**Autonomous Target Robot**

**EE 175AB Final Report**

**Department of Electrical Engineering, UC Riverside**

|  |  |
| --- | --- |
| **Project Team Member(s)** | Taylor Che  Francisco Munoz  Samuel Choi |
| **Date Submitted** | 03/16/2018 |
| **Section  Professor** | Dr. Ping Liang |
| **Revision** | 1.2 |
| **URL of Project Wiki/Webpage** | https://github.com/fmuno003/SeniorDesign |
| **Permanent Emails of all team members** | [tche001@ucr.edu](mailto:tche001@ucr.edu)  [fmuno003@ucr.edu](mailto:fmuno003@ucr.edu)  [schoi044@ucr.edu](mailto:schoi044@ucr.edu) |

**Summary**

This report presents review of our autonomous rover, detailing our conceptualization, design considerations, trade-offs, challenges, and their resolutions our team encountered during our six month endeavor .

**Revisions**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Description of Version** | **Author(s)** | **Date Completed** | **Approval** |
| 1.0 | First Draft of Report | Francisco Munoz  Taylor Che  Samuel Choi | 03/02/2018 |  |
| 1.1 | Second Draft | Samuel Choi | 2/6/2018 |  |
| 1.2 | Third Draft | Francisco Munoz  Taylor Che  Samuel Choi |  |  |

**Table of Contents**

[**1 Executive Summary**](#_3znysh7) **6**

[**Introduction**](#_tyjcwt) **7**

[**Design Objectives and System Overview**](#_1t3h5sf) **7**

[**Backgrounds and Prior Art**](#_2s8eyo1) **7**

[**Development Environment and Tools**](#_17dp8vu) **8**

[**Design Considerations**](#_35nkun2) **8**

[**Realistic Constraints**](#_1ksv4uv) **8**

[**System Environment and External Interfaces**](#_44sinio) **8**

[**Industry Standards**](#_z337ya) **9**

[**Knowledge and Skills**](#_3j2qqm3) **9**

[**Budget and Cost Analysis**](#_1y810tw) **10**

[**Safety**](#_4i7ojhp) **11**

[**Performance, Security, Quality, Reliability, Aesthetics etc.**](#_2xcytpi) **12**

[**Experiment Design and Feasibility Study**](#_2bn6wsx) **12**

[**Experiment Design**](#_qsh70q) **12**

[**Experiment Results, Data Analysis and Feasibility**](#_3as4poj) **12**

[**Architecture and High Level Design**](#_1pxezwc) **13**

[**System Architecture and Design**](#_49x2ik5) **13**

[**Hardware Architecture**](#_147n2zr) **13**

[**Software Architecture**](#_3o7alnk) **14**

[**Rationale and Alternatives**](#_23ckvvd) **15**

[**Low Level Design**](#_32hioqz) **16**

[**gui.py**](#_h1xqwblutw5y) **18**

[**RandNum.py**](#_kwyb8wd79k4r) **19**

[**angles.py**](#_kg54pi9sbjqn) **19**

[**calibrateIMU Functionality**](#_tcvdmmj5rekl) **19**

[**Turning Functionality**](#_6al0fjx13tht) **19**

[**Movement Functionality**](#_8zcuc6l5cvr6) **19**

[**armUp.py**](#_fqjc9b6l5gvz) **19**

[**armDown.py**](#_81jr4vnp9uz4) **20**

[**platformCCW.py**](#_25xcctgtims1) **20**

[**platformCW.py**](#_uf8z4swmt9nz) **20**

[**vibration.py**](#_u5lhqbeotrex) **20**

[**Technical Problem Solving**](#_3fwokq0) **21**

[**The Calibration of IMU Problem**](#_5q15fs41p0tw) **21**

[**Solving the Calibration of IMU Problem**](#_g1zlf5pvkwf) **21**

[**USB Buffer Issue**](#_9zb4m0pfcecm) **21**

[**USB Buffer Resolution**](#_wct1gt1aws2k) **21**

[**The Battery Supply Problem**](#_mr5u741g3qe3) **21**

[**Solving the Battery Supply Problem**](#_5t8kfewcyj5d) **22**

[**Chassis Limitations**](#_4tyhh39upcvg) **22**

[**Chassis Limitation Solution**](#_4qmwf35g67fw) **22**

[**Tracked Vehicle Problem**](#_mwhdoyj6448p) **22**

[**Tracked Vehicle Resolution**](#_1vw29yevc8bg) **22**

[**The WiFi Adapter Problem**](#_2t22t4dnmqxw) **22**

[**Solving the WiFi Adapter Problem**](#_b3q4rn8mejqb) **22**

[**User Interface Design**](#_3tbugp1) **23**

[**User Interface Screens**](#_koyuy95zq0b8) **23**

[**Test Plan**](#_37m2jsg) **24**

[**Test Design**](#_1mrcu09) **24**

[**Bug Tracking**](#_46r0co2) **24**

[**Identification of critical components**](#_111kx3o) **24**

[**Test Report**](#_206ipza) **25**

[**Test 1: IMU and Turning**](#_4k668n3) **25**

[**Test 2: GPS and Turret**](#_2zbgiuw) **25**

[**Conclusion and Future Work**](#_3ygebqi) **26**

[**Conclusion**](#_2dlolyb) **26**

[**Future Work**](#_sqyw64) **26**

[**Acknowledgement**](#_3cqmetx) **27**

[**References**](#_1rvwp1q) **28**

[**Appendices**](#_4bvk7pj) **29**

[**Appendix A**](#_mdzu8tuska0t) **30**

[**Appendix B**](#_yl0glmv8tk1k) **31**

[**Appendix C**](#_ujzlx8nhm5w7) **32**

# 1 Executive Summary

* + - * Summary

The objective of our project is to design an autonomous robot which functions as a target for long range target practice. Our design will communicate with the base station via cellular network, from which we connect to from base station’s virtual network. The base station will provide the operational boundary for which we will pull random GPS destinations, for which the robot will navigate to whilst positioning itself in the environment. At each destination, we will activate our turret to face the origin location and raise the target, counting the number of hits on target upon returning to its origin location.

* + - * Overall Goals

1. GPS Boundary Mapping and Boundary Generation
2. Autonomous GPS Navigation
3. Implementation of Kalman Filter through the BNO055
4. Targeting System, Oriented towards Origin
5. Serial Communication between Raspberry Pi and Arduino
   * + - Key Features

The key features of this project are automated movement, drawing from computed data, gathered from the provided GPS coordinates. In addition to this, we also rely on a Kalman filter to provide optimized data from our IMU to accurately trace traversal.

* + - * Objectives

1. Understand the functionalities and implementation of Kalman filters
2. Formulate an accurate method for the GPS distance calculation with feedback
3. Construct a responsive target for training providing hit feedback for review
   * + - Achievements

Our major accomplishment was constructing a low cost scale model while challenging size restrictions whilst retaining as much functionality as possible. In using the Raspberry Pi as a low cost real time programming device required us to learn Python in order to utilize its full capabilities, as well as synchronizing with an Arduino; exploiting the always on functionality to constantly parse the IMU for data.

# Introduction

*This space may be used to provide an introduction for the design and ties to other project materials.*

## Design Objectives and System Overview

Our team was tasked with designing an autonomous ground target vehicle for training purposes for Naval Sea Systems Command (NAVSEA). This vehicle will allow for dynamic targeting practice and improved reaction times in users. This is especially useful for NAVSEA because although a traditional target range can offer similar traits of dynamic targeting and reaction tests, the mobility of this vehicle allows for a much more coherent way to develop new training courses. This project is meaningful because it represents the culmination of our four years of instruction at UCR in a real life setting. The intended applications for this project as aforementioned are for dynamic targeting and improving reaction times in users. This project would be more geared towards military usage rather than civilian usage. This however does not eliminate the possibility of implementing this project privately on civilian shooting ranges.

With regards to electrical engineering, this project does not specifically delve into the technicalities of signal processing, circuit design, or embedded systems. Rather, this system is a holistic representation of all of the things we have learned over our years as students of engineering. We did utilize basics of circuit design such as power analysis and voltage regulators but it was a relatively simple approach as these are considered fundamentals we should have prior knowledge about before completing this project. We also heavily utilized the Python programming language because we were in the Raspbian environment. As mentioned before, the overall goal of this project was to provide a tangible example of the culmination of our studies. This project acts as an autonomous targeting vehicle for use in military training applications. We would successfully be able implement randomly generated training exercises so the users may get training in various scenarios.

## Backgrounds and Prior Art

Marathon Targets is a company based out of Columbus, GA which manufactures robotic targets for live fire training. The biggest similarity between our project and their product is the dynamic targeting system. Marathon Targets’ product does not specify if they utilize GPS to generate a random course whereas our project is heavily dependent on GPS.

Some of the shortcomings we have faced while implementing our design of the robotic target vehicle include: budgeting, time constraints, and experience. The biggest difference between our project and their product is the budgeting allowed for development. Marathon has the budgeting to develop a sturdy all metal frame to protect the internal components and allow for better off road performance. Our project is based off a hobby tank chassis and all the internals, including the batteries for motor drivers and the Pi, are exposed to possible shots. Regarding the internals, the biggest hindrances come from the GPS module and the IMU. We used cheap components which gave us the wide margin of error that was causing difficulties in successfully demonstrating. Furthermore, given our time constraints, it was difficult to completely debug our code. Also we did not have time to implement possible upgrades to the chassis to make it more reliable. Finally, experience was a major shortcoming throughout this project. Although we do possess strong electrical and computer engineering skills, we did have to learn how sensor fusion functions, how to properly use a GPS module, and how to incorporate all the peripherals onto the Raspberry Pi.

## Development Environment and Tools

* + 1. Raspberry Pi 3 Model B
       - Python 2.7
       - Raspbian GNU/Linux 9
       - Bash Terminal
    2. Arduino Uno
       - Arduino Uno IDE
    3. Digital Multimeter

# Design Considerations

This section describes issues that need to be addressed or resolved prior to or while completing the design as well as issues that may influence the design process.

## Realistic Constraints

Size constraints had the most impact on our project. Originally, we had kept the cnc metal palates and intended on 3D printing a cover to hold all the components. However, we did have to place both the Raspberry Pi and the Arduino and this method did not allow for adequate spacing of the wires. We disassembled the bot and used tetrix beams for extra spacing and support.

Power consumption was the next biggest constraint of our project. The Raspberry Pi needs 5V @ 2.5A to function properly. We utilized a 7.2V battery in conjunction with 5v regulator to provide stable power to the Pi. However, the regulator only supplies 2A of current so we do have some issues with undervolting the Pi. Throughout our testing, we did not run into many issues however with calculations or executing the code. We mitigate this issue by replacing the regulator with a 3A buck converter.

## System Environment and External Interfaces

Raspberry Pi 3 Model B

* Chassis Movement Motor Control (L298N)
* BU-353-S4 (GPS) Monitoring
* SW-420 Vibration Sensor
* Turret Motor Control (TB6612)
* NEMA-17 Stepper Motors
* USB Communication Interface
* I2C Interface
* IEEE 802.11ac

Arduino Uno

* Adafruit BNO055 Absolute Orientation Sensor (IMU)
* I2C Interface
* USB Communication Interface

## Industry Standards

* + 1. I2C

I²C (Inter-Integrated Circuit), pronounced I-squared-C, is a synchronous, multi-master, multi-slave, packet switched, single-ended, serial computer bus invented in 1982 by Philips Semiconductor (now NXP Semiconductors). It is widely used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication.

* + 1. USB

USB, short for Universal Serial Bus, is an industry standard that was developed to define cables, connectors and protocols for connection, communication, and power supply between personal computers and their peripheral devices.

* + 1. IEEE 802.11ac

IEEE 802.11ac is a wireless networking standard in the 802.11 family (which is marketed under the brand name Wi-Fi), developed in the IEEE Standards Association, providing high-throughput wireless local area networks (WLANs) on the 5 GHz band.

* + 1. NMEA

NMEA 0183 is a combined electrical and data specification for communication between marine electronics such as echo sounder, sonars, anemometer, gyrocompass, autopilot, GPS receivers and many other types of instruments. It has been defined by, and is controlled by, the National Marine Electronics Association.

***Describe the industry standards involved with the hardware and software in your design. This may include hardware, software and protocols, such as RS232, I2C, USB, PCI, IEEE802.11, Bluetooth, Zigbee, RFID, UAL, safety standards, etc. Also discuss de facto standards such as operating system APIs, device drivers, programming language/environments, popular interfaces such as SPI, etc. used in your project.***

***You must reference the standards documents and describe how you design will comply with the standards.***

***You must describe how the standard requirements affected your design and include examples of aspects of the standards that you encountered in your design projects, e.g., voltage levels, packet size, baud rate, data rate, carrier frequency, etc.***

## Knowledge and Skills

Coding Languages:

* Python 2.7
* C
* Bash

Courses required:

1. Francisco Munoz
   1. Courses Related
      1. CS141 (Intermediate Data Structures and Algorithms)
      2. CS010 (Introduction to Computer Science I)
      3. CS012 (Introduction to Computer Science II)
      4. CS120B (Introduction to Embedded Systems)
      5. CS122A (Intermediate Embedded and Real-Time Systems)
      6. CS100 (Software Construction)
      7. EE100A (Electronic Circuits)
   2. New Knowledge/Skills
      1. Python scripting
      2. Bash Scripting & Commands
   3. Previous Knowledge/Skills
      1. Python Coding
      2. Calculus
      3. Trigonometry
      4. C++/C Programming
2. Samuel Choi
   1. Courses Related
      1. EE100A/B (Electronic Circuits)
      2. EE128 (Data Acquisition, Instrumentation, and Process Control)
      3. CS010 (Introduction to Computer Science I)
      4. CS013 (Introduction to Computer Science II)
      5. EE120B (Introduction to Embedded Systems)
   2. New Knowledge/ Skills
      1. Python
   3. Previous Knowledge/Skills
      1. Calculus
      2. Trigonometry
      3. C++/C Programming
3. Taylor Che
   1. Courses Related
      1. EE100A/B (Electronic Circuits)
      2. EE120B (Introduction to Embedded Systems)
      3. CS010 (Introduction to Computer Science I)
      4. CS013 (Introduction to Computer Science II)
      5. EE141 (Probability in Electronic Circuits)
   2. New Knowledge/Skills
      1. Python
      2. GPS Interfacing
   3. Previous Knowledge/Skills
      1. C++/C Programming
      2. Calculus
      3. Robotics (First Tech Challenge)

## Budget and Cost Analysis

|  |  |  |
| --- | --- | --- |
| ***Component*** | ***Price*** | ***Quantity*** |
| **11.1V Lithium Polymer (LiPo) Battery** | **$48.90** | **1** |
| **Dean’s Connectors** | **$12.65** | **1** |
| **SMAKN DC/DC Converter 12V Step Down to 5V/3A Power Supply Module** | **$6.99** | **1** |
| **Raspberry Pi 3 Model B** | **$35.87** | **1** |
| **Voltage Regulator** | **$14.90** | **1** |
| **Raspberry Pi Case** | **$17.99** | **1** |
| **Arduino Uno** | **$23.38** | **1** |
| **NEMA-17 Stepper Motor** | **$14.00** | **2** |
| **L298N Motor Driver** | **$6.89** | **3** |
| **WZY569 RC Tank** | **$125.00** | **1** |
| **Adafruit BNO055 Absolute Orientation Sensor** | **$34.95** | **1** |
| **SW-420 Vibration Sensor** | **$5.89** | **1** |
| **Adafruit TB6612 Motor Driver** | **$4.95** | **1** |
| **BU-353-S4 USB GPS Receiver** | **$36.90** | **1** |
| **Long Range WiFi USB with Antenna for Raspberry Pi** | **$11.96** | **1** |

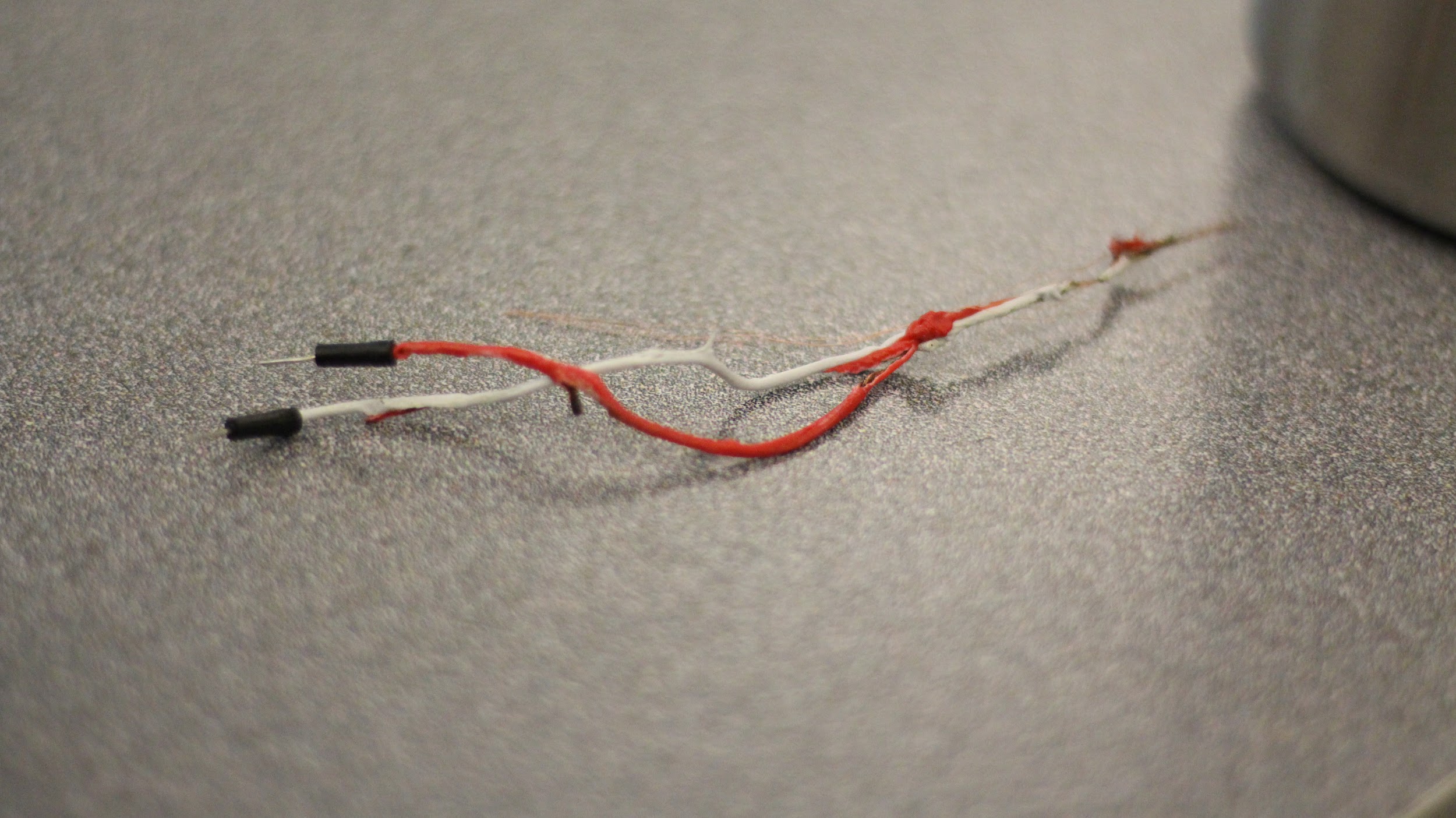
## Safety

Our primary objective in consideration of safety was to protect and isolate the wiring and power supplies to protect them from contact damage or disconnection, as well as to facilitate hardware troubleshooting. This is completed by compacting the wiring to the rear and underside breadboard of the robot.

Our primary power supply outputs approximately 11.1V to two L298N motor drivers which have the capacity to deliver 12 volts at 2 amps, significantly higher than our current power requirements. We have wired these drivers neatly across the chassis in such a manner where they do not contact any metal components directly and away from any moving components by affixing them to the underside breadboard, eliminating the need for long supply wires which are more susceptible to disconnection. Another consideration we made was for supplying the turret peripherals with power. This was done by zip tying the wires in such a way where we were allowed 180o of freedom relative to the front of the robot and securing the leads on the breadboard with electrical tape. We further protect these long leads by placing a acrylic sheet at the base of the turret to minimize the chance of disconnection.

Our second power supply, a 7.2 NIMH (Nickel Metal Hydride) battery, is passed through a buck converter to effectively convert it to supply 5V at 3 amps to our Raspberry Pi. This is affixed to the side of the robot such to maintain visibility of its trailing leads.

Careful segregation of wiring components allowed us to attach three peripherals, IMU, motor driver, and regulator side by side on the breadboard, fed by both supplies.



*Melted wires due to reversed polarity of the buck converter*

## Performance, Security, Quality, Reliability, Aesthetics etc.

As referenced in section 3.6, our primary emphasis of our design was to minimize the risk of shock by segregating our wiring on the underside breadboard and optimizing or isolating power supply leads. This attention to detail allowed us to maintain a relatively clean top side appearance, as there are few visible wires on the upper chassis. This was at great benefit to reliability, our hardware experienced no significant failure during our six month development time.

Performance and quality was somewhat hampered by the quality of the 12V drive motors chosen for this project. These motors provided the minimum torque required for smooth surface traversal. However, when operating on rougher terrain, this minimal satisfaction of our requirements becomes immediately apparent, as the motors struggled to turn the tracks. However, the chassis was more than sufficient for our project, demonstrated by an accidental 12 foot drop test; the resulting damage was minor and easily fixed.

Our Arduino, unlike our Raspberry Pi did not pose any challenges as we programmed our project. Our Raspberry Pi was susceptible to wireless disconnection, most likely due to the unreliability of the internal Broadcom WiFi chip. This was alleviated by utilizing an external WiFi module to bypass the internal chip and we experienced little to no issues past that point.

Battery performance and longevity was a concern for our project, a single supply was insufficient to both supply the 6 motors on top of the Raspberry Pi and Arduino combination. We alleviate this issue by using two different batteries to extend the operational life of the system, one connected to the motor drivers, the other the Raspberry Pi and Arduino. This allows us approximately 5 hours of Raspberry Pi and Arduino uptime and 2 hours of full power motor operation.

# Experiment Design and Feasibility Study

## Experiment Design

Our project relies on a BU-353-S4 GPS module for positioning and navigation. The accuracy of this module is ±1 meter. We pass the resulting information into the Raspberry Pi via USB to process this data for calculation.

In addition to the GPS, we use a BNO055 IMU to direct our traversal angle relative to North to orient our travel vector. This suffers from compounding error over time and an expected drift of up to 20o is expected. This information is passed into the Arduino via USB and processed by opening a serial port on the Raspberry Pi.

## Experiment Results, Data Analysis and Feasibility

Upon testing by Samuel Choi and Francisco Munoz, it was found that the GPS was fairly accurate with an error of 1.6 meters (approximately 6 feet) indoors, and 0.3 meters (approximately 1 foot) outdoors. This information confirmed the reliability of the resulting coordinates and calculations needed to perform our navigation algorithm.

In Taylor Che’s various testing of the IMU’s Kalman algorithm, we calculated an average drift of 1o at 111520 baud, much less than expected, but a resulting baudrate conflict occurred, where the Raspberry Pi’s serial buffer and onboard USB buffer would overflow; where the data would be output erroneously, instead of an expected float value, a hex value would be output. The resolution to this was to adjust the baud rate to 9600 which fixed this issue, but in turn yielded an acceptable 6o of drift.

# Architecture and High Level Design

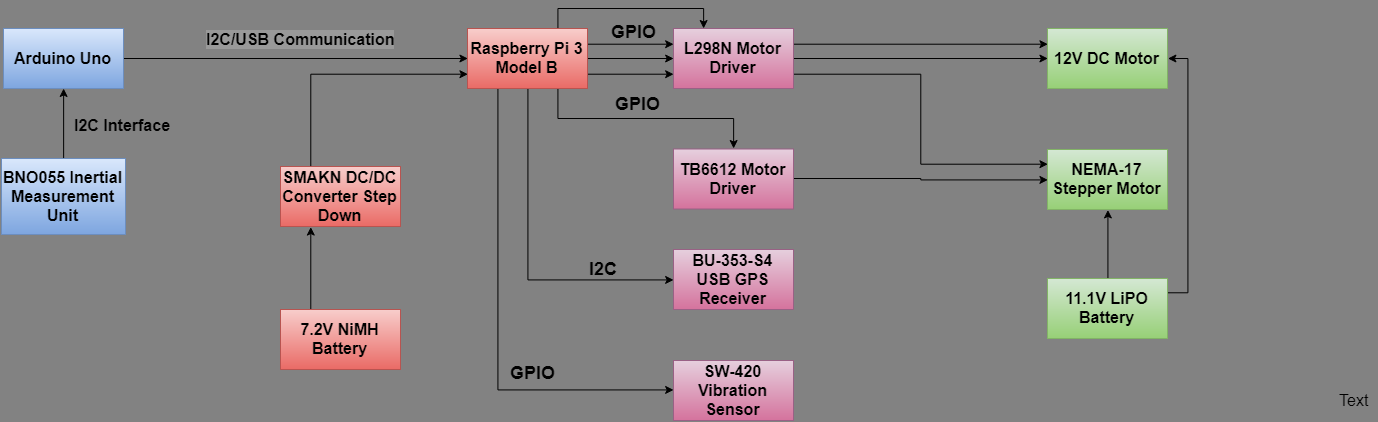
## System Architecture and Design

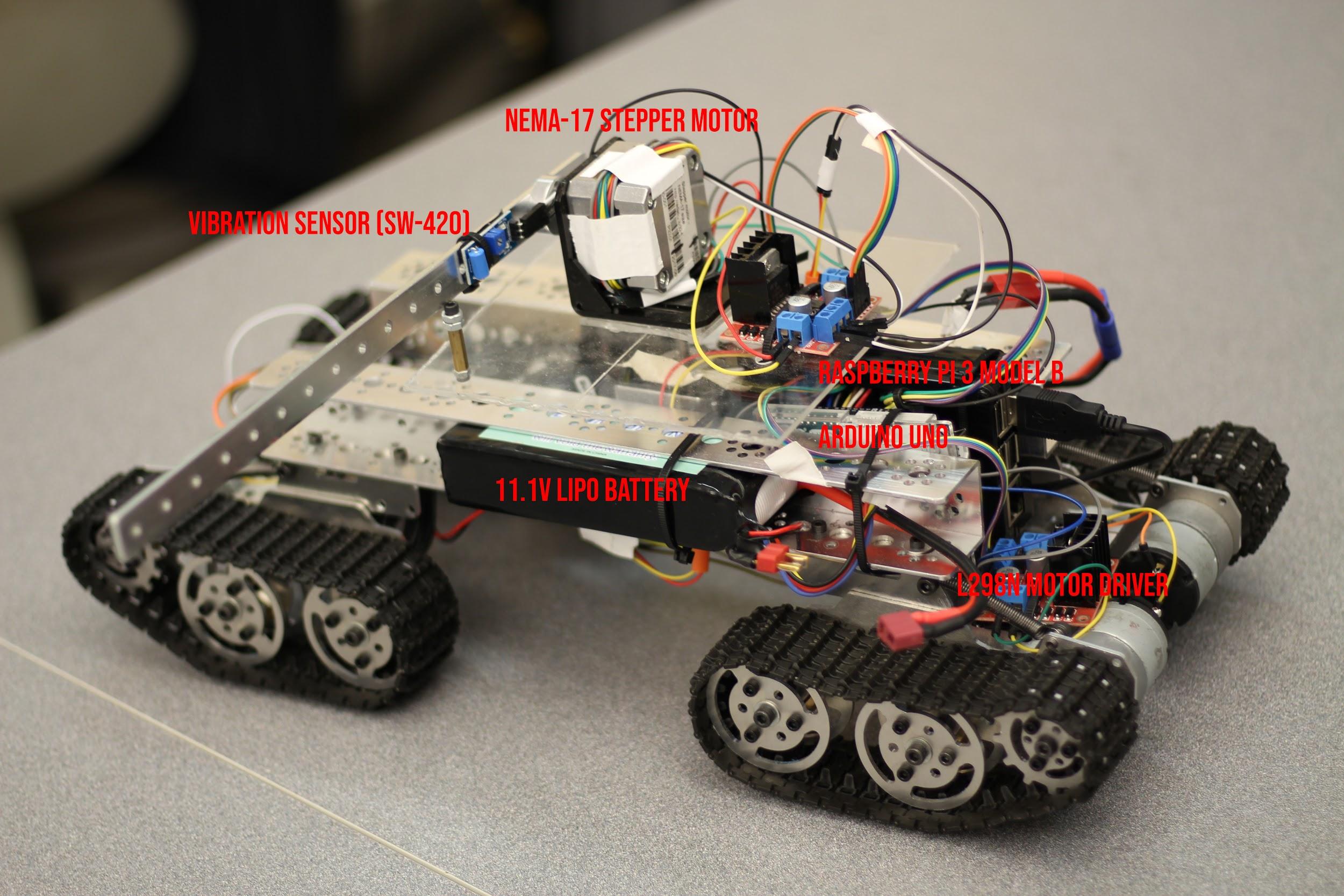
This section provides a high level overview of the structural and functional decomposition of the system. Focus on how and why the system is decomposed in a particular way rather than on details of the particular components. Include information on the major responsibilities and roles the system (or portions thereof) must play. A block diagram representation of the architecture must be included, which should show the hierarchical structure of the modules; interaction and interface among modules and with databases, external software, system, and networks

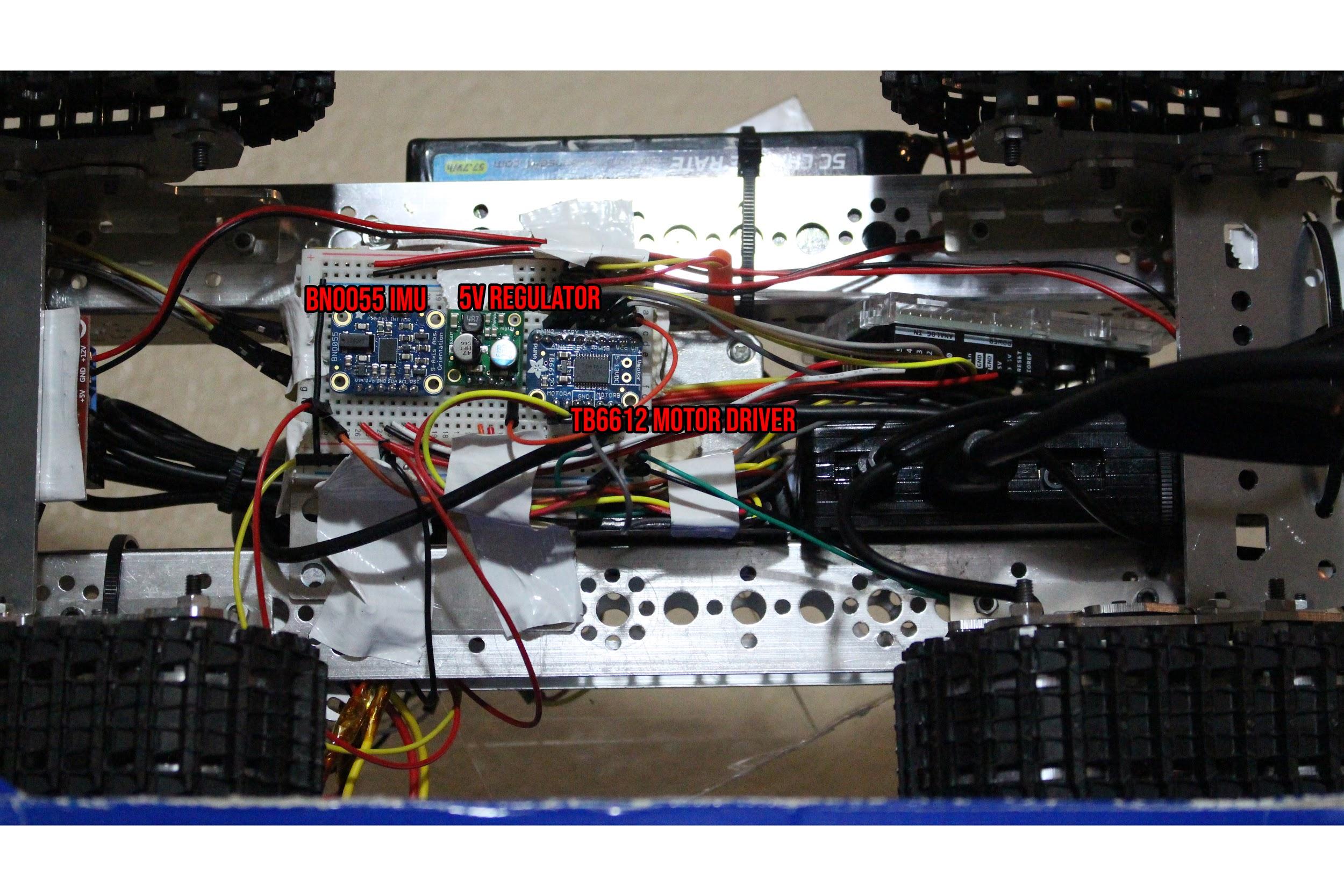
Provide a description and block diagrams of a system element or set of elements at a conceptual or functional level that describes a clearly defined view or model of the entire system or a subset of the system. Each component represents a similar grouping of functionality.

State clearly who is responsible for which module/task

## Hardware Architecture

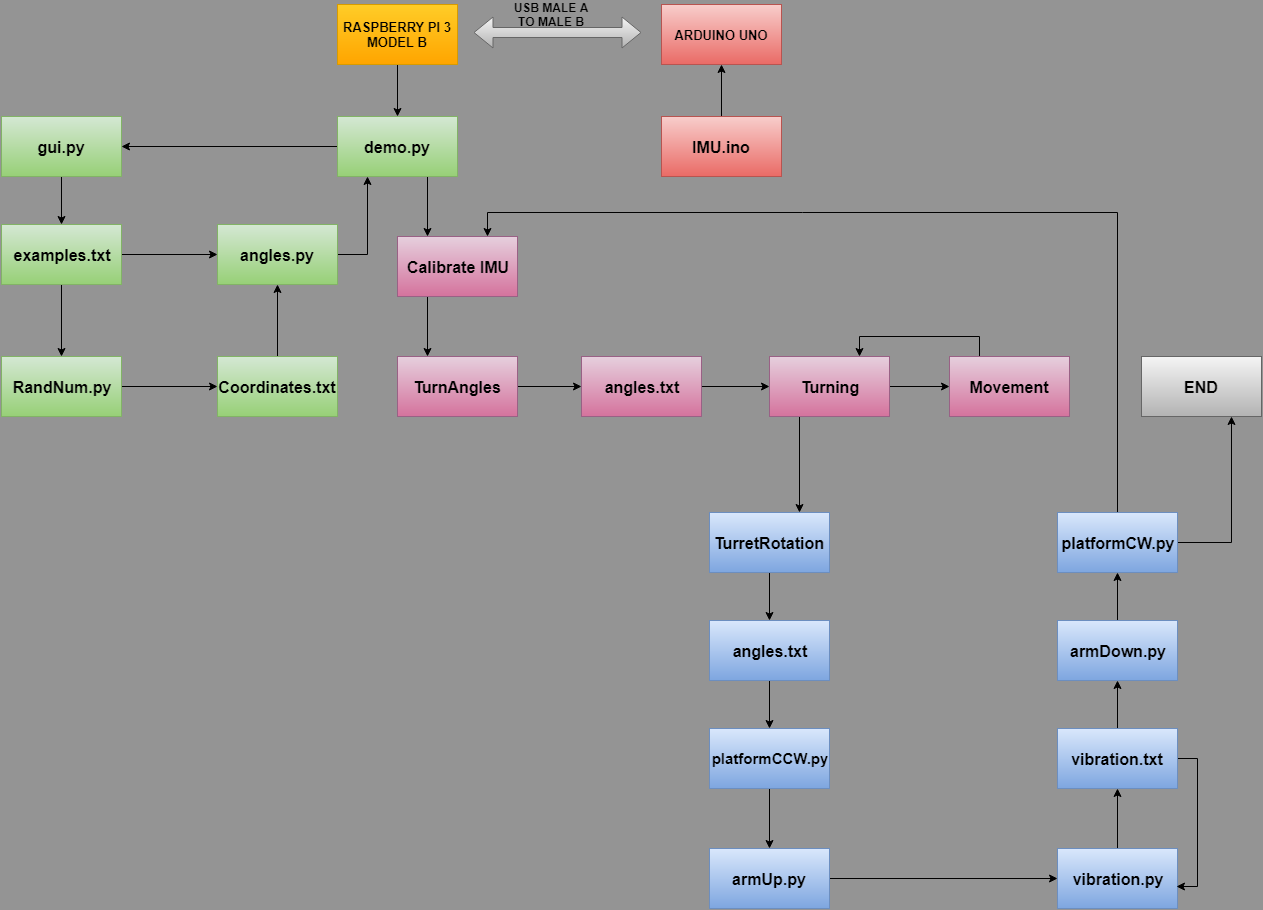


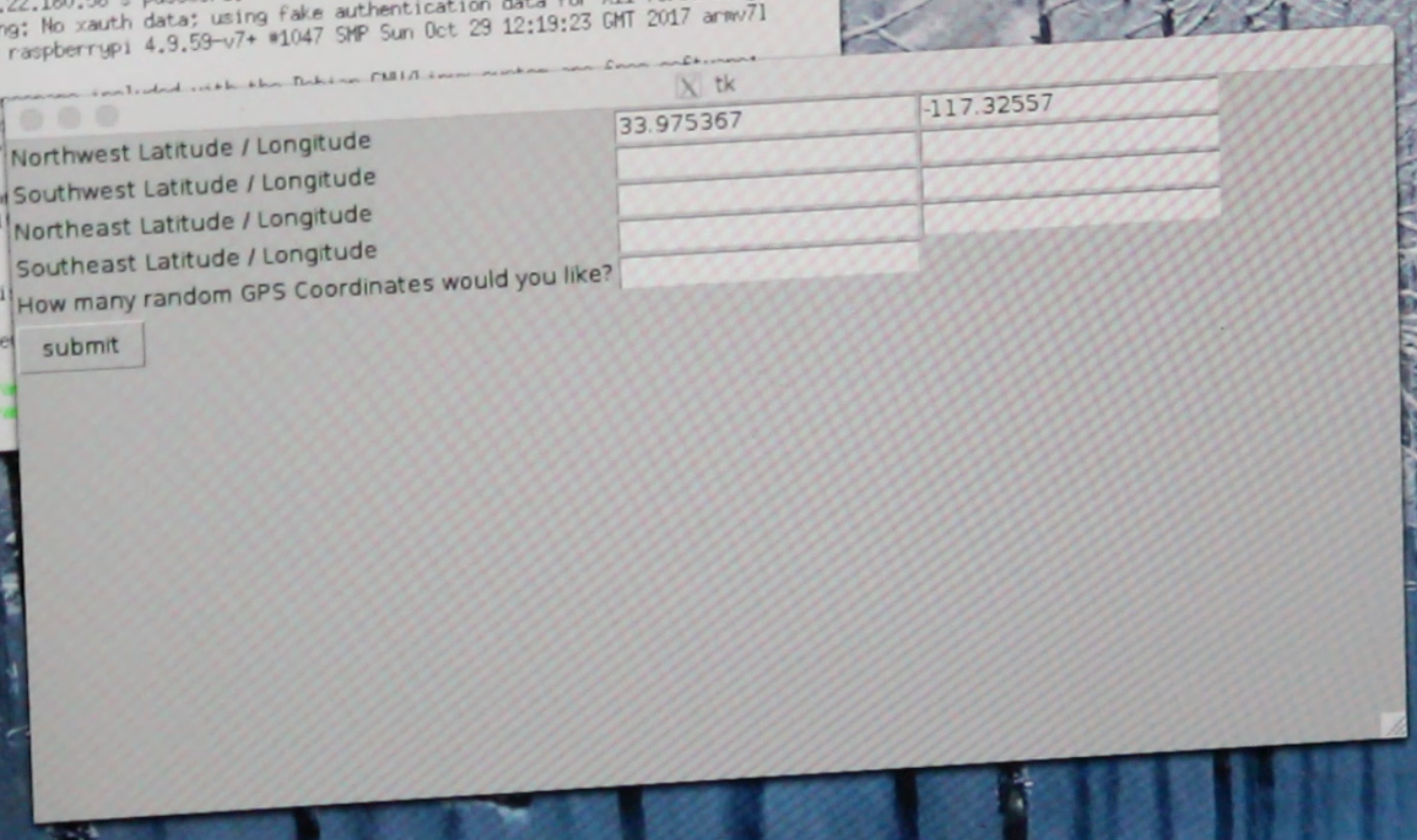




|  |  |
| --- | --- |
| **Module** | **Responsibility** |
| BNO055 Inertial Measurement Unit | Taylor Che |
| SMAKN DC/DC Converter Step Down | Taylor Che, Samuel Choi |
| L298N Motor Driver | Taylor Che, Samuel Choi |
| TB6612 Motor Driver | Taylor Che, Samuel Choi |
| BU-353-S4 USB GPS Receiver | Francisco Munoz |
| SW-420 Vibration Sensor | Taylor Che, Samuel Choi, Francisco Munoz |
| 12V DC Motors | Samuel Choi |
| NEMA-17 Stepper Motor | Francisco Munoz |
| 11.1V LiPO Battery | Taylor Che |
| 7.2V NiMH Battery | Taylor Che |

## Software Architecture





*GUI where the user can input GPS coordinates for the boundaries and number of random points*

|  |  |
| --- | --- |
| **Module** | **Responsibility** |
| IMU.ino | Samuel Choi, Taylor Che |
| demo.py | Francisco Munoz, Taylor Che, Samuel Choi |
| gui.py | Francisco Munoz |
| Calibrate IMU | Taylor Che, Samuel Choi |
| RandNum.py | Francisco Munoz |
| angles.py | Francisco Munoz |
| Turning | Taylor Che, Samuel Choi |
| Movement | Taylor Che, Samuel Choi |
| Turret Rotation | Francisco Munoz |
| armUp.py | Francisco Munoz, Taylor Che, Samuel Choi |
| vibration.py | Taylor Che, Samuel Choi |
| platformCCW.py | Francisco Munoz, Taylor Che, Samuel Choi |
| platformCW.py | Francisco Munoz, Taylor Che, Samuel Choi |
| armDown.py | Francisco Munoz, Taylor Che, Samuel Choi |
| TurnAngles | Taylor Che, Samuel Choi |

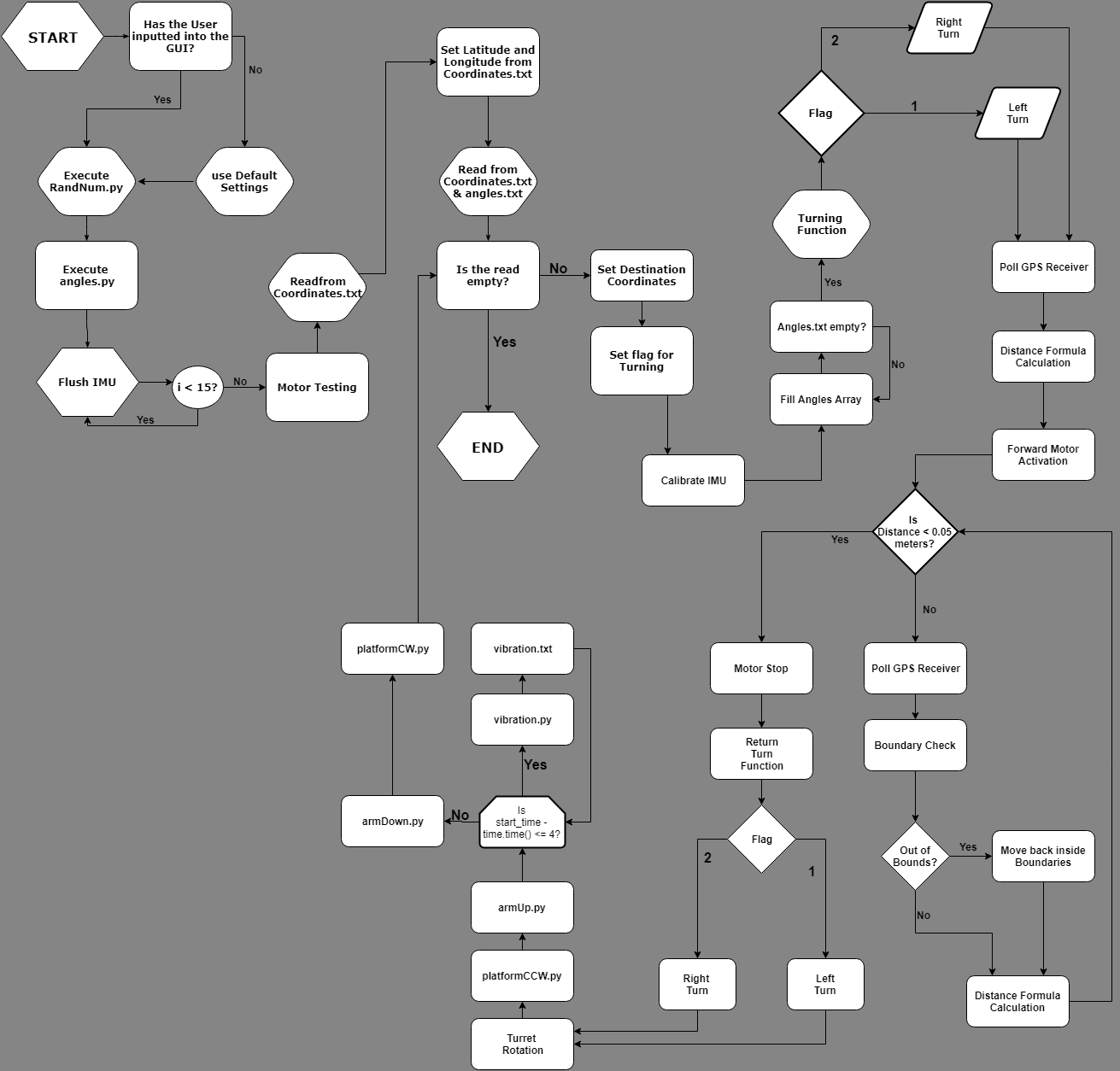
## Rationale and Alternatives

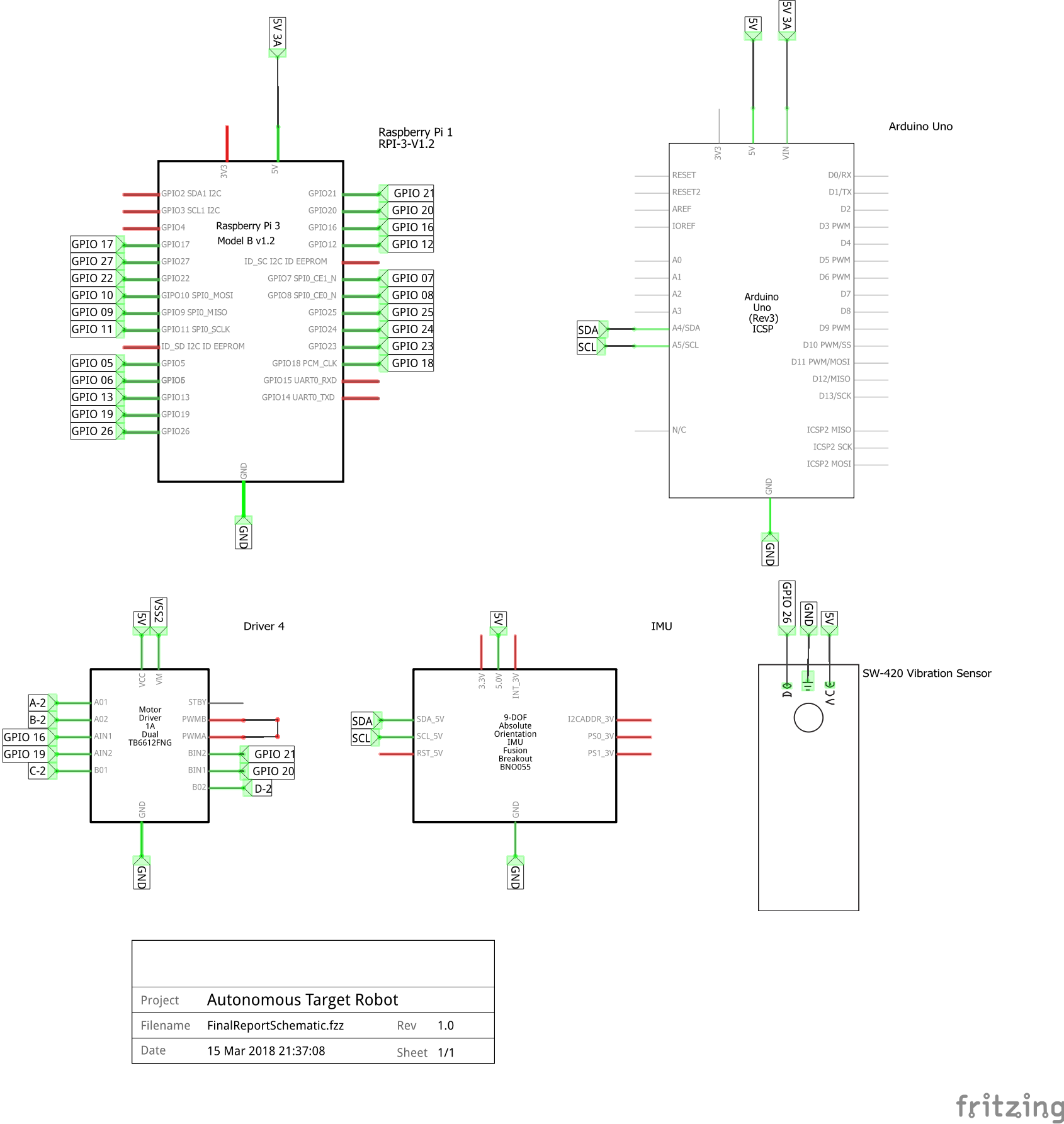
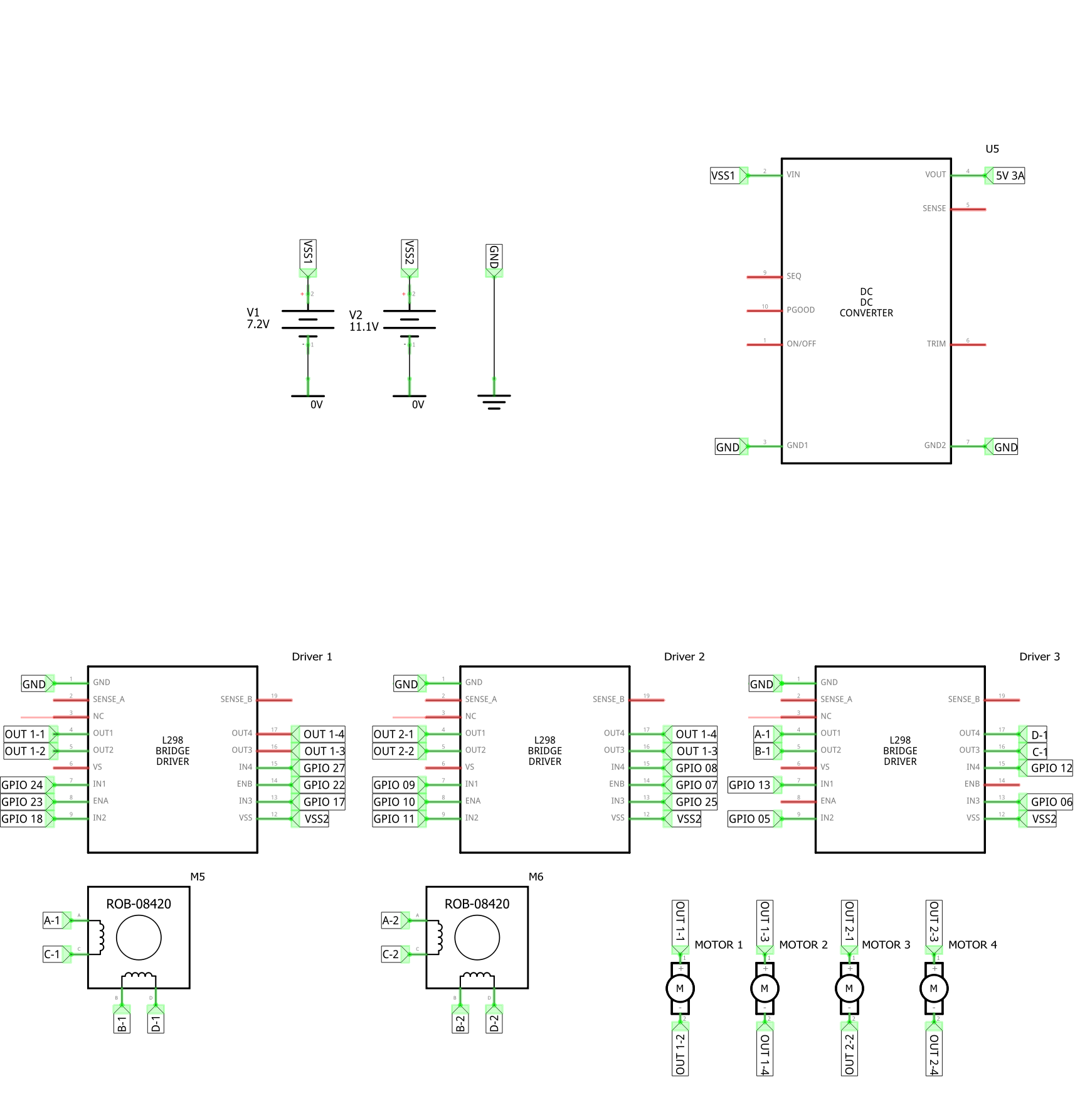
This section discusses why you are using the architecture or approach you have decided upon. A discussion of other architectures or approaches considered should be presented here.

Our first block diagram consisted of delegating the motor control, which encompassed the target functionality and movement, to the Arduino. The Raspberry Pi would then be solely responsible for GPS, WiFi connectivity, and the logic to configure our path and registering target hits. This proved to be difficult because we would have to interface between the Arduino and Raspberry Pi via UART. Sending all this data over UART was not a practical solution because of transfer rates and wiring. Therefore, we decided to implement the entirety of the project excluding the IMU onto the Raspberry Pi. We utilized the GPIO pins on the Raspberry Pi to integrate the DC motor control, stepper motor control, and target detection. On the USB ports we placed the GPS module, WiFi adapter, and the USB cable that connects the Arduino. Although this approach resulted in cluttered wiring, it was the method that provided the best results for our project.

# Low Level Design

This section provides low-level design descriptions that directly support construction of modules. Normally this section may be split into separate documents for different areas of the design. For each component we now need to break it down into its fundamental units or modules. Each module or block may be hardware or software or a subsystem implemented using hardware and software. Make sure to provide a well-commented schematic of all modules and/or system blocks described in the system block diagram (SBD).





## gui.py

gui.py is designed to create an input method for our robot. The GUI begins by asking the user for coordinates. The coordinates that are asked for are the Northwest, Northeast, Southwest, and Southeast. By asking for the latitude and longitude coordinates of these points, it creates a boundary box. The user is also asked for how many random coordinates to be generated inside this boundary box.

## RandNum.py

RandNum.py reads the coordinates inputted from the GUI in a file called examples.txt. The random generator is seeded with time so that no 2 iterations can produce the same numbers. Depending on how many numbers the user specified to be produced, it runs a for loop to produce a random longitude and random latitude point. Once the numbers are generated, they are exported to a file called Coordinates.txt.

## angles.py

In angles.py, we are calculating the angles required for the Turret Rotation and for the robot to turn. We read information from Coordinates.txt and examples.txt to calculate these angles using the atan2 function. Each angles is calculated by using point A and point B and determining how many degrees from 0-360 we have to have the robot turn. After these angles are calculated, they are exported to a file called angles.txt.

## calibrateIMU Functionality

Calibrating the Inertial Measurement Unit (IMU) is a process inside the file demo.py. To calibrate it, we begin by flushing the IMU of 15 previous values and then begin reading in all the values that we need for the calculation. After we see how far the IMU is off by, we determine that offset and store it in a variable to use later on. The offset will be set after arriving at every destination to ensure that angular drift over time is accounted for.

## Turning Functionality

Using the Offset calculated in Calibrating the IMU, we add this drift to our current heading in order to “zero” the heading. The cause for such implementation is documented in Section 7.1, Calibration of IMU. This function turns the motors high in opposite directions to turn the robot about its z-axis. As the robot traverses, we poll the IMU, checking if we have reached our desired heading, after which we turn the motors back low to cease movement.

## Movement Functionality

Following the turning function, we refer to the destination generated by RandNum and our current GPS location to determine the distance between both points. We then turn on the motors high at a randomized duty cycle. While the robot is moving, the distance formula is updated through repeated polling of the GPS location until a threshold is reached. Once the chassis is in the desired location, the motors are returned to a low state and movement is halted.

## armUp.py

armUp.py has predetermined steps to move the target a complete 90 degrees. The variable in this file is the time it takes to run the target up. It can do it instantly or it can take some time depending on the code. In line 40, changing the fraction will change the speed of the target popping up.

## armDown.py

armDown.py is predetermined to move the target down 90 degrees with a certain amount of steps. The variable in this file is the time it takes to run down the target. It can be done instantly or it can take its time depending on the user. Line 41 in the code is the determining factor of how long it takes for the arm to come down.

## platformCCW.py

platformCCW.py is the first file to begin the Turret Rotation mechanism. The steps to turn the platform is predetermined in the file angles.txt. That number is inputted into the for loop and then the motors are turned on in sequence to turn the motor by that many steps.

## platformCW.py

platformCW.py is the last file in the Turret Rotation mechanism. The steps to return to the starting spot are the same from platformCCW.py. This file pulls from angles.txt and use those steps to run the for loop to turn the platform.

## vibration.py

In vibration.py, the vibration sensor records data period of 5 seconds. The user can extend this window of time to in one line of code to their liking. The vibration sensor reads if the target is hit during this time period and records how many times it has been hit. The number is then exported onto the terminal for the user to see the result.

# Technical Problem Solving

## The Calibration of IMU Problem

Upon testing the IMU in Section 4, Taylor found the IMU to have 6o of drift on execution, as well as a digital compass error which resulted in an offset of approximately 120-160o from North. The compass error was most likely caused by the relative proximity to metal components which comprise the majority of our chassis.

## Solving the Calibration of IMU Problem

This issue was tackled by adding a negative offset to all calculations involving the IMU dynamically. For example, if the initial reading was 120o, we would add -120o to every following IMU value to effectively change the relative readings to a more reasonable value.

## USB Buffer Issue

In section 4, Taylor also uncovered an issue with the serial buffer and USB buffer on the Raspberry Pi. With the preset 1115200 baudrate, the serial buffer experienced overflow, outputting hex values in place of the expected float values.

In addition to that it was found that both buffers would not flush themselves upon program completion, resulting in an average of 7 expired values followed by 3 zeroes to be output to the console. For example, the output would be 110.0, 110.0, 0.0, 0.0, 117.0, the last number being the most current reading.

## USB Buffer Resolution

Our resolution to the first baudrate issue was to optimize the baudrate. Samuel Choi’s previous experience interfacing with the Arduino expedited this process and found that a value of 9600 would effectively eliminate the overflow of the buffer and remove 2 of the 7 expired values in the second issue.

After extensive troubleshooting, Taylor found that manually flushing the IMU by repeatedly outputting values before performing any calculations would fully update the buffers to the latest values available from the Arduino. This effectively fixed this issue, at the expense of adding 1 second to the execution time; acceptable in our system runtime.

## The Battery Supply Problem

An issue we encountered during live testing of our navigation algorithms on battery power was the undercurrenting of the Raspberry Pi. The result of which was frequent disconnections of the WiFi modules as our program was executed, interrupting our SSH connection and halting all testing until connection was re-established. This was most likely a result of us trying to meet the minimum power requirements of the Raspberry Pi, 5V at 2.5amps, with a 5V 2 amp voltage regulator connected to our 7.2V NiMH battery.

## Solving the Battery Supply Problem

Resolution of this error was completed by utilizing a buck converter which expanded our power supply capacity to 5V 3 amps. This resulted in higher battery drain, shortening our battery life which was already more than sufficient for our application. We encountered no further issues with undercurrenting beyond that point.

## Chassis Limitations

A major issue we encountered in our program was the relative size and accessibility of the chassis. The original size of the vehicle was ¾ the current size of the robot and featured few mounting holes which could accommodate the stepper motors and the turret.

## Chassis Limitation Solution

The solution to this limitation was to expand the length of the chassis to both accommodate mounting holes as well as expand the size for our Raspberry Pi and Arduino. This was resolved by installing Tetrix channels from Taylor’s supply of robotics parts. This allowed a free range of mounting options from 6mm machine screws to simple zip ties across the entire chassis.

## Tracked Vehicle Problem

A major issue we encountered with our vehicle was the usage of tracks in place of conventional wheels. Under the assumption that a tracked vehicle would be able to challenge the majority of terrain, we designed our robot to traverse about its Z axis. However it became apparent upon testing that our motors were incapable of this due to the lack of torque they provided.

## Tracked Vehicle Resolution

Upon review, we have deduced this issue is currently unresolvable without a complete rebuild of the chassis. If given enough time, we would expand this chassis to accommodate more powerful motors as well as mechanum wheels, allowing us a higher top speed, as well as the original traversal capability we initially desired in our conceptual design.

## The WiFi Adapter Problem

After resolving our previous undercurrent issues, Taylor found through online research that a portion of the issues we encountered with our constant disconnection is the unreliability of the onboard Broadcomm WiFi module.

## Solving the WiFi Adapter Problem

Our solution to this inherent unreliability is to completely replace the functionality of the onboard chip with an external adapter. After implementing this fix, we completely resolved this issue, finally achieving a stable remote connection to the Raspberry Pi on the robot.

# User Interface Design

## User Interface Screens

To create a user interface for this robot, we resorted to Python and coded using Tkinter. Tkinter has functions implemented in its library to create modules. However, it took awhile to understand the library completely to store data and to have user input the data. The Graphic User Interface (GUI) opens up at the beginning of demo.py and has 9 boxes to fill out. The first 8 boxes ask the user for the boundary box. The last input is the number of random coordinates the user would like. After the user presses the button at the bottom of the window, the values are all stored into text file where it is accessible throughout the process of the script.

Francisco Munoz was responsible in computing and creating this GUI. The process was then debugged and tested by Samuel Choi and Taylor Che. At the beginning of the process there were some issues because the modules were not being imported correctly. We all had to do some research to match up the Python version and the Tkinter version and make sure all the correct function calls were being performed.

# Test Plan

## Test Design

**Tests and expected responses**

This test is a breakdown of our entire program into two separate parts, turning with the IMU and forward movement with GPS and turret implementation.

1. Test Case 1: IMU - Conducted by Taylor Che
2. The objective of this test to observe the results of our IMU and turning functions repeatedly.
3. The environment we will test this on tile, specifically that of the engineering lab.
4. We foresee an average IMU inaccuracy of 6o and a drift of 120-150o over the entire execution of the file. As we progress, we expect to further develop our method of zeroing the function.
5. We expect to encounter debugging issues in both logic and coding practices until we successfully complete the function without software crashes and to be able to easily turn about our current position in our environment.
6. Test Case 2: GPS - Conducted by Francisco Munoz and Samuel Choi
7. The objective of this test is to demonstrate our ability to navigate in a straight line to different GPS positions along its path and face towards our origin position.
8. We will conduct this test case outdoors, so that the GPS does not encounter any issues receiving data.
9. We foresee inaccuracy in our GPS position, caused by inherent inaccuracy of our GPS positioning as well as slight imperfections in our chassis which causes our robot to drift from its intended path.
10. Excluding the above predicted issues, we expect to successfully navigate to positions along our path, as well as be able to accurately account for the drift in straight line movement.

## Bug Tracking

Previous IMU issues have been ironed out to the best of Taylor Che’s abilities, inherent drift and offset values have been accounted for and the corresponding resolutions of these issues have already been implemented from past trials.

GPS issues have also been accounted for by both Francisco Munoz and Samuel Choi, drift and inaccuracy have been taken into careful consideration and adjustments to the execution have been accounted for in their software implementation of navigation.

## Identification of critical components

A recurring issue of our program is the structural integrity of the chassis. The robot has a tendency to loosen its own chassis screws while executing more intensive operations, such as performing turns on rough terrain.

# Test Report

## IMU

Taylor Che

**Test 1: Initial Readings**

Results:

In this test run, after coding the original function, Taylor found, when the robot was oriented towards North, the IMU was reading 122-130o instead of our expected 0o value. We found our 9600 baudrate to be adequate for our continued testing, and will keep it at our constant value.

Solutions:

We have determined an offset must be used to set the initial IMU reading to “zero”.

**Test 2: Offset Calculation and Multiple Turn Test**

Results:

In this sequential test run, the calibration of the IMU was implemented. On executions, only the turns following the first were executed. Upon viewing the values read from the IMU, hex values and zeroes were found instead of the expected float numbers. Additionally, on the second and following turns, there were instances where negative values were being received from the IMU after being offset. These negative values would not be functional in our code because the calculations done on finding the angles between the GPS coordinates would always return positive values.

Solutions:

The negative values were being output because we had already establish a predetermined offset within the Arduino code during the very beginning stages of testing the IMU. This offset was removed from the Arduino code and implemented onto the Raspberry Pi as a basic function that would determine if the value was negative and apply the appropriate value. Although this function was successful, the origin of the hex and zero values found upon running the first turn needed to be found.

**Test 3**

Results:

With the successful offset calculation, we were able to test the multiple turning functionality. Reading in the IMU values was accomplished via opening the serial port that the Arduino was connected to and converting the string into a float to be used for calculations. Although we were now able to turn successfully, hex and zero values that were found upon executing the first turn would cause the program to never accurately read the correct angle. In order to execute the program for the second turn, the serial port would have to be opened again and data values read in. What we realised is that everytime that we opened the serial port, there would be hex and zero values.

Solution:

After running this test case multiple times and conducting research about the transfer properties of the serial buffer on the Raspberry Pi, we determined that the serial buffer needed to be flushed of at least ten values before any coherent data could be read in.

**Test 4: Flushing Raspberry Pi buffer**

Results:

Our first attempt at flushing the buffer was not entirely successful. During the first few iterations of tests of determining the values in the buffer that needed to be flushed, we determined that there was an average of twenty values that were being sent over. We were able to reduce the number of erroneous values from twenty down to ten, by just changing the timing of the buffer clear but this was still leaving us with erroneous values.

Solution:

Within the Raspberry Pi, we implemented a for loop that would take in the remaining ten erroneous values and clear those values from the turning function. This for loop would execute every time a new turn was about to take place so that when the IMU data was being read in a compared to the already calculated angles between waypoints stored in an array, we would not run into errors. After this testing, we documented that the iteration through the array was successful but would require fine tuning. Because of the nature chassis, the turning of the bot would not be consistent and would sometimes skip the correct value. A threshold of 10o was needed as a range that the IMU data could be within from the actual calculated angle.

**Test 5:**

Results:

By testing the threshold limit from the previous test, we recorded that on the return turns would almost always be over/under turning. This was an interesting find because the on the first turns, we would be able to successfully turn to the correct angle and in theory by turning in the opposite direction by that same angle measure, we would be at our starting position. However, the over/under turning issue caused us much difficulty.

Solution:

To mitigate the error in over/under turning on the return function, the original angle threshold of 10o was increased to 15o. This five degree increase encompassed the necessary range of error that the return turn execution was introducing to our system. We also increased the baud rate on the Arduino from 9600 to 115200 to receive more values in the same period of time. This allowed for more readings and led to a precise reading of the IMU angle value.

**Overall Results**

By carrying out these tests, we were successfully able to calibrate and read values from the IMU. Because our IMU was connected to the Arduino and we were reading in the values through the serial port, we did have a very difficult time in the beginning with just starting to understand how to approach this issue. However, these tests allowed us to solve these issues albeit with some caveats. When we do call the IMU in between each call for turning, there is a delay during which the Raspberry Pi serial buffer is flushed. Out vehicle now has the ability to be within 5o of the intended target and return to within 5o ot the start location. We have gotten rid of all of the bugs in the execution and this code is now fully implemented into the main code without any errors.

## GPS

**Test 1:** **GPS Accuracy**

Results:

In this test we were looking to quantify the accuracy of our GPS module. To test the GPS module, we researched into the pynmea library as a method to parse and store the data from the GPS module. NMEA consists of sentences, the first word of which is the data type. This data type defines the interpretation of the rest of the sentence and each data type has its own unique interpretation of GPS data. For our purposes, we researched into GGA and RMC sentences. GGA provides essential fix data which provide 3D location and accuracy data while RMC provides essential gps pvt (position, velocity, time) data. We also attempted to use this data for determining heading.

Solutions:

In the end we decided on using the RMC sentence because it provided the best results for what we were looking for. The transfer rate was set to 4800 baud and at first this seemed like it would be too slow. However, Francisco was able to cut down the program enough so that the refresh rate of the GPS module was at 1 Hz. We also attempted to use RMC to find the heading to measure the turning angle.

**Test 2:**

Results:

With further testing using the RMC sentence however, we determined that the accuracy was not as good as we needed it to be. Indoors it was reading with an error of approximately 6m and outdoors it was giving us an error of about 1m. This would be considered relatively good in many other large scale applications but for our use, we would need to be able to move to points that are within that 1m threshold.

Solutions:

We eventually switched back to GGA and relegated the turning angle calculation to the IMU that we got debugged. The program would call the forward movement function, poll the GPS to gather the current location, and compare this to the target location. This is where the distance formula calculation would come in and set the flag to determine if the distance from the next location to the current location was within the acceptable 0.05 m range.

* 1. **Test 3: Turret Approach**

Results:

The distance formula calculations were carried out by first calculating what one meter would be translated to in terms of latitude and longitude. These constants were then stored as global variables which could be accessed each time the program was executed. As the program carried out and iterated through the array of randomly generated GPS coordinates, the distance calculation would take in the current GPS coordinate and the next GPS coordinate, calculate the difference in latitude and longitude by subtracting the current position from the succeeding position, use the pre assigned constants to convert the latitude and longitude values into meters, and finally use the mathematical formula for calculating distance to determine the actual value.

Solutions:

This test case was carried out at Lot 21 located in the northern side of UCR. We inputted the NW boundary point and the SW boundary point as a test case to determine if the calculations were accurate. We calculated the distance by using a tape measure to have a base measurement we could compare to. Once we executed the program, we determined that the distance between the two points was 19.0835 meters. The tape measure value we recorded was 19.05 meters. With this relatively low error, we successfully concluded that this approach to calculating distance was accurate.

* 1. **Vibration Sensor**

**Test 1: Reading Sensor Values**

Results:

The vibration sensor (SW-420) functions by sending a logical high or low into the GPIO pin it is connected to. For initial testing of the vibration sensor, we wrote a simple program that would detect the movement and output “Movement Detected” onto the terminal of the Raspberry Pi. The sensitivity was not set to the appropriate value and the sensor would pick up every little vibration.

Solutions:

The vibration sensor was integrated into the full code and the sensitivity was set so that only very hard hits (i.e. a bullet hitting the target) would trigger a response. Furthermore, to prevent bouncing of the chassis as registering as a hit on the target, we would not read in values from the sensor until the target arm was in position.

# Conclusion and Future Work

## Conclusion

This senior design project provided us with valuable hands on experience with all the skills we had acquired over the course of our education. We were tasked with developing a functional autonomous targeting vehicle that could be implemented as a high level training bot and we successfully fulfilled the requirements. Although there were many technical obstacles that we had to overcome in order to achieve this goal, our design and implementation functioned properly and allowed us to demonstrate the plausibility of this project. As detailed in previous sections of this report, there are many things that can be implemented to improve this project such as an improved chassis and more reliable components. However, within the given scope of this project, we accomplished the main goal that we were tasked with.

Samuel Choi Conclusion:

This project was a great measure of my skills I’ve gained as an Electrical Engineering student here at UCR. I was not familiar with the Python programming language and this was a good way for me to expand my programming language skills. GPS and IMU functionality was also something that I had to familiarize myself with in order to develop this successful project. I also developed great communication and time management skills.

Taylor Che Conclusion:

Our project was a learning experience for myself, as well as an apt test of my current skills as an Electrical Engineer. In the early stages of the project I familiarized myself with components which I have had limited interactions with, such as the IMU’s Kalman filter and the H bridge. This process furthered my understanding of concepts I have previously learned, as well as prompted me to delve further into similar topics to broaden the my capabilities as an engineer. In addition to hardware skills, I reeducated myself in programming practices and learned a new programming language, Python in the process. I feel that this project has made a significant impact in my professional and educational development.

Francisco Munoz Conclusion:

The amount of debugging and critical thinking pushed my limits of a Computer Engineer. This project demonstrated some key attributes that made me adapt and have a steep learning curve. Understanding the ins and outs of Python helped myself become a better coder and being able to think before hand before implementing the code. Developing time management skills more precisely and communication skills made the completion of this project possible. Overall, the magnitude of this project and amount of work required were key learning skills in life and in engineering.

## Future Work

The goal of this project was to provide the military with a piece of training equipment which would improve a soldier’s ability to engage with an unpredictable and mobile enemy; a situation more common in modern day combat environments. The objective of the conceptualization of this project was to improve the skills of the individual combatant as well as increase their survivability by improving their reactions to an elusive enemy.

Further development of this project, in our opinion, would be an implementation of computer vision, both as a method of obstacle avoidance and a potential approach to a mechanism that follows a manually dictated path for the robot to follow; ie: a line drawn in chalk. This would improve the functionality of the robot, as well as provide a solution to operations in indoor environments. In addition to this, a larger, fully sized version of this project would allow for man sized targets to be hoisted by the robot, as well as allow a more extensive and complex range of peripherals to be attached to this project.

In its current state, the relatively low cost of this product would be a primary marketing ploy. The robot is lower cost than other GPS-guided targeting solutions available to the public. Autonomy and unpredictability is another selling point of this project, as predetermined target placement in contemporary training cannot offer the benefits that our design provides.

## Acknowledgement

Be considerate and credit all those who have helped you in this project, especially credit the people who provided ideas or solutions.

We extend our thanks to our professor, Dr. Ping Liang and his assistants, Dr. Madhi Maaref and Dr. Kimia Shamaei for their guidance and encouragement in development of our project. Without their help in troubleshooting the various issues we encountered, such as leaving the motors high while polling the GPS, this project would have not been completed in the designated period of time. Their attentiveness to our needs and deadlines were a major factor in meeting our initial objectives and bypassing obstacles hindering our project.

We also thank our friends Matt Lumanatas and Baldomero Vargas for their guidance and attention. They served as extremely useful resources of hardware and programming knowledge and troubleshooting for many minor issues we encountered during our development.

# References

[1]K. Townsend, "Adafruit BNO055 Absolute Orientation Sensor", *Cdn-learn.adafruit.com*, 2018. [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-bno055-absolute-orientation-sensor.pdf. [Accessed: 16- Mar- 2018]

[2]"BU-353S4", *Usglobalsat.com*, 2018. [Online]. Available: http://usglobalsat.com/store/download/688/bu353s4\_ds.pdf. [Accessed: 16- Mar- 2018]

[3]"Tutorial - L298N Dual Motor Controller Module 2A and Arduino", *Hackerstore.nl*, 2018. [Online]. Available: https://hackerstore.nl/PDFs/Tutorial298.pdf. [Accessed: 16- Mar- 2018]

[4]B. Earl, "All About Stepper Motors", *Cdn-learn.adafruit.com*, 2018. [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/all-about-stepper-motors.pdf. [Accessed: 16- Mar- 2018]

[5]"Raspberry Pi Compute Module 3", *Raspberrypi.org*, 2018. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/computemodule/RPI-CM-DATASHEET-V1\_0.pdf. [Accessed: 16- Mar- 2018]

[6]L. Ada, "Adafruit TB6612 1.2A DC/Stepper Motor Driver Breakout Board", *Cdn-learn.adafruit.com*, 2018. [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-tb6612-h-bridge-dc-stepper-motor-driver-breakout.pdf. [Accessed: 16- Mar- 2018]

[7]"SW-420 Vibration Sensor Arduino Interface", *theoryCIRCUIT - Do It Yourself Electronics Projects*, 2018. [Online]. Available: http://www.theorycircuit.com/sw-420-vibration-sensor-arduino-interface/. [Accessed: 16- Mar- 2018]

# Appendices

**Appendix A:** Parts List

**Appendix B:** Software List (URL to online drive or SVN server, with sharing set to Public. Can omit this appendix if your project didn’t involving writing a program)

**Appendix C:** User's Manual - If your design requires instructions for future use, here is the place to put that information.

# Appendix A

Parts List:

1 - Raspberry Pi 3 Model B

1- Arduino Uno

3 - L298N Motor Drivers

1 - Adafruit BNO055 Absolute Orientation Sensor

1 - BU-353-S4 USB GPS Receiver

1 - 11.1V Lithium Polymer (LiPo) Battery

1 - 7.2V Nickel-metal Hydride (NiMH) Battery

2 - NEMA-17 Stepper Motor

1 - Adafruit TB6612 Motor Driver

1 - SMAKN DC/DC Converter 12V Step Down to 5V/3A Power Supply Module

1 - SW-420 Vibration Sensor

# Appendix B

Software List:

[GitHub Repository](https://github.com/fmuno003/SeniorDesign)

*Raspbian Operating System:* Raspberry Pi

*Arduino Genuino:* Arduino Interfacing

# Appendix C

User Manual:

1. Connect to the onboard Raspberry Pi through a Hamachi server via SSH.

Note: Launch Xlaunch for access to the GUI

1. Execute demo.py to begin the program
2. Enter the boundaries of your desired target area in the GUI, your starting location is the South-West corner of this boundary, and enter the number of stops you want to execute. Opting not to enter information defaults to a square boundary located in the Winston Chung loading dock. Changing examples.txt changes this default.
3. The robot begins, self testing the drive motors by moving back and forth and executing one turn.
4. It will navigate to random points in the designated boundary, the randomized GPS destinations are available in coordinates.txt for debugging purposes.
5. The robot will execute a turn, outputting the current IMU heading to the console.
6. This will repeat n-times, n being the number of stops initially entered into the GUI.
7. Upon arriving at the final destination the robot will return to home base and cease operation until demo.py is re-executed.