog output. Raspberry Pi devices do not have any analog input pins and to utilize the data, it must be converted to a digital format. A simple solution to this issue is to use an analog to digital converter, in this case the Adafruit MCP3008. The rotary encoders must be connected to an analog to the MCP3008 digital converter which is connected to the Raspberry Pi GPIO. The general schematic for this design is presented in the following diagram.

	Hardware				
	MCP3008	Rasperry Pi (Bit Banging)	Raspberry Pi (SPI)	Left Rotary Encoder	Right Rotary Encoder
Pins	VDD (16)	3.3V (1)	3.3V (1)	VCC	VCC
	VREF (15)	3.3V (1)	3.3V (1)	VCC	VCC
	GND (14)	GND (14)	GND (14)	GND	GND
	CLK (13)	(18)	SCLK (40)	-	-
	DOUT (12)	(23)	MISO (35)	-	-
	DIN (11)	(24)	MOSI (38)	-	-
	CS (10)	(25)	CE0 (24)	-	-
	DGND (9)	GND (14)	GND (14)	GND	GND
	CH0 (1)	-	-	Output Pin	-
	CH1 (2)	-	-	-	Output Pin

Table 8-2: Pin Connection Chart

The MCP3008 must be attached to a number of Raspberry Pi GPIO pins in order to function properly. The analog to digital converter draws its power from the Raspberry Pi and the VDD pin, number 16 on the MCP3008, needs to be connected to GPIO pin 1 to draw 3.3V. The VREF pin on the MCP3008 also needs to be connected to the same pin as VDD and they should be linked together. Next the converter should be grounded by attaching the fourteenth pin on the MCP3008 to a ground pinout on the Raspberry Pi, in this case number fourteen. Now that the power pins are connected, the communication output needs to be attached.

There are two possible configurations for the output from the converter to the Raspberry Pi. The first implementation uses bit banging and can communicate with any pin on the GPIO. This is a flexible option and allows for a more adaptable design because, by using any available pin, it enables the creation of new configurations with minimal difficulty. If a new component is needed and requires any of the pins being currently utilized for communication, the alteration could be achieved without disrupting the current design. The bit banging configuration is implemented by attaching the CLK pin on the converter, number thirteen, to GPIO pin 18. The DOUT pin, number 12 on the MCP3008, is attached to the GPIO pin number 23 and the DIN, number 11, is attached to Raspberry Pi pin 24. The CS pin on the converter, number 10, is attached to the GPIO pin 25 and finally the DGND pin, number 9, is attached to a GPIO pin ground, in this case the same one as the converter pin number 14 for the power ground. The primary drawback of this configuration is that the bit banging is slower than the dedicated hardware communication pins. Though this design is more flexible, the robot does not have an excessive number of peripherals. Since the faster configuration is available it should be used in this design.

The second configuration, and the one to be used in this design, is the specific hardware Serial Peripheral Interface (SPI) of the Raspberry Pi. The power is connected the same way as the bit banging, as well as both grounds. MCP3008 pin 13, CLK is attached to GPIO pin SCLK, number 40, and DOUT, pin number 12, is attached to the Raspberry Pi pin for MISO, number 35. Converter pin 11, DIN, is attached to the Raspberry Pi pin 38, which corresponds to MOSI. Finally, the pin for CS on the converter, number 10, attaches to the GPIO pin for CE0, which corresponds to GPIO 24. The increased speed of this configuration makes it more desirable for our purposes. The primary benefit of the previous design was the ability to use any pin for communication. Since there is no shortage of pins in the project design, this benefit does not offset the decreased speed of the bit banging configuration. For these reasons the hardware SPI is the superior choice for this design.

The encoders now need to be attached to the assembled analog to digital

converter. The encoders have three pins each, a power pin, a ground and a communication pin. The ground can be connected to one of the grounds for the MCP3008. Similarly, the power pins can be attached to the 3.3V source pins for the converter. Finally, the communication pin needs to be connected to one of the input pins of the MCP3008. Any of the eight input pins would work but to stay consistent with the convention used for the motors, the left rotary encoder should be connected to CH0, pin 1, and the right should be connected to CH1, pin 2. Once the two encoders are assembled in this way, their hardware is fully implemented.

The rotary encoders now need to be tested and calibrated. The encoders send information to the Raspberry Pi which tells the computer how many times the axle has rotated. The specific encoders used in this design send 16 pulses per rotation. This information does not directly tell the computer how far the robot has traveled. The rate of conversion needs to be determined, so that the distance can be gathered from rotary encoder information. A simple method of determining the amount of distance traveled per packet sent from the encoder would be to have the robot straddle a meter stick. The rover would be arranged so that the tip of the chassis is above the beginning of the stick. The rover would be told to travel forward and told to stop as it nears the end. The exact distance traveled would be measured and this distance would be divided by the number of data points sent by the encoders. If this process is done ten times and the amount of distance per data point is averaged, a reasonable rate of distance per pulse from the encoders can be found.

8.3 Software Testing

This section will contain an overview of all testing that will be completed on our software as well as the objectives to be met in each of the testing to follow. This will include an overview on how to conduct this testing.

8.3.1 Communication Testing

This section will contain the test procedures required to ensure each module is communicating. Communicating for the purposes of these tests is defined as the ability for the microcontroller to send commands to the module being tested and for the module to respond in a trackable way such as with an LED flashing or sending data back to the microcontroller. Each subsection of this section must be completed before continuing on with integration of the module into the complete system. Such that section 8.3.1.1 Digital Compass Communication Testing must be completed before integrating the digital compass module into the Robotic Surveillance Vehicle system. Preferably all communication testing will be completed before any of the modules are integrated however this is unlikely due to timing of when parts are to be obtained as well as time to develop the Robotic Surveillance Vehicle system.

8.3.1.1 Digital Compass Communication Testing

The objective of this section is to confirm that the digital compass module is correctly communicating with the microcontroller as defined in section 8.3.1 Communication Testing. This will confirm that no part of the digital compass module or microcontroller are defective and that the digital compass module is properly installed.

1. Connect the digital compass module to the Raspberry Pi microcontroller.
2. Load the Digital Compass Communication Testing software from Appendix C-Datasheets into the Raspberry Pi microcontroller.
3. Run the loaded software and confirm that the normalized X, Y, and Z axis values are printed to the screen. Record the results and a Pass/Fail in the Software Communication Testing Data Sheets.
4. If a failure occurs follow the failure instruction steps outlined in section 8.3.1.8 Communication Failure Management.

8.3.1.2 Ultrasonic Sensors Communication Testing

The objective of this section is to confirm that the ultrasonic sensors are correctly communicating with the microcontroller as defined in section 8.3.1 Communication Testing. This will confirm that no part of the ultrasonic sensor or microcontroller are defective and that the ultrasonic sensor is properly installed.

1. Connect the ultrasonic sensor to the Raspberry Pi microcontroller.
2. Load the Ultrasonic Sensors Communication Testing software from Appendix C-Datasheets into the Raspberry Pi microcontroller.
3. Run the loaded software and confirm that the ranging and timing values are printed to the screen. Record the results and a Pass/Fail in the Software Communication Testing Data Sheets.
4. If a failure occurs follow the failure instruction steps outlined in section 8.3.1.8 Communication Failure Management.

NOTE: The ultrasonic sensor communication testing code in Appendix C-Datasheets is given to test a single ultrasonic sensor additional commented code is added and highlighted in yellow to allow this to test all three ultrasonic sensor communications at once. Just uncomment the highlighted sections of the code then follow the steps above.

8.3.1.3 Servos/Servo Controller Communication Testing

The objective of this section is to confirm that the servo controller and servos are correctly communicating with the microcontroller as defined in section 8.3.1 Communication Testing. This will confirm that no part of the servo controller, servos, or microcontroller are defective and that the ultrasonic sensor is properly installed.

1. Connect the servo controller to the Raspberry Pi microcontroller.
2. Connect the two servos to the servo controller.
3. Load the servo controller communication testing software from Appendix C-Datasheet into the Raspberry Pi microcontroller.
4. Run the loaded software and confirm that the heartbeat LED is blinking. Record the results and a Pass/Fail in the Software Communication Testing Data Sheets.
5. Confirm that the Error Byte is not showing an error which is a value from 0-2. Record the result and a Pass/Fail in the Software Communication Testing Data Sheet.
6. Confirm that the Servos were able to drive forward, backward then stop. Record the result and a Pass/Fail in the Software Communication Testing Data Sheet.
7. If a failure occurs follow the failure instruction steps outlined in section 8.3.1.8 Communication Failure Management.

8.3.1.4 Camera Communication Testing

The objective of this section is to confirm that the Raspberry Pi Camera is correctly communicating with the microcontroller as defined in section 8.3.1 Communication Testing. This will confirm that no part of the Raspberry Pi Camera or microcontroller are defective and that the Raspberry Pi Camera is properly installed.

1. Connect the Raspberry Pi Camera to the Raspberry Pi microcontroller.
2. Load the camera communication testing software from Appendix C-Datasheet into the Raspberry Pi microcontroller.
3. Connect the Raspberry Pi microcontroller to a monitor
4. Run the loaded software and confirm that the video preview is displayed to the connected monitor. Record the results and a Pass/Fail in the Software Communication Testing Data Sheets.
5. If a failure occurs follow the failure instruction steps outlined in section 8.3.1.8 Communication Failure Management.

8.3.1.6 Wi-Fi Module Communication Testing

The objective of this section is to confirm that the Raspberry Pi Wi-Fi module is correctly communicating with the smart device as defined in section 8.3.1 Communication Testing. This will confirm that no part of the Raspberry Pi Wi-Fi module is defective as well as proper communication between the microcontroller and the smart device has been implemented.

1. Load the Wi-Fi Module Communication Testing software from Appendix C-Datasheet into the Raspberry Pi microcontroller.
2. Connect the Raspberry Pi microcontroller and smart device to the same

Wi-Fi network.
3. Run the loaded software and confirm that the data to be transferred is displayed to the smart device and back to the microcontroller. Record the results and a Pass/Fail in the Software Communication Testing Data Sheets.
4. If a failure occurs follow the failure instruction steps outlined in section 8.3.1.8 Communication Failure Management.

8.3.1.7 Communicating to Modules Simultaneously

The objective of this section is to confirm that the Raspberry Pi microcontroller is correctly communicating with the multiple sensors connected as defined in section 8.3.1 Communication Testing. This will confirm that all the sensors connected to the Raspberry Pi microcontroller can communicate properly to the Raspberry Pi microcontroller during the same session.

1. Connect all sensors to the Raspberry Pi microcontroller.
2. Load the Simultaneous Communication Testing software from Appendix C-Datasheet into the Raspberry Pi microcontroller.
3. Connect the Raspberry Pi microcontroller to a monitor
4. Run the loaded software and confirm that the video preview is displayed to the connected monitor. Record the results and a Pass/Fail in the Software Communication Testing Data Sheets.
5. Confirm that the normalized X, Y, and Z axis values are printed to the screen. Record the results and a Pass/Fail in the Software Communication Testing Data Sheets.
6. Confirm that the ranging and timing values are printed to the screen. Record the results and a Pass/Fail in the Software Communication Testing Data Sheets.
7. Confirm that the heartbeat LED of the servo controller is blinking. Record the results and a Pass/Fail in the Software Communication Testing Data Sheets.
8. Confirm that the Error Byte is not showing an error which is a value from 0-2. Record the result and a Pass/Fail in the Software Communication Testing Data Sheet.
9. Confirm that the Servos were able to drive forward, backward then stop. Record the result and a Pass/Fail in the Software Communication Testing Data Sheet.
10. Confirm that the data to be transferred is displayed to the smart device and back to the microcontroller. Record the results and a Pass/Fail in the Software Communication Testing Data Sheets.
11. If a failure occurs follow the failure instruction steps outlined in section 8.3.1.8 Communication Failure Management.

8.3.1.8 Communication Failure Management

This section outlines the procedure to be followed when a failure occurs during one of the communication testing procedures. The objective of this section is to resolve communication testing failures quickly and efficiently. The flowchart below represents the failure management process that will be used during integration and testing of the Robotic Surveillance Vehicle.

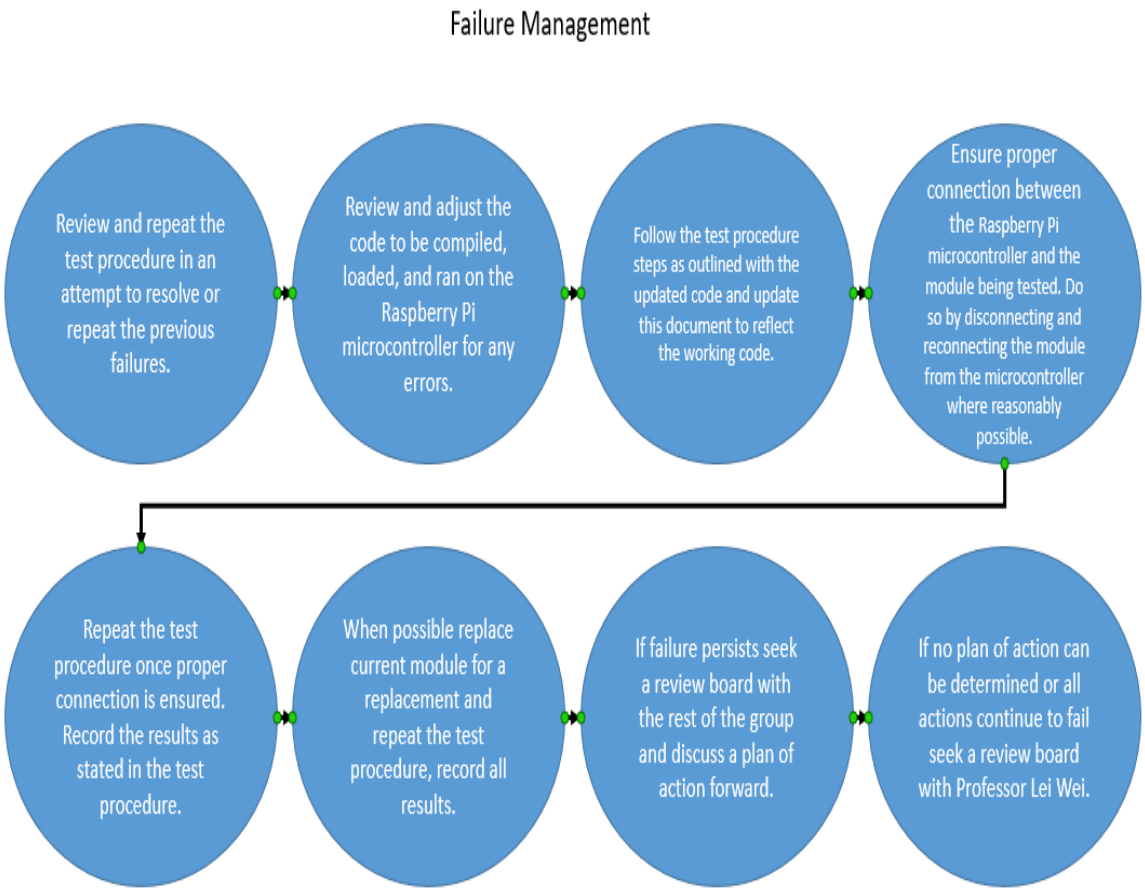


Figure 8-11: Failure Management Diagram

1. Review and repeat the test procedure in an attempt to resolve or repeat the previous failures. Record the result in the Communication Failure Management Sheet.
2. Review and adjust the code to be compiled, loaded, and ran on the Raspberry Pi microcontroller for any errors.
3. Follow the test procedure steps as outlined with the updated code and update this document to reflect the working code.
4. Ensure proper connection between the Raspberry Pi microcontroller and the module being tested. Do so by disconnecting and reconnecting the module from the microcontroller where reasonably possible.

5. Repeat the test procedure once proper connection is ensured. Record the results as stated in the test procedure.
6. When possible replace current module for a replacement and repeat the test procedure, record all results.
7. If failure persists, seek a review board with the rest of the group and discuss a plan of action forward
8. If no plan of action can be determined or all actions continue to fail seek a review board with Professor Lei Wei.

8.3.2 Performance Testing

This section will be used to define the testing required to determine whether performance based requirements will be met by the Robotic Surveillance Vehicle. These tests will be developed based on the requirement being tested. Each of the following sections will define tests for each software related requirement.

8.3.2.1 Voice Command Testing

This section is to verify the software requirement below:

- The smart device will receive voice commands as input.
- The Robotic Surveillance Vehicle shall receive voice commands including: survey, track, find, and stop.
- The Robotic Surveillance Vehicle shall have a response delay of less than three seconds from the user giving the command to the vehicle responding.
- The Robotic Surveillance Vehicle shall have a maximum range of at least 25ft.

This requirement is to ensure commands can be received by the mobile application on the smart device to send to the Robotic Surveillance Vehicle setting the current mode of the Robotic Surveillance Vehicle.

1. Open mobile application on smart device.
2. Navigate to the Voice Commands page of the mobile application.
3. Say one of the following commands: Survey, Track, Stop.
4. Verify the mobile application acknowledges the command and echoes it on screen. Record the results and any pass/fail in Appendix C - Datasheets for Performance testing voice command.
5. (Optional) Verify a response and if the appropriate response by the Robotic Surveillance Vehicle is observed. Record the results and any pass/fail in Appendix C -Datasheets for Performance testing voice command.

8.3.2.2 Verify Data transmission over Wi-Fi

This section contains the test procedures to verify the software requirement below:

- The software shall command the vehicle through Wi-Fi based on the input command.
- The Robotic Surveillance Vehicle shall accept functional commands over Wi-Fi.
- The smart device shall communicate to the Robotic Surveillance Vehicle wirelessly.
- The Robotic Surveillance Vehicle shall have a maximum range of at least 25ft.

Wi-Fi communication is the primary and only form of communication between the Robotic Surveillance Vehicle and the mobile application. Verifying that the vehicle can both send and receive information from the mobile application for various situations.

1. Establish Wi-Fi connection between the Raspberry Pi microcontroller and test environment on the computer.
2. Open the test software as found in Appendix C - Datasheets Performance Wi-Fi transmission
3. Verify the print command is echoed on the computer screen and there is acknowledgement on the Raspberry Pi for both sending and receiving. Record the results and any pass/fail in Appendix C -Datasheets for Performance testing.

8.3.2.3 Verify Tracking or Reacquire determination.

This section contains the test procedures to verify the software requirement below:

- The software shall determine whether a track is being held or if the vehicle needs to reacquire.

This requirement is to understand that the vehicle is aware of the current tracking position it is in whether that be having seen no point of interest, identified a point of interests, marked a point of interest, tracking a point of interest, or lastly having lost the track. This is important to set up what the Robotic Surveillance vehicle should be doing such as, attempting to reacquire the track, continue autonomous movement to keep the track, or wait for a command from the user.

1. Open and load the test software as found in Appendix C - Datasheets Performance Tracking or Reacquire.

2. Run the software and verify the tracking and reacquiring determination is echoed to the screen in the appropriate order. Record the results and any pass/fail in Appendix C -Datasheets for Performance testing.

8.3.2.4 Verify Collision Detection and Avoidance

This section contains the test procedures to verify the software requirement below:

- The software shall prevent the Robotic Surveillance Vehicle from colliding with objects.

This requirement is to verify the Robotic Surveillance Vehicle's collision detection and avoidance algorithm. This is one of the most integral parts of the software design and will require thorough testing beyond this section. This is to verify that the Robotic Surveillance Vehicle will be able to run autonomously during the tracking.

1. Open and load the test software as found in Appendix C - Datasheets.
2. Run the software and verify the collision algorithm by setting up a track.
3. Observe the Robotic Surveillance Vehicle during the 30 second track.
4. Verify no collisions occur during the track and record your observations in the Data sheet as well as the Pass/Fail.
5. Make comments on the Data sheets relevant to the observations for future integration and testing if a fail occurred, or improvements even if a pass was observed.

8.3.2.5 Verify Point of Interest Identification

This section contains the test procedures to verify the software requirement below:

- The software shall identify a predetermined point of interest (POI).
- The Robotic Surveillance Vehicle shall be able to find a point of interesting that is 5ft away from the vehicle.

This requirement is to confirm that the camera is working with the computer vision algorithm to ensure that the Robotic Surveillance Vehicle can determine edges, shapes, and colors. This will all be used to "identify" a point of interest as stated in the requirement. Points of interest can be marked through the surveillance mode that the Robotic Surveillance Vehicle has to specifically search for and identify points of interest.

1. Open and load the test software as found in Appendix C - Datasheets.
2. Run the software and verify the computer vision software is displaying the

video feed.
3. Verify that the target is identified by the computer vision software, record the results in the Appendix C- Datasheets and whether the test was a pass/fail.
4. Measure and verify the target is at least 5ft from the Raspberry Pi camera when the track is acquired after identifying. Record the results in Appendix C-Datasheets and whether the test was pass/fail.

8.3.2.6 Verify Tracking.

This section contains the test procedures to verify the software requirement below:

- The software shall allow the Robotic Surveillance Vehicle to track a point of interest.
- The Robotic Surveillance Vehicle shall hold a track on a point of interest for at least 30 seconds.
- The Robotic Surveillance Vehicle shall provide imaging back to the user from the vehicle's camera.

This requirement requires that section 8.3.2.5 has been verified as the Robotic Surveillance Vehicle will have to identify and mark a point of interest before being able to track. This is in hopes that there will be a lower likelihood that a misidentification will occur. This tracking will also be verified through a duration of holding and following the point of interest through the track.

1. Open and load the test software as found in Appendix C - Datasheets Performance Tracking or Reacquire.
2. Run the software.
3. Initiate survey mode and verify a target is marked.
4. Initiate tracking mode and verify the Robotic Surveillance Vehicle is tracking the target. Record the results in Appendix C-Datasheets and Pass/Fail.
5. Once track is initiated time the duration of the track. Record the time and Pass/Fail in Appendix C-Datasheet, verify that the time is greater than 30 seconds.
6. Open the mobile application for the Robotic Surveillance Vehicle.
7. Go to the Robotic Surveillance Vehicle Video Feed page on the Mobile application.
8. Verify the video is being received by the mobile application record the results and pass/fail in Appendix C-Datasheets.

8.3.2.7 Performance Failure Management

This section outlines the procedure to be followed when a failure occurs during one of the performance testing procedures. The objective of this section is to resolve performance testing failures quickly and efficiently. The flowchart below represents the failure management process that will be used during integration and testing of the Robotic Surveillance Vehicle. It is important to follow the correct performance failure management process as this will help mitigate the issues quickly and identify where the issue is coming from. Even though this is the software portion of the testing environment it is still possible to have hardware failures at any point during the operation of the vehicle.

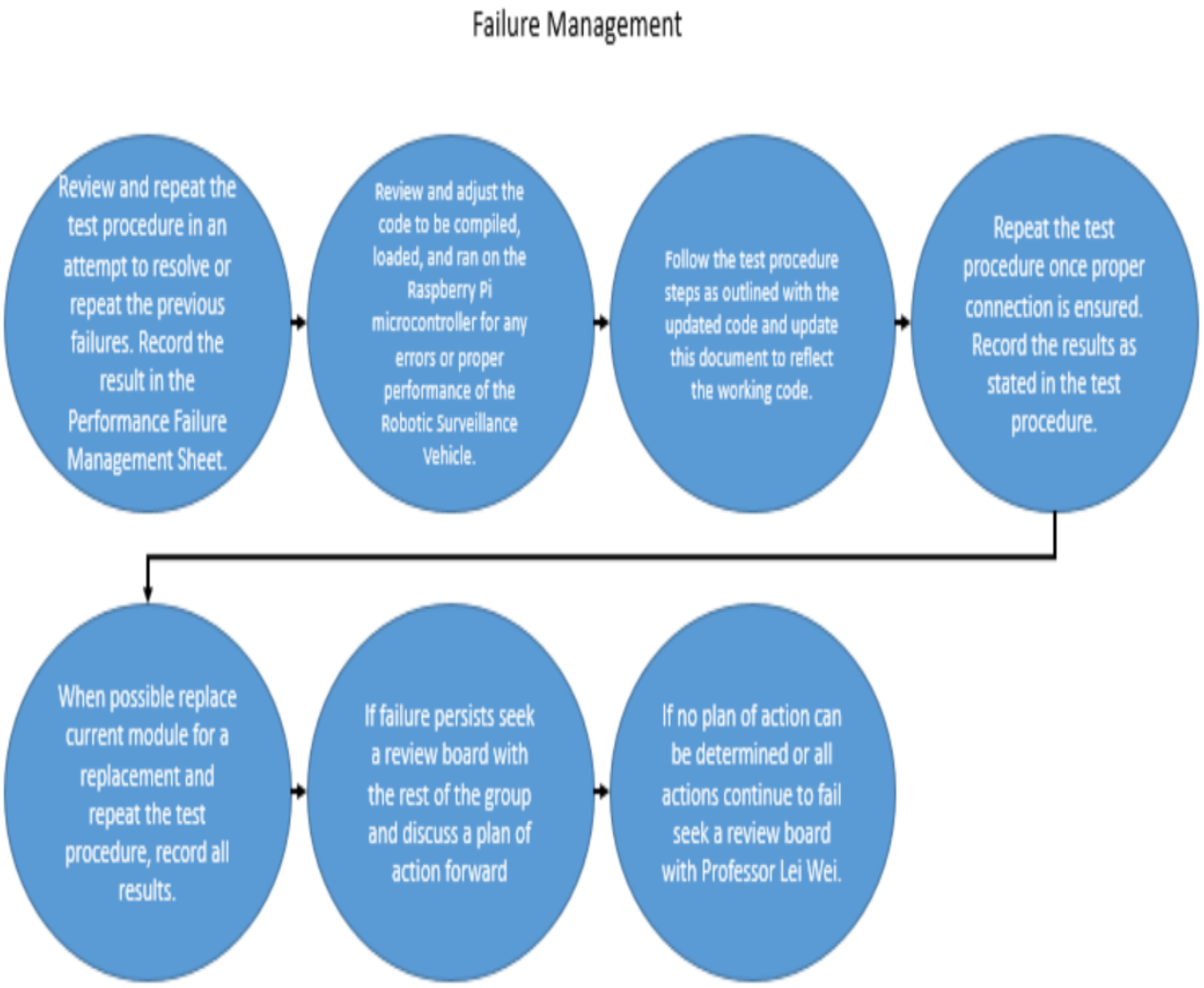


Figure 8-12: Performance Failure Management Diagram

1. Review and repeat the test procedure in an attempt to resolve or repeat the previous failures. Record the result in the Performance Failure

Management Sheet.
2. Review and adjust the code to be compiled, loaded, and ran on the Raspberry Pi microcontroller for any errors or proper performance of the Robotic Surveillance Vehicle.
3. Follow the test procedure steps as outlined with the updated code and update this document to reflect the working code.
4. Repeat the test procedure once proper connection is ensured. Record the results as stated in the test procedure.
5. When possible replace current module for a replacement and repeat the test procedure, record all results.
6. If failure persists, seek a review board with the rest of the group and discuss a plan of action forward
7. If no plan of action can be determined or all actions continue to fail seek a review board with Professor Lei Wei.

8.4 Software Test Environment

This section will contain an overview of the environments that the software will see during testing with analysis on how that will change or evolve when implemented in the final product. There will be a few major test environments for the software, these being the computer where the coding and some debugging will be taken place, the smart device which the mobile application will run on, the Raspberry Pi microcontroller with various sensors connected at any time, and lastly the final Robotic Surveillance Vehicle with everything implemented for debugging.

8.4.1 Desktop Environment

The first line to develop, test, and debug the software will be working on the computer and with the Raspberry Pi microcontroller in the initial development where the base algorithms, syntax, and compilation errors will be caught before moving to integration. Once this has been completed integration and communication to the various sensors will be needed. This will change the environment by introducing single sensors at a time to the Raspberry Pi microcontroller which will eventually move into multiple, then all the sensors integrated and communicating through the Raspberry Pi microcontroller.

8.4.2 Mobile Application Environment

On the more software and less firmware side of the Robotic Surveillance Vehicle project is the mobile application which is used to communication commands and data to and from the Robotic Surveillance Vehicle from a smart device. As in the firmware compilation and syntax errors will be caught at a computer level before implementation on an actual smart device. This is due to the explosion of mobile application over the past decade allowing for integrated development environments or IDEs that have environments to simulate mobile applications on smart devices for debugging. This system will be used as the primary testing

environment for the mobile application associated with the Robotic Surveillance Vehicle.

8.4.3 Robotic Surveillance Vehicle Environment

Lastly will be the Robotic Surveillance Vehicle as a whole to act as a software testing environment. This will be the last testing environment that the software sees, and most issues will hopefully be debugged prior to this step. However, I am expecting this to be a primary software testing environment due to the unseen bugs that will arise from the full Robotic Surveillance Vehicle being integrated. This environment will consist of testing the actual adjustments motor and mechanical made during real world interaction based on the computer vision and collision detection and avoidance algorithms. This will be to develop the success of the performance based requirements as the previous test environments have been focused on the proper working of the parts and the system as the Robotic Surveillance Vehicle is integrated. This environment is assuming all of this has been checked and completed prior to full integration and performance testing.

9. Administrative Content

9.1 Bill of materials

In an effort to see how our requirements conflict with our budget Table 9-1 was developed to see possible parts we will need and the costs associated with them. This will be taken into consideration along with power consumption, size and other aspects of the part that will help us to meet our requirements. The minimum and maximum costs are listed in the table, reflecting best and worst case scenarios.

In order to meet the cost requirement, the team will attempt to get components closest to the minimum end of the cost estimate. This parts estimate is subject to change during the course of creating the project, as extra parts may be needed or parts listed may no longer be needed if an alternate choice is made. Regardless of changes, the team will reduce the cost to under the requirement of $450. All members will keep a record of expenditures towards the project and split the cost of the project once it is completed

	Cost			
Part	Low	High	Qty	Average Total Cost
PCB	$50.00	$100.00	1	$75.00
GPS Chip	$20.00	$30.00	1	$25.00
Wifi Chip	$4.00	$20.00	1	$12.00
RF or Bluetooth Transmitter/Reciever	$5.00	$20.00	1	$12.50
Robotic Vehicle Kit	$20.00	$300.00	1	$160.00
Battery	$10.00	$20.00	2	$30.00
microphone	$2.00	$10.00	1	$6.00
Transformer	$10.00	$20.00	1	$15.00
Microcontroller	$40.00	$80.00	1	$60.00
Ultrasonic Sensors	$3.00	$30.00	3	$49.50
Camera	$25.00	$40.00	1	$32.50
Digital Compass	$3.00	$25.00	1	$14.00
Totals	$192.00	$695.00		$491.50

Table 9-1: Budget Table

9.2 Milestones

9.2.1 Semester One

1) Initial Documentation 10-page overview of project motivation, goals, function, specification, and cost.
2) Project research: similar products, relevant technologies, possible part and components, PCB possible designs and architectures.
3) Project standards, and realities, and design constraints.
4) PCB Design, part research, and part identification. This will add specifics to which parts we chose and the motivations for choosing those parts.
5) Software Design and identification, specifics of what algorithms will be developed, and how they will be implemented.
6) Master plan including building, testing, and evaluation.
7) Necessary/used test equipment and facilities.
8) Receive and test parts to ensure quality of the ordered parts before assembly.
9) Project User/Owner's Manual.
10) Final/Updated administrative details, man hours completed/needed, part and design costs, updated timeline and milestones.
11) Conclusion, references, and appendices.
12) Finished deliverable 90-120 page.

9.2.2 Semester Two

1) PCB build and prototyping.
2) Software Version 1 build, includes required functionality.
3) PCB testing, evaluate metrics of prototyped system compared to expected metrics.
4) Software metric evaluation and testing, not over processing, within designated tolerances.
5) Mid semester demo/evaluation, functional prototype limited in overall performance meet some but not all requirements.
6) Reevaluate a path forward to improve project accuracy and efficiency and complete tasks.
7) Implement additional functionality that is not required for baseline functionality.
8) Complete testing and perform corrective measures.
9) Final working demonstration meeting specified requirements.

The overall design path shown in Figure 9-1 depicts our planned general design process, which spans both semesters of Senior Design. This process will be used to keep the team organized and follow a set path for designing and implementing our product. This design process mimics a generalized design

process that is used in industry. A concept is created and then researched for feasibility and profitability. Then once a design is chosen, a detailed plan is created to bring the design to life. Prototypes are then created and tested. Ideally, multiple prototypes would be created and tested rigorously, and new prototypes would be created until a final product is made. Due to our time constraints, this processed will be shortened and simplified in order to complete a working prototype within the two semesters of Senior Design. Also the process does not necessarily have to follow a linear path as depicted below. For example, research of the design may be performed during multiple stages of the process if any issues are found or a design modification is required. Also testing may result in moving back to a previous step to fix any issues found in testing.

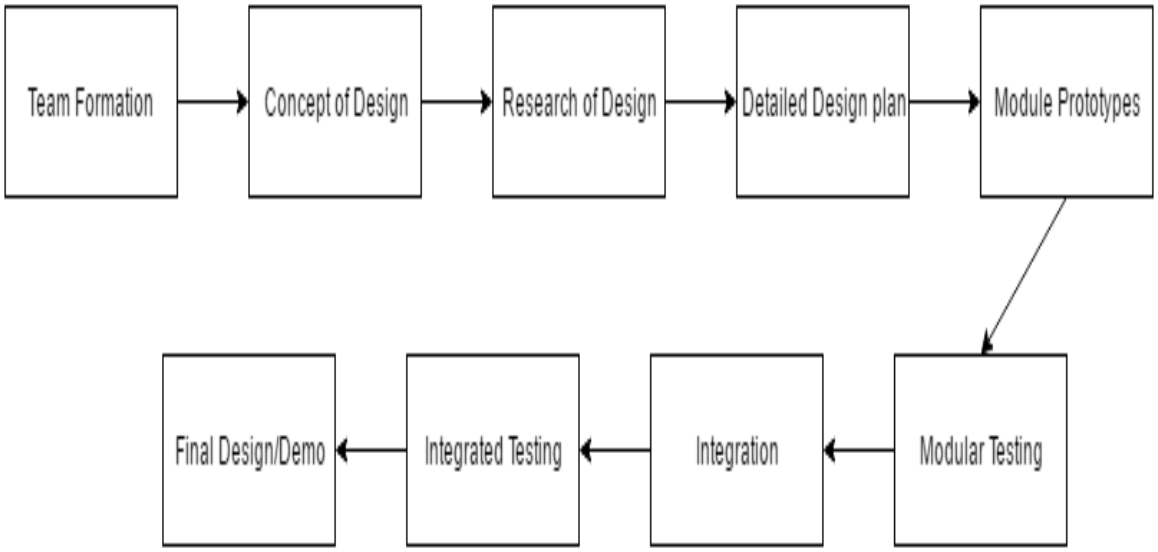


Figure 9-1: Overall Design Process

Figure 9-2 shows the milestones of the electronics design. First components will be tested on a prototype board or breadboard, and from there the PCB will be created and sensors will be assembled to the microcontrollers. A physical checkout will be done to verify that components are correctly laid out and the design matches the schematic. If any errors are found, the team will return to the previous steps with a new design or component until a physical checkout passes. The final step is the assembly of all electronic devices into one device, creating a prototype that can then be tested. It is possible however that after assembly of the hardware, issues can still be found during testing which can result in the electronics design processes being revisited to modify the previous design.

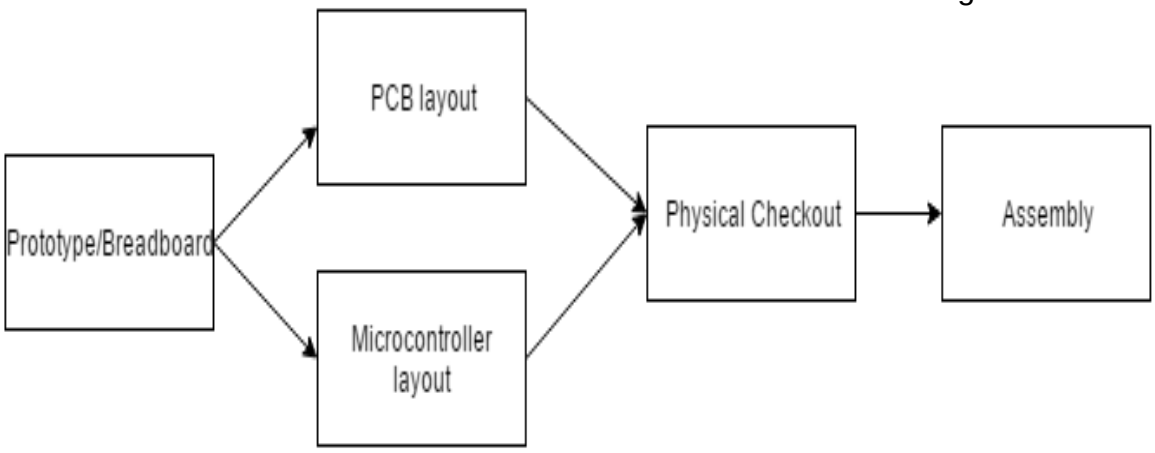


Figure 9-2: Electronics Design milestones

Figure 9-3 shows the milestones in testing for our design. Once the first design is created, the electronics and software will be tested separately to ensure each is working properly. Next, the hardware and software will be integrated and tested in a modular format, where one functionality is tested at a time to verify each module is working correctly. Finally, a full demonstration will be done with all functionalities to ensure a successful final demonstration

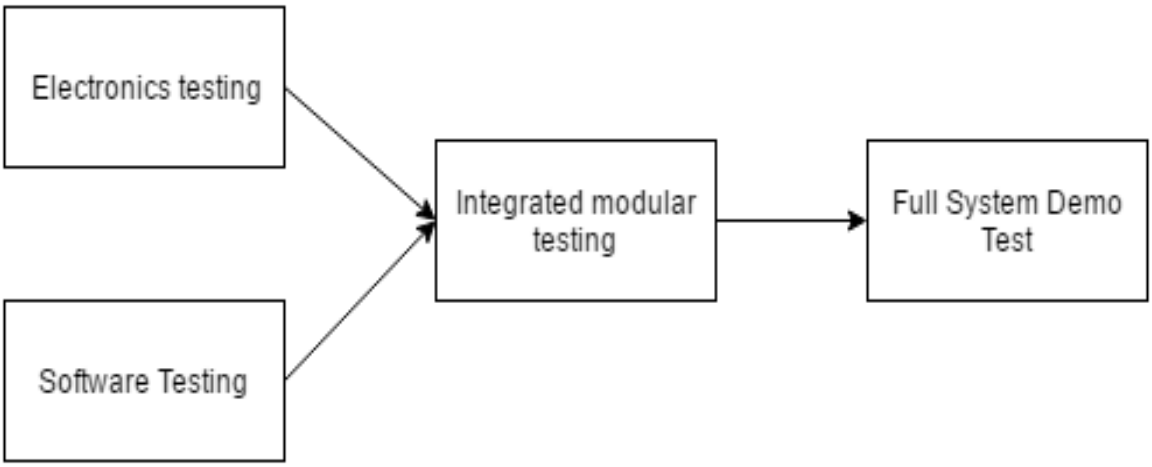


Figure 9-3: Testing milestones

9.3 Project Roles and Labor assignments

The roles of this project can be split into EE and CpE roles. The EE roles are in charge of verifying that all hardware components are integrated successfully with each other and with the whole system. They will be mainly in charge of PCB design and wiring, as well as control systems. The CpE roles will be mostly in charge of software design. The role breakout and labor assignments will be as shown below:

- Ryan Hromada: EE role- PCB design, component interfacing, electronics testing
- Adam Baumgartner: EE role- PCB design, servo control and interface, vehicle assembly
- Austin King: CpE role- Software design and testing
  Kevin Plaza: CpE role- Software design and testing

Table 9-1 and 9-2 below will give a listed outline and expected completion time frame for each section of the project. The importance of this timeline is so that all parties can operate smoothly. Since the device is the joint effort of the four persons listed above each person's completion date will affect the time in which a following member may start their portion of the project. Some of the sections can be completed in conjunction with other sections while some cannot. Most of the initial research can be completed independently but when finalizing processes and components each section must be complete sequentially.

Task	Person Responsible	Start Date	End Date
Initial Project identification	All parties	9/1/2016	9/9/2016
Further Clarify the Objective of the project	Ryan	9/9/2016	9/17/2016
Determine MCU for the device	Kevin	9/17/2016	9/27/2016
Research Methods for software implementation	Austin	9/27/2016	10/15/2016
Research Sensor Part Selection	Ryan	9/27/2016	10/15/2016
Research Servo and Chassis Design	Adam	9/27/2016	10/15/2016
Research part integration between MCU and Component selection	Kevin	10/15/2016	10/21/2016
Finalize Part Selection	Ryan/Adam	10/15/2016	10/21/2016
Complete Project Description and Related Projects	Kevin	10/1/2016	10/21/2016
Table of Contents Submission	All Parties	9/19/2016	11/4/2016
Current Draft of Senior Design	All Parties	11/4/2016	11/11/2016

Documentation			
Follow up on Status of Corrections	Ryan	11/14/2016	11/19/2016
Design Software Testing Environment	Austin	11/20/2016	11/30/2016
Design Hardware Testing Environment	Ryan	11/24/2016	12/2/2016
Complete Servo Testing Environment	Adam	12/2/2016	12/4/2016
Complete Administrative Content	Kevin	12/2/2016	12/4/2016
Reformat MCU Component Selection	Kevin	12/3/2016	12/4/2016

Table 9-1: Senior Design 1 Project Action Plan

Test Equipment	Ryan/Adam	12/6/2016	1/1/2016
Design PCB Layout	Ryan/Adam	1/1/2016	1/20/2016
Design Software Version 1	Kevin/Austin	12/6/2016	1/20/2016
PCB Testing	Ryan/Adam	2/20/2016	2/27/2016
Metric Evaluation	Kevin/Austin	2/25/2016	2/27/2016
Demo	All Parties	2/27/2016	3/1/2016
Re-evaluate performance	All Parties	3/1/2016	3/7/2016
Implement additional functionality	Kevin/Austin	3/7/2016	3/14/2016
Perform final working demonstration	All Parties	April	April

Table 9-2: Senior Design 2 Projected Project Action Plan

10.0 Military Surveillance Robot Version 2

The following section will include all the pertinent information to describe the differences in version 2 and the reason behind the changes. The largest change from version 1 to version 2 was the inclusion of the ATtiny84 and Attiny85 to the PCB design. The other inclusion was of an Op-Amp to form a basic Schmitt trigger. We quickly realized in the testing of our first design concept that some of the function didn't work out as expected.

First we realized that the number of GPIO ports required to run the Sensor Array off the Raspberry Pi caused a lot of jumper wires which wasn't as compacted as we had anticipated. Secondly the encoder wasn't giving solid readable data that was defined well in code. This proved particularly difficult for the device to make autonomous decisions without well-defined hardware performance. Lastly we were concerned that the Raspberry Pi would have difficulties calculating the movement of the vehicle, sensor array data and object detection so the inclusion of ATtiny MCU's would allow us too off load code.

10.1 ATtiny85, ATtiny84 and PCB design change

The primary motivation for the inclusion of the ATtiny85 was to use the i2c communication as a primary data line. Previously in version 1 the device was using a combination of i2c communication for the accelerometer and Pulse width modulation for the ultrasonic sensor array and encoders. When using the ATtiny85 we can consolidate all our communications into slave MCUs. This would allow the MCU's to essentially act as multiplexers with the advantage of having programmable serial data addresses.

During the version 1 design we had chosen to use three ATtiny 85's to send signals to the LM239D motor driver and collect data from our encoders as well as send and receive signals from the ultrasonic sensors. We realized that we had timing issues trying to get two ATtiny85's to match up since each was controlling a DC motor and an encoder. To reduce this issue, we decided to implement the ATtiny84. This had similar internal clock rates and memory but had extra pins to include both DC motors on one device. This helped us to avoid any issues that would arise between two different MCU's trying to communicate with each other.

The Raspberry PI was the Primary CPU in the i2c communication line. The Accelerometer was a slave device with specific commands already programmed in the device. By passing the specific code through the i2c line we could have device send back the accelerometer info which helped with the speed control of version 2 of vehicle. The same device also had specific commands receive magnetometer data that would give us values in the X-axis, Y-axis and Z-axis. This would help the device know which direct it was turning and could be

communicated via the serial i2c line. The other slave devices on the i2c communication line was the ATtiny85 and Attiny84. The ATtiny84 was assigned with the tasks of communication to both DC motors and our encoder. The ATtiny85 was relegated to the ultrasonic sensor array.

The Encoder data was sent to the ATtiny84 with a single pin. The device was always powered on like the Accelerometer and Magnetometer and would constantly send information. The ATtiny was programmed with formula to decode the PWM into speed related information. This would allow the device know its current speed. The problem with the optical encoder we chose is that depending on the lighting situation a clean pulse was not achieved. In the figure below you can see the output of the encoder data (U) versus time (t). The Top line represents that direct output of the encoder. The PWM only reads high and low for 1 or 0 data bits. The voltage of the higher section of the line can be set with the code in the ATtiny's but the issue that comes into play that there are a lot of spikes as false negatives or false positives.

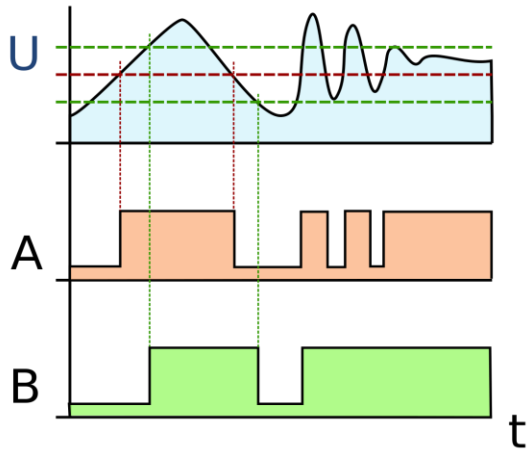


Figure 10.1 – Encoder output A:Before Schmitt Trigger B:After Schmitt Trigger

As you can see in the line A the CPU will read the data of the Pulse Width modulation depending on the threshold chosen in the code. On top of that the device will read several smaller pulses that would cause false positives. Ideally the output should be long steady pulses whose width very with speed. Their widths should be relatively similar with slight variations in speed. If the device is operating smoothly you won't see the numerous spikes followed by long pulses. This indicates that the encoder is receiving false readings. In order to clean this in line B a basic Schmitt trigger was implemented. The Schmitt trigger is basic OP-amp positive feedback circuit. Essentially we take the output of the encoder and send it through a Schmitt trigger with specific operation thresholds, this requires the voltage to drop below a certain level before switching to 0. Likewise, the voltage is require to rise above a certain threshold before switching to 1. This was particularly helpful in our application of the encoder device since certain lighting conditions would cause the encoder to give different values for the

voltage with some values being higher and lower than the others. We also realized that having two encoders was excessive so we decided to utilize only one Schmitt trigger. The reason for two Schmitt triggers originally was to determine if the device was turning in one direction or the other and based off the speed of the signals it would be easier to see the device turn left or right based of the speed of each motor. Since the inclusion of the magnetometer we were able to determine a change in the angle based of Earth's electromagnetic field. The accelerometer wasn't able to sense the changes at constant accelerations to be accurate enough to determine how much the device was turning. With the magnetometer we could have very precise data for the amount the device is turning. This allowed use to only use the data from one encoder that was primarily task with returning data on the speed of the vehicle.

As discussed in version 1 the major advantage of using the ATtiny85 was the use of the 5 voltage pins that allowed the ATtiny85 to act as a multiplexer. We soon realized that we could off load the code to calculate and determine if any objects were around the vehicle. Rather then having the pi run and calculate the sensor data we offloaded this to the ATtiny85 freeing up any processor power that would be used. This would allow the Raspberry Pi to operate the device at a higher level and focus more processing power on the Computer vision. As noted above we realized that due to a timing issue we would encounter imperfect timing switches when trying to turn the device. This would prove to be a problem as the device was required to operate autonomously. With limited processing power we wanted to simplify the commands the device used so that it didn't need to worry about error correction as much. The ATtiny84 doesn't require much processing to operate the motors. In this application it runs a very simple program to simulate the PWM output to determine Servo Speed. From there we decided to tie both the Servo enable pins to the ATtiny84 so that either both motors are on or off. While this reduced our overall we don't anticipate the vehicle needing functionality to turn on one servo vs the other as both are needed to keep the device driving straight. To turn we can simply invert the pins.

One of the flaws that we will discuss in the later sections is the noise created by the two DC motors. In version 1 of our PCB we created a common ground plane on the front and the back and we left the power line relatively large to allow for proper current. Because we were unsure the powering capabilities of the Raspberry Pi we allowed the motors to be powered both by the 5 volt power supply of the Raspberry Pi and the battery. In version 2 we realized we are getting a lot of noise over the power line. In order to help mitigate this issue we implemented two designs. One was to remove the common ground across the PCB to ensure we weren't creating extra capacitance between it and the power lines. We also decided to increase the thickness of the Ground line to compensate for this. As discussed later we added filter capacitors but in addition to this we decided to modify the PCB to separate the power lines. We made sure

the DC motors were exclusively powered via the battery pack and weren't tied to any other power line. Next we powered all the Slave MCU's, Ultrasonic Sensors and Accelerometer via the Raspberry Pi's 5 volt output. Since these devices have very low current draw it wouldn't cause the pi to drop power and clean up some of the noise created by the DC motors.

The Schmitt trigger was designed to give an output that had a high of approximately 5V. The goal was to make a cleaner signal than what was provided by the analog rotary encoder. The rotary encoder values were heavily dependent on ambient light and consequently varied significantly. The Schmitt trigger values were calculated using the following equations

$$V_+ = V_i \frac{R_2}{R_2 + R_1} + V_0 \frac{R_1}{R_1 + R_2}$$

Using the following equations, it is possible to determine the high and low triggering points indicated as V_{TH} and V_{TL}. These values indicate the distance from the transition point that the trigger activates. To determine the triggering values it is also necessary to know the transition voltage which is, in this case, the reference voltage after the voltage divider circuit. The voltages can be obtained from the following equations:

$$V_{TL} = V_{ref} \frac{R_4}{(R_3 + R_4)} + \frac{V_H R_1}{R_2} \; and \; V_{TH} = V_{ref} \frac{R_4}{(R_3 + R_4)} + \frac{V_L R_1}{R_2}$$

The final values that were determined are presented below:

R_1	15kΩ
R_2	68kΩ
R_3	4kΩ
R_4	11kΩ
V_{ref}	5V
V_H	5V
V_L	0V
V_{TL}	3.66V
V_{TH}	4.77V

Table 10.1 – Values used to determine Schmitt Trigger Thresholds

The final design solves many of the issues inherent in the analog rotary encoder. In spite of how far or below the trigger voltages the analog input is, the trigger will not vary its output. The resistance to input change is important because the encoder is susceptible to changes in ambient light. By including this module in the design, the final robot is significantly more robust and adaptable.

10.2 i2c Communication Issues and Troubleshooting

The primary impediment in the development of the autonomous tracking vehicle was the stability of the I2C communication. I2C was used to coordinate the activities of the Raspberry Pi and the two ATtiny microprocessors. The design required basic sensor and movement functions of the robot to be conducted by the ATtinys, which acted as slaves. The Raspberry Pi was the master and devoted most of its processing to computer vision and sending higher level instructions to the other microprocessors. Stable efficient communication was essential for the design because decisions about movement were being reached on one processor and executed on a second. Issues with communication stability were a serious obstacle in the execution of this design.

The I2C communication protocol was unreliable, particularly when communicating with the ATtiny84 and ATtiny85. The first issue encountered with I2C was an issue with clock stretching. Ironically to increase communication stability, both ATtiny microprocessors will hold the clock line (SCL) low if they do not understand a signal. This is a command intended to induce the master to resend the last signal. Raspberry Pi devices do not support clock stretching on I2C and the mismatch resulted in the ATtinys holding the clock low indefinitely and stopping all communication. It seemed possible that this might be a temporary status that would end after a timer that could be altered, but this was not the case. The clock line would be held low until the power was restarted and communication reestablished. We found no expedient way to stop the ATtinys from holding the line low directly, so the logical resolution was to increase the reliability of communication.

Further examination of the issue led to a possible conclusion. If the ATtinys did not miss commands they would not need to employ clock stretching. A two-sided approach was employed to resolve this issue. It was determined that the baud rate difference between the Raspberry Pi and ATtinys was responsible for the missed signals, causing the clock line to be held low. The solution was to make the I2C baud rates for the ATtinys and Raspberry Pi closer. The ATtinys could not be made fast enough to match the baud rate of the Raspberry Pi. The Raspberry Pi would be seriously limited if it had its baud rate slowed to equal that of the ATtinys. Logically, the most suitable solution was to meet in the middle. The I2C baud rate of the ATtinys would be increased and the Raspberry Pi baud rate would be decreased. The baud rate for the ATtinys was increased by setting the appropriate fused in the firmware code. The Raspberry Pi also had its baud rate altered by a code command. Once these alterations were made, the I2C no longer suffered the same clock line issue and communication between one

ATtiny and the Raspberry Pi was achieved. The next issue involved communication between the Raspberry Pi and two ATtinys.

An additional issue rendered communication impossible when communicating with two ATtinys and the Raspberry Pi simultaneously. Communication with each ATtiny individually was stable and functional, but when both were attached to the I2C lines, it was not functional. Checking the I2C addresses on the Raspberry Pi showed no devices attached. To debug the communication, an oscilloscope was attached to the two I2C lines between the Raspberry Pi and ATtinys. The output was confused with the data line oscillating in irregular square waves in contrast to the stable five volt high of an idle I2C line. The power was cycled many times to see how the irregular output developed. It started out as a normal output for a second and then the random oscillations began. A logical conclusion was that the cause of the waves concerned the powering on of the ATtinys. Potentially, the issues were due to the increased time it took to power on the larger network of chips. As each component powered on it pulled the I2C lines momentarily low, which could have been interpreted as a signal by one of the devices. The components would then send signals in response which would contribute to the confusion. This arrangement would explain the output observed on the oscilloscope. With this working theory, a solution was devised. A switch was installed on each of the two I2C lines and the robot would power on without connecting I2C. Once the power was established the I2C busses would be connected. This solution worked and allowed communication between the two ATtinys and the Raspberry Pi. The success of the implemented fix supports the idea that the error was caused by the uneven powering on of components.

Once the ATtinys and Raspberry Pi were communicating stably it was time to add the magnetometer and accelerometer to the I2C bus. The component was designed to function on I2C and it was easy to establish communication between the magnetometer and the Raspberry Pi. Unfortunately, by adding this chip to the device it again knocked out communication to the ATtinys. Again, an oscilloscope attached to the I2C lines was used to debug the issue. The lines were stable, but appeared a bit “fuzzy” which implied some high frequency noise. The most likely situation was that the magnetometer was causing a bit of noise that was interfering with the ATtinys communication. Since the noise was high frequency and the robot was a direct current device, bypass capacitors seemed to be a good solution.

Capacitors act as an open circuit to direct current, but a short circuit to alternating current, depending on the frequency. Since I2C functions based on oscillating square waves, it was necessary to be careful when selecting the capacitor size. If the capacitor was too large, it would act as a short to the data being sent. The observed noise appeared to be of a much higher frequency than the data, so the

most appropriate choice was to select a very small capacitor. After some trial and error, 22 nanofarads was found to be a capacitor size that minimized noise without impeding communication. By attaching one capacitor from each I2C line to ground, the Raspberry Pi could communicate stably with all of the I2C devices. The successful establishment of communication with the capacitors lends support to the hypothesis that the magnetometer was impeding communication by introducing noise to the I2C lines.

The last significant I2C issue that influenced the design of the robot was noise contributed by the motors. The two motors were brushed direct current servos and were exceptionally noisy. Once communication had been established with all of the I2C capable components, basic commands were run. The ultrasonic sensors were successfully tested and sent data to the Raspberry Pi. An attempt was made to control the motors by sending commands from the Raspberry Pi to the ATtinys. Immediately communication with all I2C capable devices was lost and could not be reestablished until the power was cycled for the entire robot. Again, the oscilloscope was attached to the I2C busses and used to debug the communication. There was a large amount of visible noise in the two lines when the motors were enabled. The images seemed fuzzy and high frequency noise was the likely culprit. There was significantly more noise observed than with the magnetometer alone and attempts to alter the capacitors already attached to the I2C lines were insufficient to resolve the issue. Noise mitigation techniques directly on the motors were required.

Brushed direct current motors contribute a significant amount of noise to a circuit and several techniques were required to mitigate it. First the power for the motors was isolated from the rest of the circuit. There was a common ground but the voltage was provided independently from the rest of the robot. The lines to and from the motors were made as short as possible and wound around each other in an attempt to further mitigate the noise. Finally, three capacitors were attached to each motor. The motors had two leads which acted as the positive or negative terminal depending on the direction of motion. Capacitors were attached to each lead and to the outside of the motor housing. As with the capacitors on the I2C lines this was intended to function as a short to noise, but an open circuit to the power required to drive the motors. A further capacitor was attached from one lead to the other in hopes of minimizing more noise. Once these modifications were made it was possible to run the motors while maintaining communication with the I2C capable devices. The success of this fix supports the notion that the communication was being dropped because of excess motor noise.

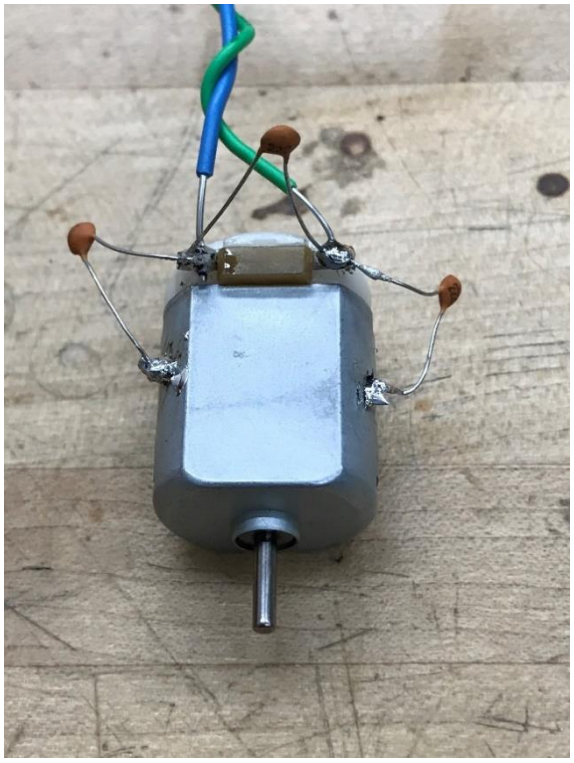


Figure 10.2 – Brushed DC motor with Capacitor Noise Filters

The obstacles to implementing I2C communication were significant and not anticipated in our design. There were four significant hurtles to I2C functioning properly and they consumed more of the design time than any other one issue. In spite of the difficulties implementing the communication protocol, once it was functional it performed admirably. After these fixes I2C was stable and robust, fulfilling all of the tasks demanded of it. It was a challenge to implement but the design became stronger as a result.

10.3 Software Design Changes

Through initial research, most of the software framework was already found. The major steps in implementing the software for the design were first programming the microcontrollers to do their required processing, next programming the raspberry pi to take the data from the microcontrollers and execute functions based on the data, then implementing the computer vision used to detect a target from the camera, and finally developing a web app that could communicate with the Raspberry Pi.

One of the biggest challenges with software in the design was programming the ATtiny microcontrollers. With the team having limited experience with the ATtinys and AVR, programming them became a large task in of itself. Most of the projects and information on the internet about using our ultrasonic sensors and

servos were for interfacing with an Arduino. Looking at some example code for programming with an Arduino, interfacing the sensors with the ATtiny85 and Arduino was an extremely easy task that only took a few lines of code. For our design however, we had to use a Raspberry Pi and I2C lines for communication. Without the built in plugins provided by an Arduino board and the additional task of interfacing the microcontrollers with the I2C lines, all the programming had to be done manually in C. All the pins that needed to used had to be declared as inputs or outputs and referenced using bitwise logic. Because of the unfamiliarity with this type of controller programming, a few software bugs caused weeks of troubleshooting due to a lack of knowledge of where the issue was coming from. I2C connection also became an issue as the implementation for connecting to I2C was quite volatile and any small error in the microcontroller code or the speed at which the raspberry pi communicated to them would cause the I2C communication to completely drop which would require a hard reset. Once this was stabilized, the final programming challenge was implementing the ability for the microcontrollers to ping the sensors appropriately. Since the sensors were made to interface with an Arduino, the online guides to using them simply pinged the sensor once, delayed for 10-15 microseconds, and then waited for the sensor to get an echo back. The challenge here was setting up a delay in the ATtinys. Since we were doing the code on the tinys instead of an Arduino or on the Pi, we had to read the manual for the controllers many times to find how to read the CPU clock and convert that into a usable time and then converting that time into a distance based on the specifications of the ultrasonic sensor. After doing this, the sensors were finally fully functional.

Once the ATtiny85 was programmed, the ATtiny84 was thought to be a much easier task to program. To program the 84, the 85 code was copied over and modified to use the same I2C callback routine used as well as a similar programming structure. Based on the necessary pin locations to control the servos, the 84 was also programmed using bitwise logic to access each individual pin. The software for the 84 was also much more straightforward than initially thought because it was simply a matter of turning certain pins high or low that would cause one or both of the servos to turn a difference direction. It didn’t require two way communication like for the sensors. When actually connecting to the servos however, there were issues with keeping I2C communications, especially when the 85 was also connected to the I2C line. Later this would be found to be a noise problem coming from the servos, however it caused a delay in software because the problem wasn’t known and the code had to be inspected multiple times and changed to try to fix the issue.

When programming the Raspberry Pi as the master, changes also had to be made from the initial design. In order to keep I2C connections stable, the baudrate and command delays had to be adjusted to make sure that the Pi wasn’t communicating faster than the microcontrollers could handle. Also in the

final stages of our prototyping, it was discovered that running the sensor code on the ATtiny85 at the same time as the ATtiny84 caused issues with the entire communication line. The first idea to fix this was implementing threading so that the Pi could do multiple actions at the same time. This threading was implementing using the Pthread library for the Raspberry Pi. This would be useful for adding in greater functionality and efficient resource use in the future, although wasn't fully used in the final design. To solve the communication issues, new "stop" commands were added to both the microcontrollers so that the Raspberry Pi could get both controllers to stop running code while it switched to a different command. This made communications much more stable, although it made the vehicle move a little more rigidly.

The next hurdle to overcome was installing and implementing SimpleCV on the Raspberry Pi to add tracking functionality to the project. This was initially thought of as one of the easiest portions of the software in the initial research stage, however it was quickly realized that the documentation was fairly limited and most installation methods were somewhat outdated and required special steps to install on the Raspberry Pi 3. It was also discovered that the Raspberry Pi camera was extremely slow when interfaced with SimpleCV due to the fact that the camera had to first communicate with the Pi and then SimpleCV, instead of just directly interacting with SimpleCV. After some research, a driver was found that could be activated on the Pi that would allow for the Picamera to detect as a USB camera, which is what SimpleCV is able to directly interface with. This greatly increased the speed at which the Raspberry Pi could execute the computer vision. Even with this speed increase though, the Raspberry Pi's limited processing power set a hard ceiling on how fast the computer vision could execute. The fastest time that was achieved was processing about one frame per second. This added a challenge of dealing with the delay on the processing that would sometimes cause the robot to not detect the target immediately even though it was directly in front of it. There really wasn't solution to this problem other than finding a larger and more expensive computer to run the computer vision on. Since it was too late in the design to obtain and integrate a completely new master computer, it was decided that the live video feed from the Pi could not work as the computer vision could not process that many frames per second. Instead, the camera would have to take one image a time and process it before it could take the next image. A change that was also made in the final prototype was adding regions to each image and having the computer vision detect on what side of the vehicle the target was on. As shown in figure 10.3, the original image was split into 3 smaller images, left, middle, and right This made the design more impressive because now the vehicle could move left or right to adjust itself so that the target will be in the center of it's field of view. This allowed for it to execute true tracking as long as the target stayed within the image frames.

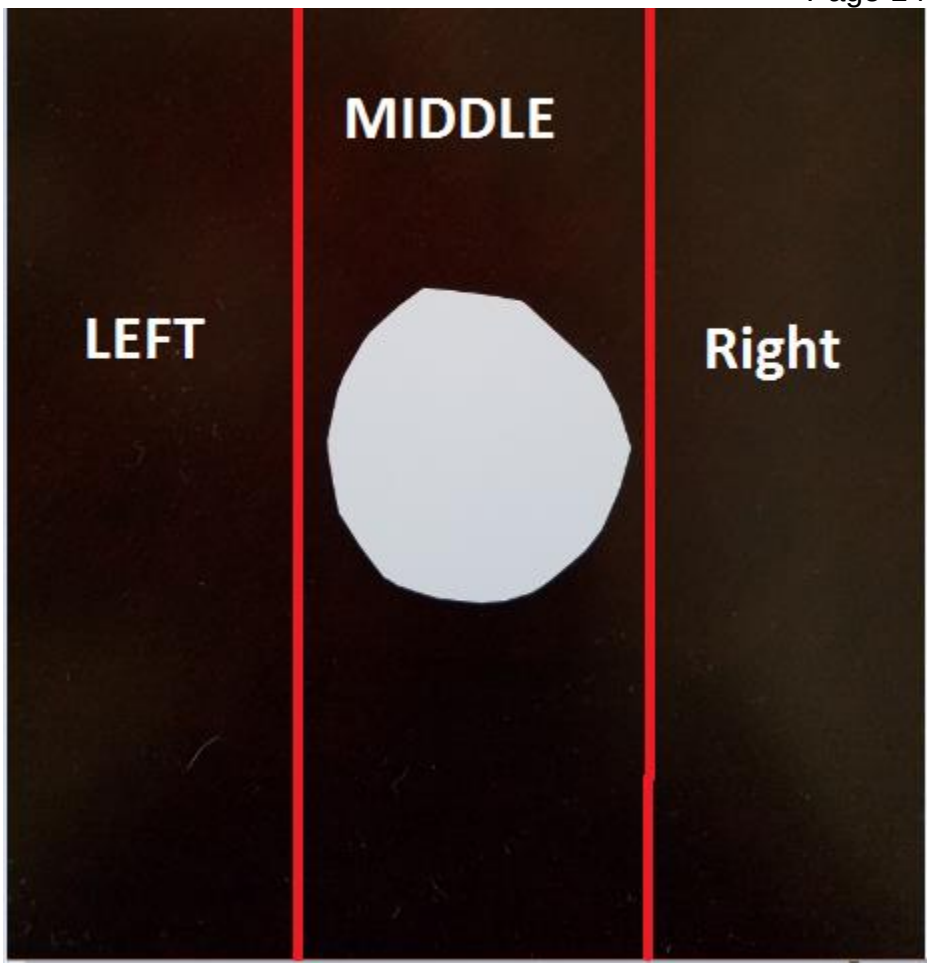


Figure 10.3 – Computer Vision Image Regions

As for the actual execution of the computer vision, some tinkering had to be done with how the images were and actually processed using the given libraries in SimpleCV. Initial testing showed that the computer vision was very sensitive to light. The color that the computer vision detected in the image varied greatly with how dim or bright the lighting was. This was because the algorithm would actually invert the colors of the image and would detect blobs that were the opposite of the desired target, as this increased accuracy in finding the correct colors. For example, a blue target would be expected to detect when the algorithm was set to filter objects that were red, since red is the opposite of blue. Depending on the lighting however, the blue target was actually detected as orange or yellow after inversion. To make sure of the reliability of the tracking, it was determined that the design would have to be demonstrated in an area with decent lighting. In the future however, this problem could be avoiding by looking into the HSV color spectrum used in computer vision, that takes into account the lighting quality when detecting colors. This phenomenon can be seen in figure 10.3 as the program doesn't the very bottom of the target as it appears to be a different color than the rest of the target due to the shadows and lighting.

Figure 10.4 - Object Detection using white circle

End.

Appendix A - References

[1] ReconRobotics, Inc, *Throwbot version1.16*
November 2016
http://www.reconrobotics.com/wp-content/uploads/2016/11/Throwbot-XT-User-Manual-v1.16-November-2016.pdf

Ryan Hromada
Hello, I'm a senior at the University of Central Florida and I am contacting ...

Andrew Drenner <Andrew.Drenner@reconrobotics.com>
to Ryan, Support

Hi Ryan

You are welcome to use the image, thank you for asking. Out of curiosity, would you be able to share what your Senior Design Project is? I'm always on the lookout for talented engineers and Senior Design Projects are generally a good indicator ☺

Regards

Andrew

--
Dr. Andrew Drenner, PhD
Chief Technology Officer
ReconRobotics, Inc
5251 W 73rd St, Ste A
Edina, MN 55439

V:(952)935-5515x112
F:(952)935-5508

PERMISSION GRANTED

[2] *ARDUINO Uno*
August 2015. JTag Electronics
http://www.jtagelectronics.com/?p=75

Ryan Hromada <rhromada@knights.ucf.edu>
to jtagelectronic.

Hello,

I'm a senior at the University of Central Florida and I am contacting you in the hopes you could grant me permissions to use a picture from your website to help facilitate my Senior Design Project. The picture is used for demonstration purposes only and will only be reprinted on Electrical Engineering Website of UCF under the Senior Design Section. The project will contain a bibliography with reference to your picture and the original author. Below is a screenshot of the image and how I intend to include it in the report.

Thanks for your time,

Ryan Hromada

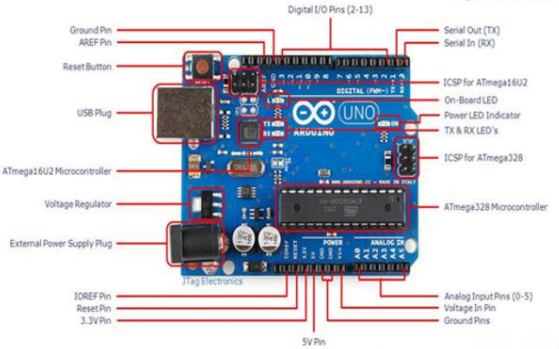


Figure 3-5: Arduino UNO REV 3 [2]

PERMISSION PENDING

[3] *USB Battery Pack for Raspberry Pi - 4000mAh*
Adafruit.
https://www.adafruit.com/products/1565

Adafruit Industries <support@adafruit.com>
to Ryan

Hi!

Please send the URL or the photo.

Thanks,
Adafruit

On Tue, Dec 6, 2016 at 8:55 AM, Ryan Hromada <support@adafruit.com> wrote:
> contactname : Ryan Hromada
> email address : rhromada@knights.ucf.edu
> message text :
> I'm a senior at the University of Central Florida and I am contacting you in the hopes you could grant me permissions to use a picture from your website to help facilitate my Senior Design Project. The picture is used for demonstration purposes only and will only be reprinted on Electrical Engineering Website of UCF under the Senior Design Section. The project will contain a bibliography with reference to your picture and the original author.
> Client IP: 104.129.204.84

Ryan Hromada <rhromada@knights.ucf.edu>
to Adafruit

Thanks for your quick reply i was hoping to use the following pictures as i intend to use the components in my project

https://www.adafruit.com/products/1565 - The battery pack main picture
https://learn.adafruit.com/16-channel-pwm-servo-driver?view=all The diagram showing how to hook up the Servo Driver

Adafruit Industries
to Ryan

Sure, all good, please use.

Ryan Hromada <rhromada@knights.ucf.edu>
to Adafruit

Thank you very much!

PERMISSION GRANTED

[4] *HS-322HD Standard Heavy Duty Servo*
Hitec RCD USA, Inc.
http://hitecrcd.com/products/servos/sport-servos/analog-sport-servos/hs-322hd-standard-heavy-duty-servo/product

Ryan Hromada <rhromada@knights.ucf.edu>
to service

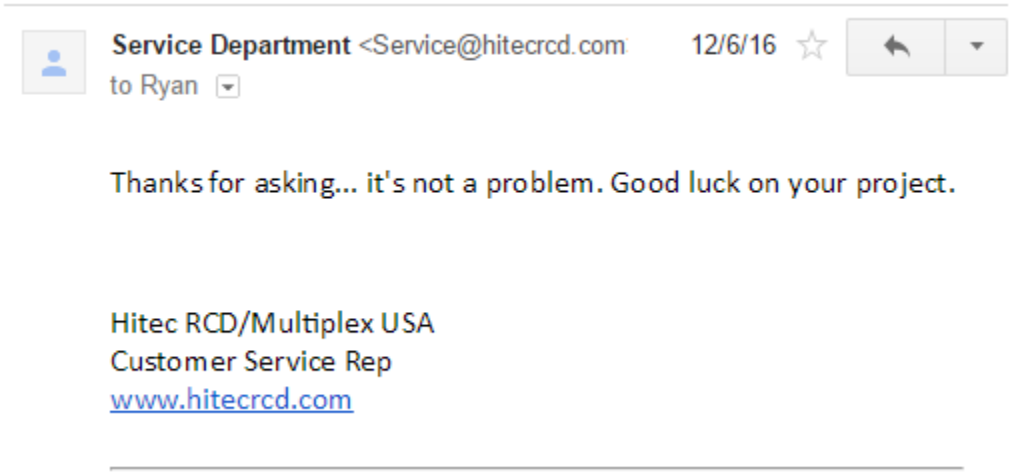
Hello,

I'm a senior at the University of Central Florida and I am contacting you in the hopes you could grant me permissions to use a picture from your website to help facilitate my Senior Design Project. The picture is used for demonstration purposes only and will only be reprinted on Electrical Engineering Website of UCF under the Senior Design Section. The project will contain a bibliography with reference to your picture and the original author. Below is a screenshot of the image and how I intend to include it in the report.

Thanks for your time,

Ryan Hromada

Figure 3-18: HS-322 [4]



Service Department <Service@hitecrcd.com 12/6/16

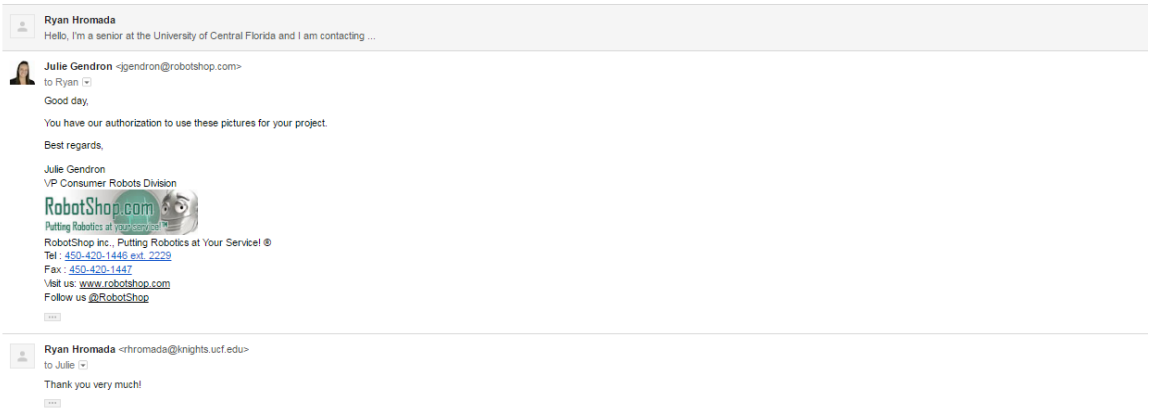
to Ryan

Thanks for asking... it's not a problem. Good luck on your project.

Hitec RCD/Multiplex USA
Customer Service Rep
www.hitecrcd.com

PERMISSION GRANTED

[5] *DFRobotShop Plate Rev A.*
RobotShop Distribution Inc.
http://www.robotshop.com/media/files/images2/dfrobotshop-rover-lexan-dimensions.jpg



Ryan Hromada
Hello, I'm a senior at the University of Central Florida and I am contacting ...

Julie Gendron <jgendron@robotshop.com>
to Ryan

Good day,

You have our authorization to use these pictures for your project.

Best regards,

Julie Gendron
VP Consumer Robots Division
RobotShop.com
Putting Robotics at your service!™
RobotShop inc., Putting Robotics at Your Service! ®
Tel : 450-420-1446 ext. 2229
Fax : 450-420-1447
Visit us: www.robotshop.com
Follow us @RobotShop

Ryan Hromada <rhromada@knights.ucf.edu>
to Julie

Thank you very much!

PERMISSION GRANTED

[6] *DFRobotShop Rover Chassis Kit*
RobotShop Distribution Inc.
http://www.robotshop.com/en/dfrobotshop-rover-chassis-kit.html#RelatedProducts

Ryan Hromada
Hello, I'm a senior at the University of Central Florida and I am contacting ...

Julie Gendron <jgendron@robotshop.com>
to Ryan

Good day,

You have our authorization to use these pictures for your project.

Best regards,

Julie Gendron
VP Consumer Robots Division
RobotShop.com
Putting Robotics at your service!
RobotShop inc., Putting Robotics at Your Service! ®
Tel : 450-420-1446 ext. 2229
Fax : 450-420-1447
Visit us: www.robotshop.com
Follow us @RobotShop

Ryan Hromada <rhromada@knights.ucf.edu>
to Julie

Thank you very much!

PERMISSION GRANTED

[7] Atmel. *ATtiny25/V / ATtiny45/V / ATtiny85/V Summary*
August 2013.
http://www.atmel.com/Images/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet-Summary.pdf

Ryan Hromada <rhromada@knights.ucf.edu>
to customerservice

Hello,

I'm a senior at the University of Central Florida and I am contacting you in the hopes you could grant me permissions to use a picture from your website to help facilitate my Senior Design Project. The picture is used for demonstration purposes only and will only be reprinted on Electrical Engineering Website of UCF under the Senior Design Section. The project will contain a bibliography with reference to your picture and the original author. Below is a screenshot of the image and how I intend to include it in the report.

Thanks for your time,

Ryan Hromada

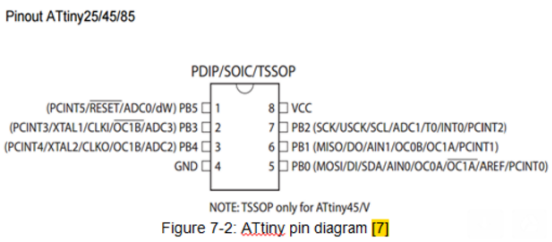
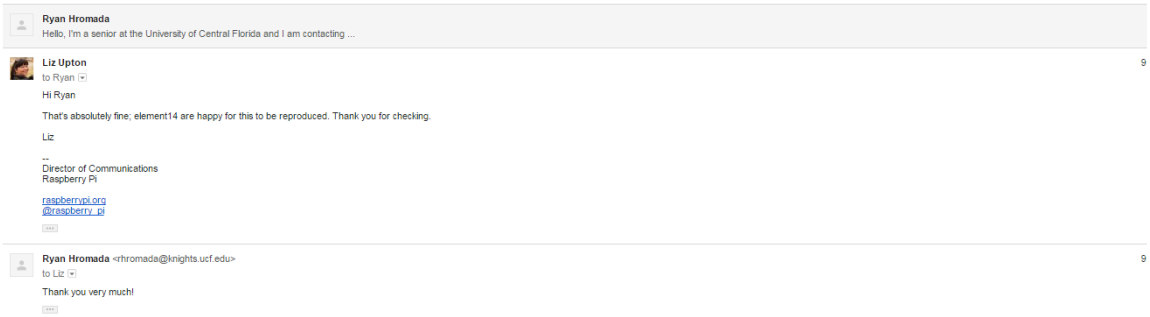


Figure 7-2: ATtiny pin diagram [7]

PERMISSION PENDING

[8] Lunarkingdom, *Pi B+ 40 pin GPIO how to use buttons and how many can I use?*
July 2014
https://www.raspberrypi.org/forums/viewtopic.php?f=78&t=82397



Ryan Hromada
Hello, I'm a senior at the University of Central Florida and I am contacting ...

Liz Upton
to Ryan
Hi Ryan

That's absolutely fine; element14 are happy for this to be reproduced. Thank you for checking.

Liz

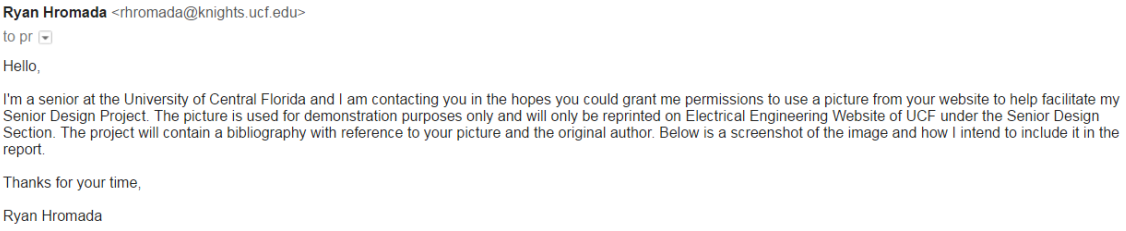
--
Director of Communications
Raspberry Pi

raspberrypi.org
@raspberry_pi

Ryan Hromada <rhromada@knights.ucf.edu>
to Liz
Thank you very much!

PERMISSION GRANTED

[10] UM10204 User manual Rev.6
April 2014. NXP:
http://www.nxp.com/documents/user_manual/UM10204.pdf



Ryan Hromada <rhromada@knights.ucf.edu>
to pr

Hello,

I'm a senior at the University of Central Florida and I am contacting you in the hopes you could grant me permissions to use a picture from your website to help facilitate my Senior Design Project. The picture is used for demonstration purposes only and will only be reprinted on Electrical Engineering Website of UCF under the Senior Design Section. The project will contain a bibliography with reference to your picture and the original author. Below is a screenshot of the image and how I intend to include it in the report.

Thanks for your time,

Ryan Hromada

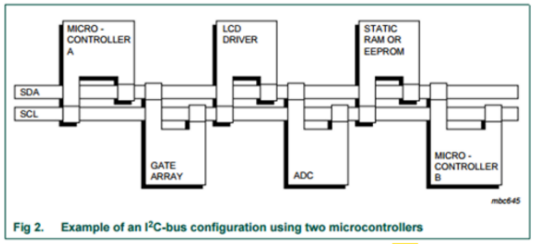


Fig 2. Example of an I²C-bus configuration using two microcontrollers

Figure 5-2: I2C Communication Diagram [10]

PERMISSION PENDING

[11] Bill Earl, *LSM303 Accelerometer + Compass Breakout*
Adafruit:
https://cdn-learn.adafruit.com/downloads/pdf/lsm303-accelerometer-slash-compass-breakout.pdf

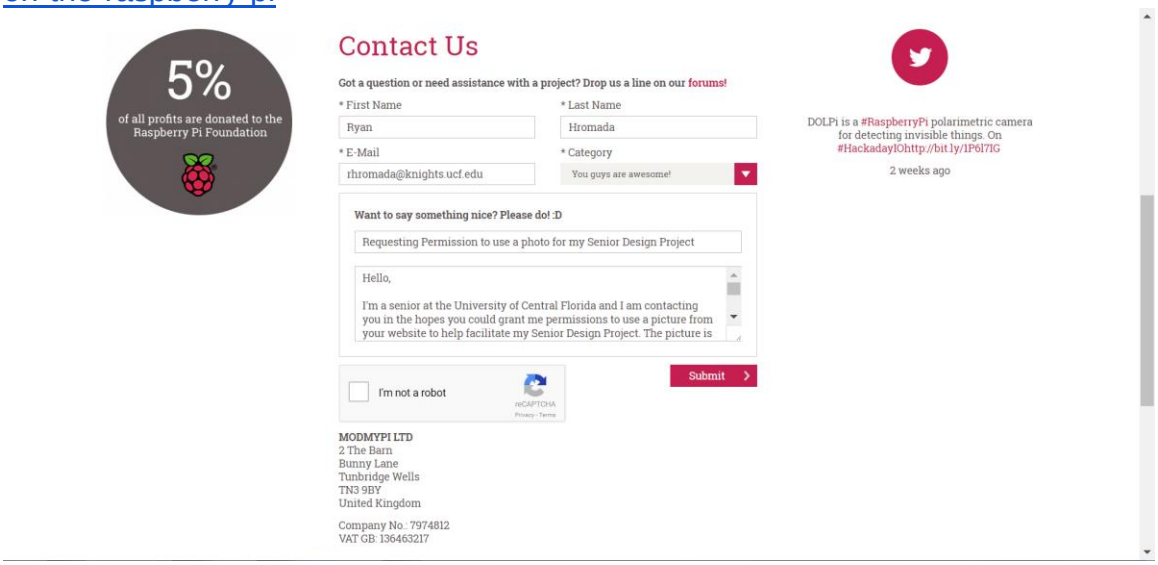
[12] *Schematic & Fabrication Print*
https://learn.adafruit.com/16-channel-pwm-servo-driver/downloads

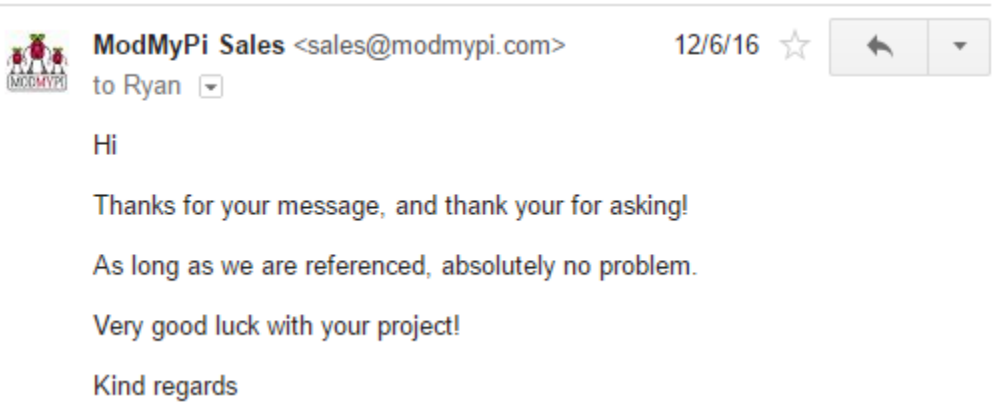
[13] MIPI Alliance. *MIPI Alliance Working Groups Overview*
http://mipi.org/

[14] *HS-SR04 User's Manual*
Cytron Technologies Sdn. Bhd.:
https://docs.google.com/document/d/1Y-yZnNhMYy7rwhAgyL_pfa39RsB-

x2qR4P8saG73rE/edit

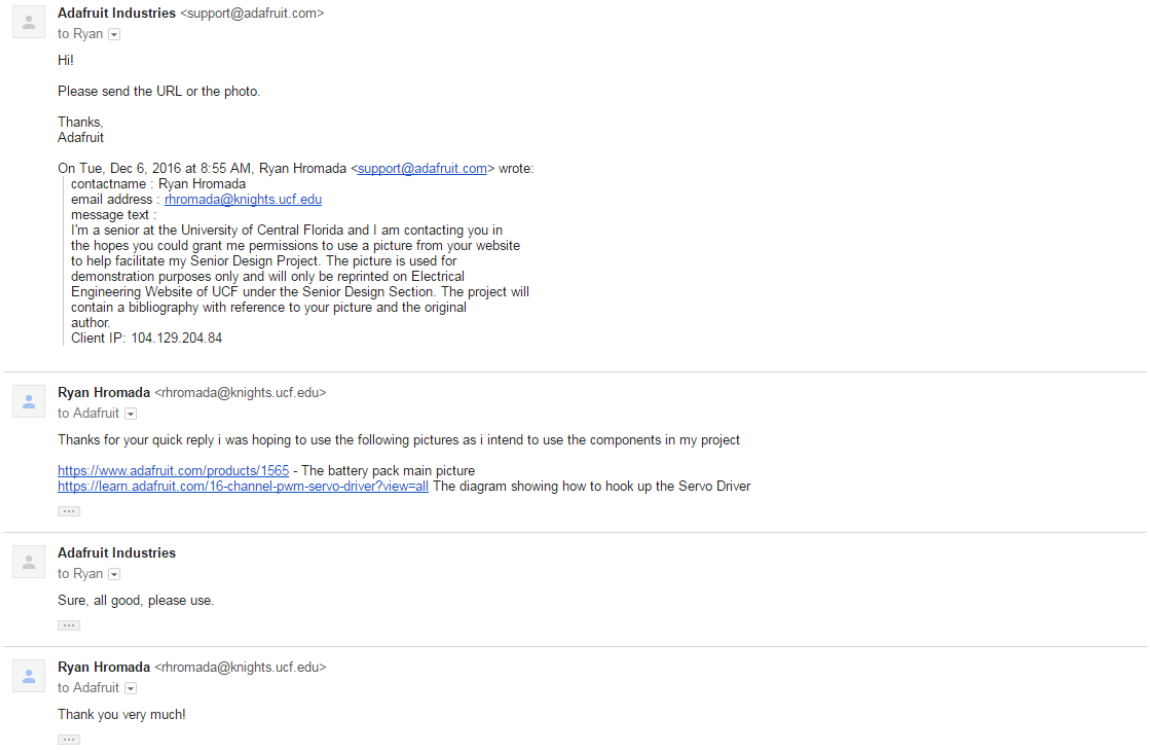
[15] *HC-SR04 Ultrasonic Range Sensor on the Raspberry Pi*
July 3rd 2014: https://www.modmypi.com/blog/hc-sr04-ultrasonic-range-sensor-on-the-raspberry-pi





PERMISSION GRANTED

[16] Bill Earl, *Adafruit 16-Channel Servo Driver with Arduino*
https://learn.adafruit.com/16-channel-pwm-servo-driver?view=all



Adafruit Industries <support@adafruit.com>
to Ryan

Hi!

Please send the URL or the photo.

Thanks,
Adafruit

On Tue, Dec 6, 2016 at 8:55 AM, Ryan Hromada <support@adafruit.com> wrote:
contactname : Ryan Hromada
email address : rhromada@knights.ucf.edu
message text :
I'm a senior at the University of Central Florida and I am contacting you in the hopes you could grant me permissions to use a picture from your website to help facilitate my Senior Design Project. The picture is used for demonstration purposes only and will only be reprinted on Electrical Engineering Website of UCF under the Senior Design Section. The project will contain a bibliography with reference to your picture and the original author.
Client IP: 104.129.204.84

Ryan Hromada <rhromada@knights.ucf.edu>
to Adafruit

Thanks for your quick reply i was hoping to use the following pictures as i intend to use the components in my project

https://www.adafruit.com/products/1565 - The battery pack main picture
https://learn.adafruit.com/16-channel-pwm-servo-driver?view=all The diagram showing how to hook up the Servo Driver

Adafruit Industries
to Ryan

Sure, all good, please use.

Ryan Hromada <rhromada@knights.ucf.edu>
to Adafruit

Thank you very much!

PERMISSION GRANTED

[17] Total Phase, *Spi Background*
http://www.totalphase.com/support/articles/200349236-SPI-Background

Jurie Weidemann, *Instrumentation, How Does an Encoder Work?*
23, July 2014: http://hightechsa.blogspot.com/2012/07/instrumentation-how-does-encoder-work.html

Michael Sklar, *Analog Inputs for Raspberry Pi Using the MCP3008*
https://learn.adafruit.com/reading-a-analog-in-and-controlling-audio-volume-with-the-raspberry-pi?view=all

Appendix B-Software Libraries

Magnetometer/Accelerometer

```
// Author: Austin King
// Date: November 8, 2016
// Original Source Code cited below:
/* HMC5883L Triple Axis Digital Compass. Compass Example. Read more: http://www.jarzebski.pl/arduino/czujniki-i-sensory/3-osiowy-magnetometr-hmc5883l.html
GIT: https://github.com/jarzebski/Arduino-HMC5883L
Web: http://www.jarzebski.pl
(c) 2014 by Korneliusz Jarzebsk i*/

#include <Wire.h>
#include <HMC5883L.h>

HMC5883L compass;

void setup()
{
Serial.begin(9600);

// Initialize  HMC5883L
// Set measurement range
compass.setRange(HMC5883L_RANGE_1_3GA);

// Set measurement mode
compass.setMeasurementMode(HMC5883L_CONTINOUS);

// Set data rate
compass.setDataRate(HMC5883L_DATARATE_15HZ);

// Set number of samples averaged
compass.setSamples(HMC5883L_SAMPLES_8);

// Set calibration offset. We shouldn't need an offset.
compass.setOffset(0, 0);
```

```
}

void loop()
{
Vector norm = compass.readNormalize();

// Calculate heading
float heading = atan2(norm.YAxis, norm.XAxis);

// Set declination angle on your location and fix heading
// You can find your declination on: http://magnetic-declination.com/
// (+) Positive or (-) for negative
// For Bytom / Poland declination angle is 4'26E (positive)
// Formula: (deg + (min / 60.0)) / (180 / M_PI);
float declinationAngle = (6 + (16.0 / 60.0)) / (180 / M_PI);
heading -= declinationAngle;

// Correct for heading < 0deg and heading > 360deg
if (heading < 0)
   heading += 2 * PI;

if (heading > 2 * PI)
  heading -= 2 * PI;

// Convert to degrees
float headingDegrees = heading * 180/M_PI;
}
```

Ultrasonic sensor

```
// Author: Austin King
// Date: November 8, 2016
// Original Source Code cited below:
// Ultrasonic - Library for HR-SC04 Ultrasonic Ranging Module.
// Rev.4 (06/2012)
// J.Rodrigo ( www.jrodrigo.net )
// more info at www.ardublog.com

#include <Ultrasonic.h>
```

```
Ultrasonic left(25,8);   // (Trig PIN,Echo PIN)
Ultrasonic right(4,17);
Ultrasonic front(23, 24)

void setup() {
Serial.begin(9600);
}

void loop()
{
left.Ranging(CM);
delay(50);
right.Ranging(CM);
delay(50);
front.Ranging(CM);
delay(50);
}
```

Servo Controller

```
#define QIK_GET_FIRMWARE_VERSION        0x81
```

```
#define QIK_GET_ERROR_BYTE              0x82
#define QIK_GET_CONFIGURATION_PARAMETER  0x83
#define QIK_SET_CONFIGURATION_PARAMETER  0x84

#define QIK_MOTOR_M0_FORWARD            0x88
#define QIK_MOTOR_M0_FORWARD_8_BIT      0x89
#define QIK_MOTOR_M0_REVERSE            0x8A
#define QIK_MOTOR_M0_REVERSE_8_BIT      0x8B
#define QIK_MOTOR_M1_FORWARD            0x8C
#define QIK_MOTOR_M1_FORWARD_8_BIT      0x8D
#define QIK_MOTOR_M1_REVERSE            0x8E
#define QIK_MOTOR_M1_REVERSE_8_BIT      0x8F

Void setup()
{
Serial.begin(9600);

//Initialize Servo Controller
qik.init();
byte cmd[5]; // serial command buffer
}

Void loop()
{
//Drive Servo M0 Forward at full speed
cmd[0] = QIK_MOTOR_M0_FORWARD
cmd[1] = 127;
write(cmd, 2)

//Drive Servo M1 Forward at full speed
cmd[0] = QIK_MOTOR_M1_FORWARD
cmd[1] = 127;
write(cmd, 2)

//Stop Servo M0
cmd[0] = QIK_MOTOR_M0_FORWARD
cmd[1] = 0;
write(cmd, 2)
```

```
        //Stop Servo M1
        cmd[0] = QIK_MOTOR_M1_FORWARD
        cmd[1] = 0;
        write(cmd, 2)
        }
```

Raspberry Pi Camera

```
        //Author: Austin King
        //Date: November 10, 2016
        //Original Source Code from: The Raspberry Pi Learning Source
        //https://www.raspberrypi.org/learning/getting-started-with-picamera/worksheet/

        //Import functions
        from picamera import PiCamera
        from time import sleep


        //create object camera that is PiCamera
        camera = PiCamera()


        //Begin video preview
        camera.start_preview()
        sleep(X)     //X is the length of time you want to preview the video feed in seconds
        camera.stop_preview()     //Stops the video feed.
```

Communication testing for Digital Compass

```
        // Author: Austin King
        // Date: November 10, 2016
        // Original Source Code cited below:
        /* HMC5883L Triple Axis Digital Compass. Compass Example. Read more: http://www.jarzebski.pl/arduino/czujniki-i-sensory/3-osiowy-magnetometr-hmc5883l.html
```

```
GIT: https://github.com/jarzebski/Arduino-HMC5883L
Web: http://www.jarzebski.pl
(c) 2014 by Korneliusz Jarzebsk i*/

#include <Wire.h>
#include <HMC5883L.h>

HMC5883L compass;

void setup()
{
Serial.begin(9600);

// Initialize  HMC5883L
// Set measurement range
compass.setRange(HMC5883L_RANGE_1_3GA);


// Set measurement mode
compass.setMeasurementMode(HMC5883L_CONTINOUS);


// Set data rate
compass.setDataRate(HMC5883L_DATARATE_15HZ);


// Set number of samples averaged
compass.setSamples(HMC5883L_SAMPLES_8);


// Set calibration offset. We shouldn't need an offset.
compass.setOffset(0, 0);
}

void loop()
{
//Read Normalized bearings from compass
Vector norm = compass.readNormalize();
```

```
//Print out Normalized values for X, Y, and Z axis.
Serial.print("X Normalized = ");
Serial.print(norm.XAxis);
Serial.print("Y Normalized = ");
Serial.print(norm.YAxis);
Serial.print("Z Normalized = ");
Serial.print("norm.ZAxis);
Serial.println();
}
```

Communication testing for Ultrasonics Sensors

```
// Author: Austin King
// Date: November 10, 2016
// Original Source Code cited below:
// Ultrasonic - Library for HR-SC04 Ultrasonic Ranging Module.
// Rev.4 (06/2012)
// J.Rodrigo ( www.jrodrigo.net )
// more info at www.ardublog.com

#include <Ultrasonic.h>


//Ultrasonic left(25,8);   // (Trig PIN,Echo PIN)
//Ultrasonic right(4,17);
Ultrasonic front(23, 24)

void setup() {
Serial.begin(9600);
}

void loop()
{
Serial.print("Front Distance = ");
Serial.print(front.Ranging(CM));
Serial.print("cm");
Serial.print(Front Timing = "):
```

```
Serial.print(front.Timing());
//Serial.print("Left Distance = ");
//Serial.print(left.Ranging(CM));
//Serial.print("cm");
//Serial.print(Left Timing = "):
//Serial.print(left.Timing());
//Serial.print("Right Distance = ");
//Serial.print(right.Ranging(CM));
//Serial.print("cm");
//Serial.print(Right Timing = "):
//Serial.print(right.Timing());
}
```

Communication testing for Servo Controller

```
#define QIK_GET_FIRMWARE_VERSION        0x81
#define QIK_GET_ERROR_BYTE              0x82
#define QIK_GET_CONFIGURATION_PARAMETER  0x83
#define QIK_SET_CONFIGURATION_PARAMETER  0x84

#define QIK_MOTOR_M0_FORWARD            0x88
#define QIK_MOTOR_M0_FORWARD_8_BIT      0x89
#define QIK_MOTOR_M0_REVERSE            0x8A
#define QIK_MOTOR_M0_REVERSE_8_BIT      0x8B
#define QIK_MOTOR_M1_FORWARD            0x8C
#define QIK_MOTOR_M1_FORWARD_8_BIT      0x8D
#define QIK_MOTOR_M1_REVERSE            0x8E
#define QIK_MOTOR_M1_REVERSE_8_BIT      0x8F

Void setup()
{
Serial.begin(9600);

//Initialize Servo Controller
qik.init();
byte cmd[5]; // serial command buffer
//Commands the get error byte and then prints it out
write(QIK_GET_ERROR_BYTE);
```

```
Serial.print(read());
}

Void loop()
{
//Drive Servo M0 Forward
cmd[0] = QIK_MOTOR_M0_FORWARD
cmd[1] = 1;
write(cmd, 2)

//Drive Servo M1 Forward
cmd[0] = QIK_MOTOR_M1_FORWARD
cmd[1] = 1;
write(cmd, 2)

//Drive Servo M0 in Reverse
cmd[0] = QIK_MOTOR_M0_REVERSE
cmd[1] = 1;
write(cmd, 2)

//Drive Servo M1 in Reverse
cmd[0] = QIK_MOTOR_M1_REVERSE
cmd[1] = 1;
write(cmd, 2)

//Stop Servo M0
cmd[0] = QIK_MOTOR_M0_FORWARD
cmd[1] = 0;
write(cmd, 2)

//Stop Servo M1
cmd[0] = QIK_MOTOR_M1_FORWARD
cmd[1] = 0;
write(cmd, 2)
}
```

Communication testing for Raspberry Pi Camera

```
//Author: Austin King
//Date: November 10, 2016
//Original Source Code from: The Raspberry Pi Learning Source
//https://www.raspberrypi.org/learning/getting-started-with-picamera/worksheet/

//Import functions
from picamera import PiCamera
from time import sleep


//create object camera that is PiCamera
camera = PiCamera()


//Begin video preview
camera.start_preview()
sleep(5)     //Video feed should appear for 5 seconds
camera.stop_preview()     //Stops the video feed.
```

Appendix C-Datasheets

Software Datasheets

Communication Testing

Test	Comments/Observations	Pass/Fail
Digital Compass/ Accelerometer		
Ultrasonic Sensors		
Servo/ Servo Controller		
Camera		
Wi-Fi		

Communication Simultaneously		

Performance Testing

Test	Comments/Observations	Pass/Fail
Voice Command Test		
Data Transmission Over Wi-Fi		
Software Track Determination Test		
Collision Detection and Avoidance Test		
Point of Interest Identification Test		
Verify Track Test		