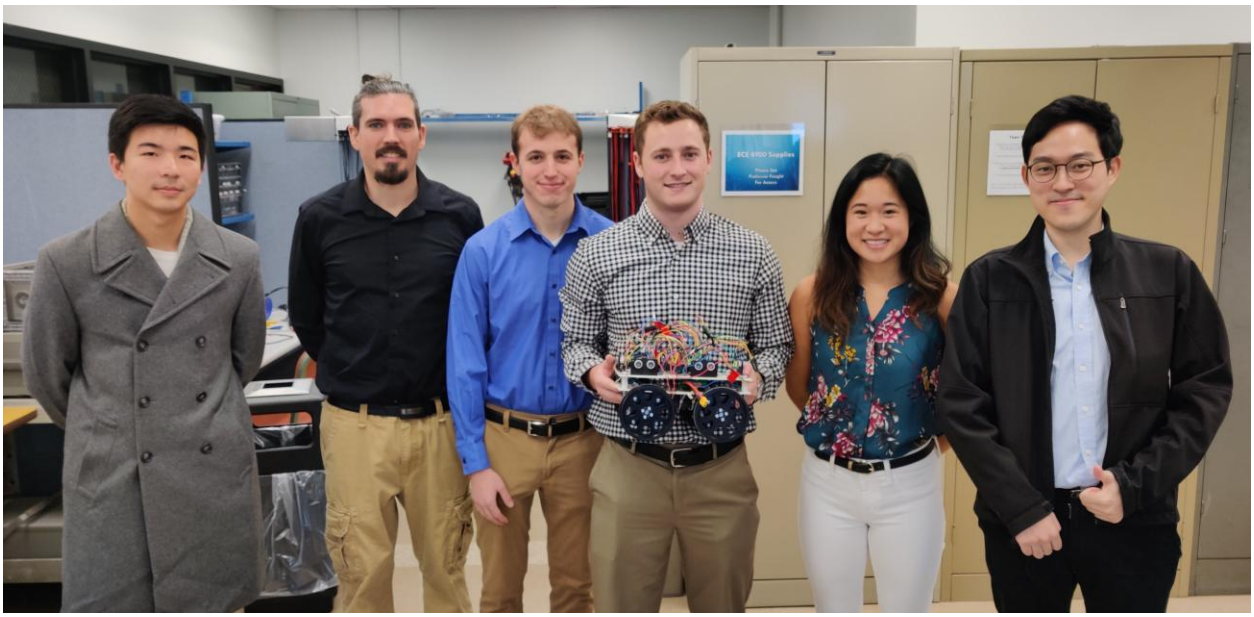
Critical Design Review Report

ECE 4900 Capstone Design II



Team 4 Members:

Ryan Hackney (hackney.16)
Devin Hensley (hensley.203)
Emily Kong (kong.250)
Yoon Jae Lee (lee.6650)
Matt Stoner (stoner.124)
Yuan You (you.189)

Due Date: 12/3/2019
Professor: Haskell Jac Fought

Table of Contents

Table of Figures:

Table of Tables:

Executive Summary

The project began with the problem statement, and a solution was developed to solve that statement. The team came up with a design for a UGV that would use sensors to navigate an area while mapping that area and finding the optimal path from one point to another. The original design consisted of a polypropylene frame with four stepper motors using Ultrasonic sensors to sense the area surrounding the UGV. While assembling and testing, problems came up that the team needed to address. The final design stayed relatively the same as the original design; however, the number of motors was reduced to make the robot more optimal.

This report goes into the process that the team followed while developing the design and build of the UGV. The proposal goes in-depth of the design for the UGV, and the potential coding structure that was planned for the final design. This includes the risks that the team predicted would arise as the project went forward. The team described the final design including the physical components, wire connections, and the code in the Raspberry Pi that was used to accomplish the task. The project schedule was affected by some unforeseen issues that arose, but the team allowed extra time in the schedule so the project was able to be delivered on time and within the required budget

1. Problem Statement

This product fulfills a demand for Unmanned Ground Vehicles (UGVs) that can navigate areas deemed potentially hazardous to humans. An autonomous vehicle would allow first responders or defense personnel to map out an unknown area, determine the optimized exit path, and potentially create a 2D blueprint for the end-user.

2. Requirement Specifications

a. The UGV will be able to turn in a 1-inch radius
b. The UGV will calculate the best path through the area in under 5 minutes
c. The UGV will cost under $500 to produce
d. The UGV will fit in a 30cm x 30cm x 30cm cube
e. The battery life will last a minimum of 60 minutes
f. The UGV will locate walls and obstacles and avoid them with 5” of clearance
g. The UGV will travel at approximately 1 m/s
h. The UGV will navigate an area with an incline of less than 3 degrees
i. The UGV will determine the quickest path out of all the options observed
j. The UGV will navigate and map one complete area before maintenance is needed

3. Analysis of Constraints

The entire project needed to be designed to be within the budgeted $500 while still meeting all the engineering requirements. Possible shock hazards and pinch points in the design needed to be considered to reduce the risk to the user of the Autonomous Unmanned Ground Vehicle. The final product is required to complete the task without needing maintenance to be done in order for the desired reliability to be achieved. After the life cycle of the product is reached, all the components need to be disposed of in the proper way. Polypropylene is recycled at designation 5 [3], and electronics/battery need to be brought to the proper facilities to be recycled.

4. Standards and Regulatory Issues

As stated in ASTM International Standards in section F3200 - 18a, an Autonomous Unmanned Ground Vehicle (A-UGV) is defined as an “automatic, automated or autonomous vehicle that operates while in contact with the ground without a human operator” [4]. This is the definition that was followed to design the A-UGV. Since the design runs on a voltage less than 50 V, the device can be worked on while energized

without risk of serious electrical shock or burn according to NFPA 70E 130.2(A)(3) [1]. According to OSHA 1910.211(d)(44), a “pinch point” is any point that a part of the body can be caught between moving and stationary parts of the equipment [1]. The pinch points on the design are around the wheels of the A-UGV where fingers or other body parts could potentially be pinched against the chassis.

5. Design Concepts Considered

The team considered a number of different design concepts to fulfill the problem. Each of these concepts was conceived through an analysis of a few different options for key components that the UGV would need to have. The UGV was identified to need, at baseline: sensors for navigation, a microcontroller for memory stage and computation, a chassis for the body, and motors for navigation and mapping.

Different types of sensors considered included: SHARP GP2Y0A21K0F IR sensors, TFMini LiDAR sensor, and the HC-SR04 ultrasonic sensor. The main three factors for picking the sensor were measuring range, ease of use and price of the senor.

The SHARP sensor had a measuring range of 4 to 80cm this was decided that this range was feasible but more range would be more practical. The output of the SHARP is an analog voltage that is based on the angle of the reflected light that the sensor emits. This would allow for the 7 to 9 sensor that would be needed to easily be connected to a single microcontroller. The price of the sensor was $14.95 for a total of around $104.65–$134.55 depending on the amount of sensors that was settled on. This left the total budget spent to under $400, with a contingency budget of approximately $100.

The TFMini sensor measuring range was .3 to 12m which had a greatly improved range over the SHARP, but the measured range of under .3m was not accurate- an important feature for the UGV. The TFMini uses UART to communicate data, but with 7 or 9 sensors, the team would have to purchase a UART to I2C to allow all the sensors to communicate with the microcontroller. The cost of TFMini was $44.75 which makes it the most expensive out of the 3 sensor. The total cost would be $320.25 which would not leave enough budget for the rest of the project.

The team ended up choosing the HC-SR04 sensor (shown in Section I of the Appendix B), due to the fact that the range was midrange, at 4cm to 4m. The sensor has a digital output but also needs digital input meaning the microcontroller will need at least 14 I/O pins. The price was $3.95, but due to its availability from past projects using the same sensors, the sensors are already available at no cost to the budget. Because these

sensors were available immediately, the team was also given the most exposure to interfacing with these sensors.

Several types of microcontrollers were considered: the Arduino UNO, Raspberry Pi 3/4 and MSP430 Launchpad. I/O pins and functionality were the two key factors the team analyzed when choosing between design concepts for this part.

The Arduino UNO has 14 digital and 6 analog I/O pins, which are just under the minimum amount of pins for either sensor. This meant that an I/O expander would have to be purchased. The Arduino is able to efficiently run a script multiple times, but the project would need to be able to run multiple scripts simultaneously. The Arduino also has many shields to increase the functionality of the microcontroller. The microcontroller was already available from past projects and would not need to come out of the budget.

Raspberry Pi has 24 I/O pins- more than enough for the minimum amount of pins needed. The Raspberry Pi has the ability to run multiple scripts because of its capability of having its own operating software. This added functionality, as it can allow programs to be run and even create or edit scripts, all on one Raspberry Pi. This controller was also available from past projects. Based on the factors from above, this was the microcontroller decided on to be the optimum choice for meeting the project's needs.

MSP430 Launchpad has 16 digital and 8 analog I/O pins that meet the minimum requirement but if more sensors had to be added then an expander would have to be purchased. MSP has low power draw but can only run one script like the Arduino. Making it lose functionality the project needs. This microcontroller would also add no cost to the budget as there were some from past projects.

6. Design Proposal

Shown below, in Figure 1, was the hardware block diagram of the UGV. The power fed into voltage regulators that step the power source down to the voltages required by each of the four motors, the seven ultrasonic sensors, and the Raspberry Pi. The Raspberry Pi then communicated to a laptop to relay important mapping feedback.

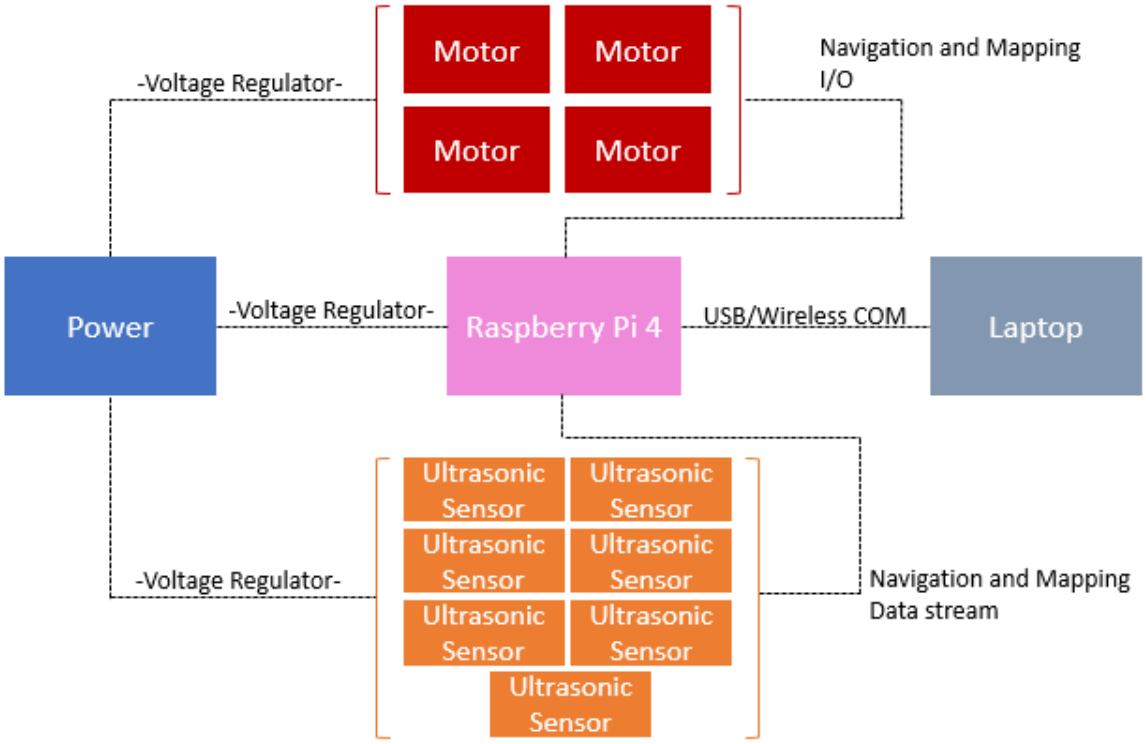


Figure 1: Proposed Project Block Diagram

The final design concept, shown below in Figure 2, originally included four DC stepper motors that drive up to four wheels simultaneously. The wheel diameter was determined to be greater than 5 inches to allow space for the DC stepper motors underneath the chassis. These motors were affixed to a 2-tier polypropylene pegboard chassis using motors mounts that were screwed into the chassis. The chassis dimensions were determined to be 22 inches by 18 inches and ⅛ inch thick for each sheet. The wheels were mounted to the shafts of each motor and secured with set screws. Ultrasonic sensors were utilized for both navigation and mapping of the vehicle, with the help of the HC-SR04 sensor. The vehicle included seven of these ultrasonic sensors with the team allowing for the potential use of more or less, dependent on future testing results. The Raspberry Pi was determined to be the best fit for the microcontroller and was used to implement autonomous navigation and mapping. With 24 input/output pins, it was satisfactory for controlling all the ultrasonic sensors as well as the motor drivers and the stepper motors. The Raspberry Pi used approximately 5 Watts (W) of power under load which was sufficient for extended battery life, while the DC stepper motors were the source of main power consumption. To account for the total high power consumption of this robot, the team recommended powering the four motors for 60 minutes, with the help of two batteries, to meet design requirements. Unit tests for each of the sensors individually, as well as running simultaneously, were also recommended before testing the entire robot together after the final assembly.

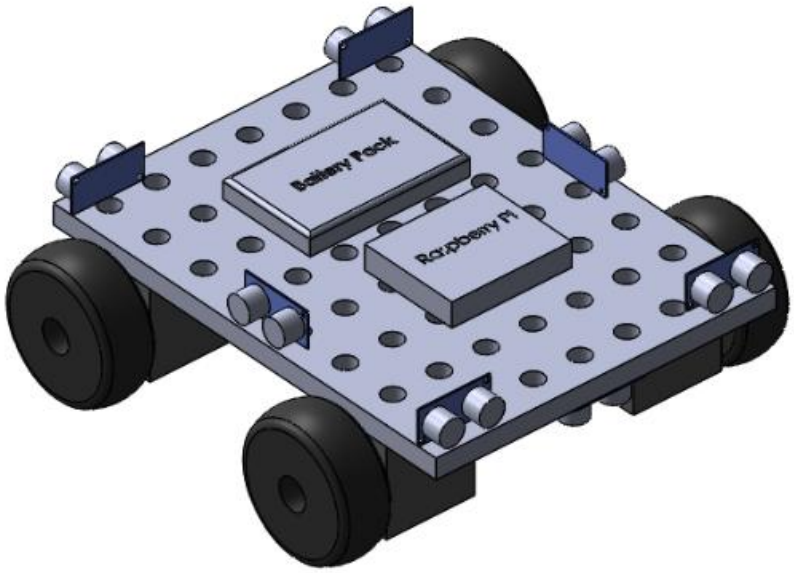


Figure 2: Design Proposal Build

7. System Schematics and Diagrams

For detail-level schematics of system parts and assemblies, please refer to Section II of Appendix B. Figure B3 shows the circuit for the motor drivers that uses the connections from figures B1(motor driver) and B2(motor). The circuit had 5 connections to each driver and one to the Raspberry Pi that ground the whole circuit. The whole circuit was grounded, as each of the 5 connections (EN, M1, M2, M3 and GND ports) all need to be zero to enable the FETs to drive motors and have full step resolution. The rest of the connections for the motor driver were A+, A-, B+, and B- which connect to the motor by red, blue, green, and black wires respectively.

The wiring diagram for each of the ultrasonic sensors (Figure B4) is shown in Figure B5. The top half rail of the circuit had an output from the Raspberry Pi from Vcc. This rail provided power to all the sensors. The next rail had an output from the Raspberry Pi that connected to the trigger pin, so all the sensors were triggered at once. The connections below the trigger rail were from each Echo pin on the sensors. This sent a signal to the Raspberry Pi after going through a voltage divider of 1k and 2k resistors. The last rail had an output from Raspberry Pi from GND to ground all the sensors and for the voltage divider. The overall UGV schematic can be found in Figure A6.

8. Software and Algorithms

The team developed a pseudocode, shown below, before beginning the processing of programming the mapping portion of the project. After writing this pseudocode, the team went through multiple kinds of mapping algorithms before finishing the final prototype

with the A* algorithm. More information on the theory of each of these algorithms may be found in Section III of Appendix B.

Pseudocode of Entire Process:

while Start is NOT Goal,

LOOP START:

1) Detecting Obstacle surrounding

2) Update map

3) Using A* or D* algorithm to search optimal path

4) Moving to next node along the optimal path

5) Mark current location as Start

LOOP END

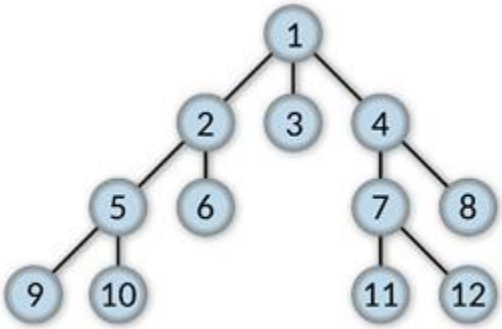


Figure 3: Breadth-First Search

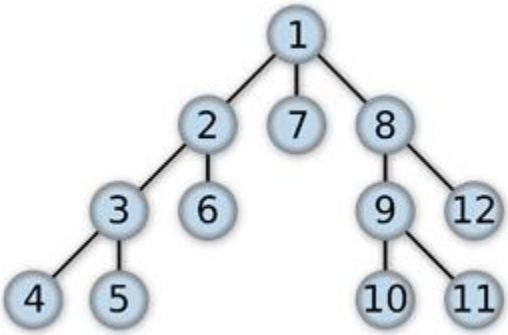


Figure 4: Depth-First Search

Shown in Figures 3 and 4, above, are two kinds of search algorithms: breadth-first search algorithm (BFS), and depth-first search algorithm (DFS). An algorithm called “Flood-Fill Algorithm” uses the concept of BFS or DFS (shown below in Figure 5). It starts searching from the starting point, first traversing the neighboring points around the starting point, and then traversing the neighboring points of the point that has been traversed, and gradually spreading out until the endpoint is found. It then uses backpropagation to following the cost decreasing, from the endpoint to the starting point, to find the optimal path.
One of the faults of this program is that it blindly searches for all possibilities, as opposed to considering the cost of each choice. Therefore, it has low efficiency.

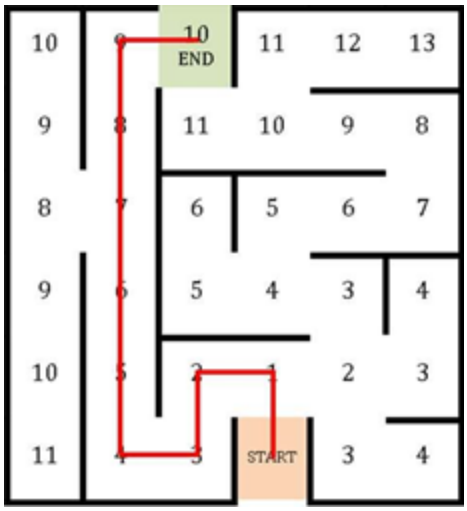


Figure 5: Flood-Filled Algorithm

In 1959, Edsger W. Dijkstra published his Shortest Path First algorithm, also known as Dijkstra's algorithm. Unlike the aforementioned strategy, Dijksra's algorithm considers the cost from the current node to the next node and the total cost in history. After comparing the cost of all possible choices, it then generates an optimal path through backpropagation. An example of Dijkstra's algorithm is shown below in Figure 6.

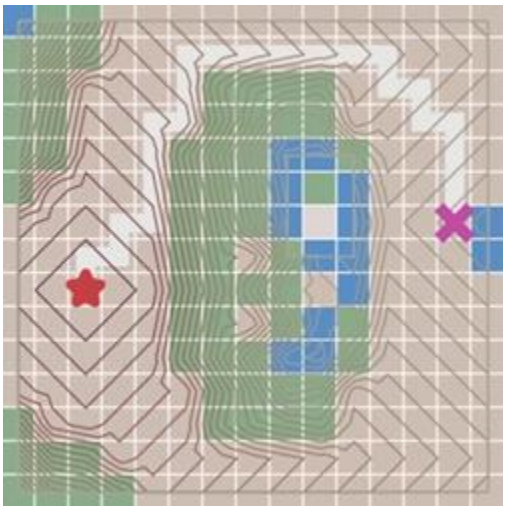


Figure 6: Dijkstra's Algorithm, considering the motion cost in mountain terrain

The goal of this algorithm was to find out the optimal policy by minimizing the total cost, and is based on the concept of dynamic programming. Since all the other path planning algorithms are based on Dijkstra's algorithm, additional information on how it works has been included in Section III of Appendix B.

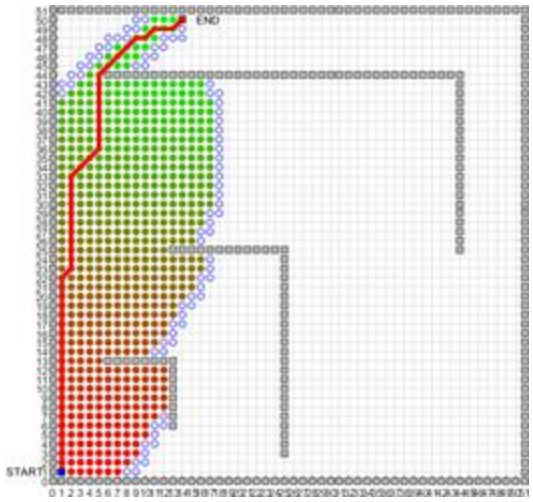


Figure 7: A* Algorithm, cost 56.14

Finally, the team worked toward the final mapping algorithm that was used for this prototype. In comparison to the previous two algorithms, the A* algorithm is faster than Dijkstra's Algorithm and maintains its optimality by having the same solution cost (the total length of path). Based on the theory discussed in the Mapping Algorithm Theory section of Appendix B, the most advantageous balance between velocity and **optimality**~~optimibility~~ is the A* algorithm itself. Figure 10, above, depicts an example of the A* algorithm.

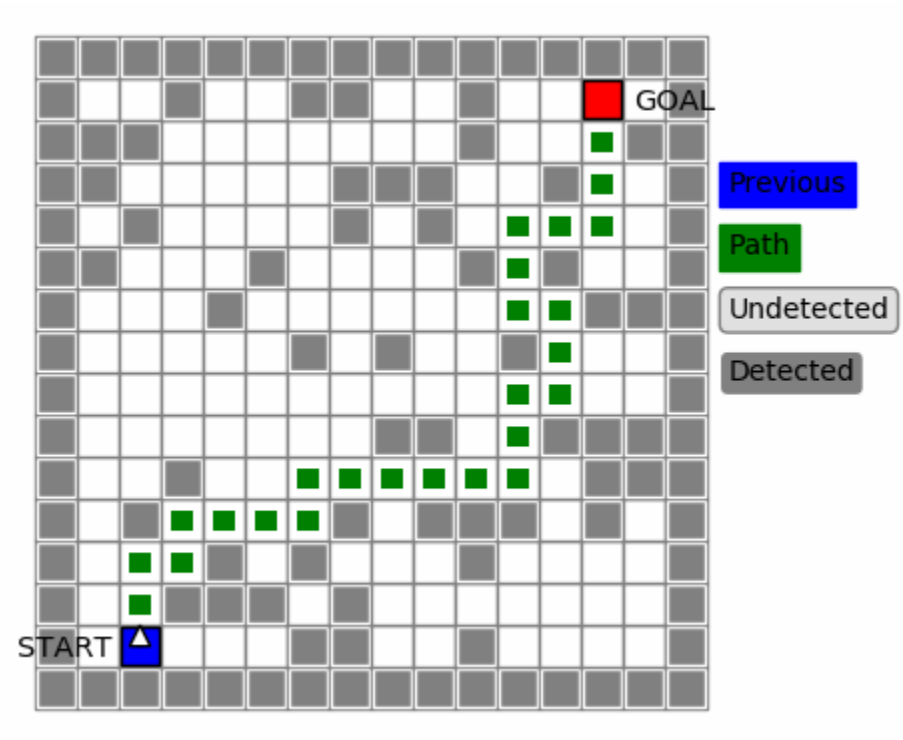


Figure 8: Dynamic A*

To solve the problem defined in the beginning of this project, the algorithm needed to have the ability to update map information in real time. This involved both the detection strategy, as well as the ability to continuously generate the optimal path to the exit, given information updates over time. This method is shown above, in Figure 11, and is the mapping algorithm the final prototype was programmed with.

9. Testing and Analysis Planning

In order to test the UGV along with each milestone, the team came up with a series of test plans for the overall robot as well as individual components and subsystems. The testing and analysis plans were as follows:

Pre-Build Testing and Preparation:

1. Test one ultrasonic sensor with the Raspberry Pi
2. Test all the ultrasonic sensors with the Raspberry Pi; gain familiarity for simultaneous data I/O
    a. Set the reference value of each of the sensors to calibrate
    b. Determine rotation rate correlation with distance
3. Test the motors with the Raspberry Pi
4. Test the sensors with the preliminary mapping algorithm

Test Plan for Navigation Trial Run:

1. Test a distance for each one step of robot navigation
2. Test a number of steps that the robot can rotate exactly 90 degrees
3. Test a number of steps that the robot can rotate exactly 180 degrees
4. Test each stepper can run correctly and all steppers can run simultaneously during multi-thread programming
5. Leave the robot running for 60 minutes to ensure a sustainable battery life that meets the requirements

Test Plan for Mapping Trial Run:

10. Implement and test the A* algorithm
    a. Test that the robot is able to communicate to the laptop and pass information back to the user on each of the distances and “nodes”
    b. The obstacles, previous location, the current location information stored inside Raspberry Pi for converting and updating data.
11. Test coordinate transformation (3D to 2D) by multiplying matrix
12. Test current attitude of the robot by doing dot product with minor bias
13. Test ultrasonic decision making (Priority and Pre-setting)

14. Test accuracy of an updated map with verifying the current existence of obstacles
15. Test real-time decision making and mapping from D* Algorithm. Using real-time detecting and updating map with ultrasonic sensors

10. Data and Testing Analysis

Ultrasonic Sensors ~~Test(~~Test (individual test):

1. Measured out various set distances measured in cm testing each sensor to find an acceptable error range from the set distances to be 5-15cm.
2. Measured distance apart from sensors by incrementing sensors close until their measurements started to interfere with each other to 10cm or greater.
3. By simply applying a sliding filter to avoid ultrasonic sensors detect something as an obstacle accidentally.

Ultrasonic Sensors and ~~Motors(~~Motors (Software):

1. Unit test for each part; stepper motors moving adjustment for forward movement of 8 directions and rotating movement of 8 directions~~.~~.
2. Adjust for alignment before hallway stand to assign reference and fixed data values.
3. Object-Oriented Programming approach with real-time feedback

~~3.~~

11. Final Changes and Finished Design

After initial product shipping, combined with the evaluation of the results from the testing detailed above, the team discovered multiple issues that had to be changed from the initial design concept for completion of a working final product. The first issue was availability of appropriately sized wheels. Because the robot needed wheels >5 inches in diameter to allow for the large stepper motors, specifically sized wheels and hub mounts were needed. The team was unable to find a pair of which neither was sold out and would arrive before the end of the project term date. To mitigate this concern, the team was able to imitate a similar design to the originally proposed wheel, and 3D printed individual wheels. This mitigation saved both time and money for the team.

The second issue was the space and cable management for each of the three batteries, along with the four motor drivers, soldered boards, and Raspberry Pi. To alleviate this problem, the team added spacers to increase the space between the two levels of polypropylene chassis, allowing for two batteries to sit in the middle layer of the robot. Large holes were drilled down through the top layer of the robot in strategic locations, which allowed for cables to be contained through the robot instead of hanging outside

the bounds of the UGV. Wires that led to the same device were labeled and soldered or taped together to prevent accidental shorts and disconnections while the robot was moving.

Because the robot initially utilized four stepper motors, the combined weight of the motors proved to be difficult to manage. During the testing runs, the loaded robot exhibited little success when trying to navigate turns and obstacles. After discussion, the team decided to replace the initial four-wheel-drive system with a two-wheel-drive system, switching the front two DC stepper motors out for mounted castor wheels. The castor wheel mounts were 3D-printed to the appropriate height of the robot and then drilled into the chassis.

The team encountered several issues with the initial design concept throughout the build and testing phases of this project. However, there were also aspects of the final product that were carried through with success from the initial design concept. One of these was the use of the polypropylene chassis. The chassis was sturdy enough to provide adequate support for the robot, and the two-layer structure allowed an additional room to store parts. The hole pattern of the polypropylene board also provided some mitigation for cable management. The ultrasonic sensors that were used proved to be accurate for the purposes of this problem and were user-friendly to work with. The DC stepper motors were another good choice because of their power, regardless of the terrain of the landscape. Lastly, the mitigation of using 3D-printed wheels saved both times and kept the design as it was originally intended. The team then used rubber tape to add cushion and friction for the robot to navigate easily over smooth surfaces. Figure 3 and Figure 4, shown below, illustrate the final logic block diagram as well as the final build of the robot.

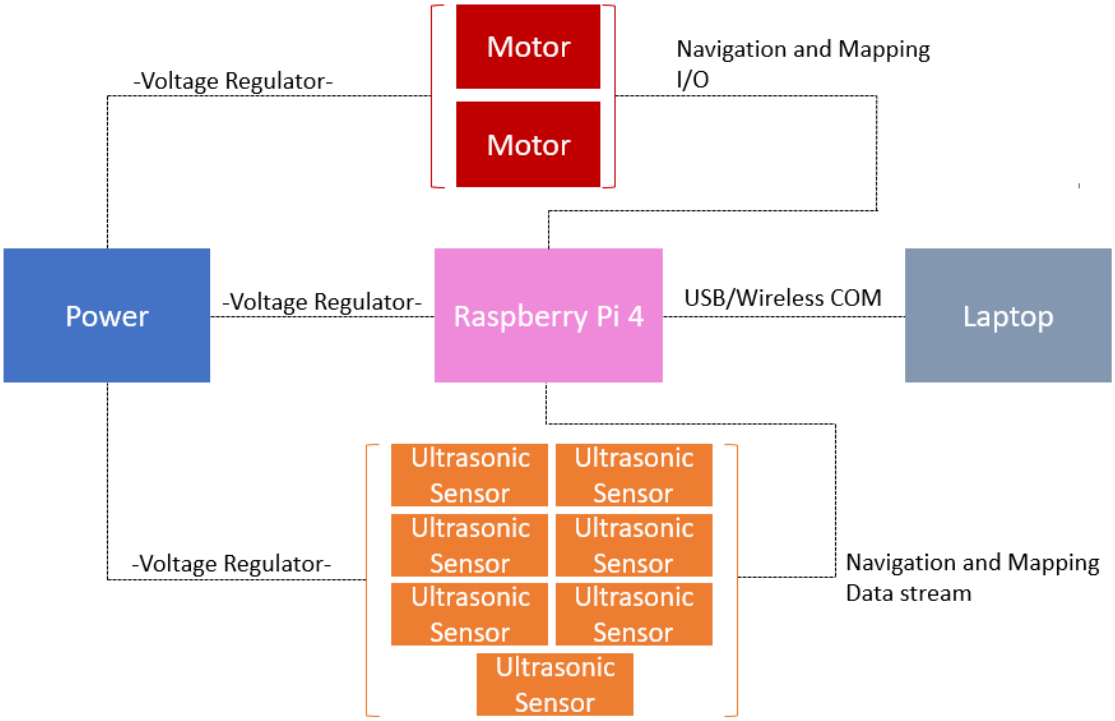


Figure 9: Final Design Block Diagram

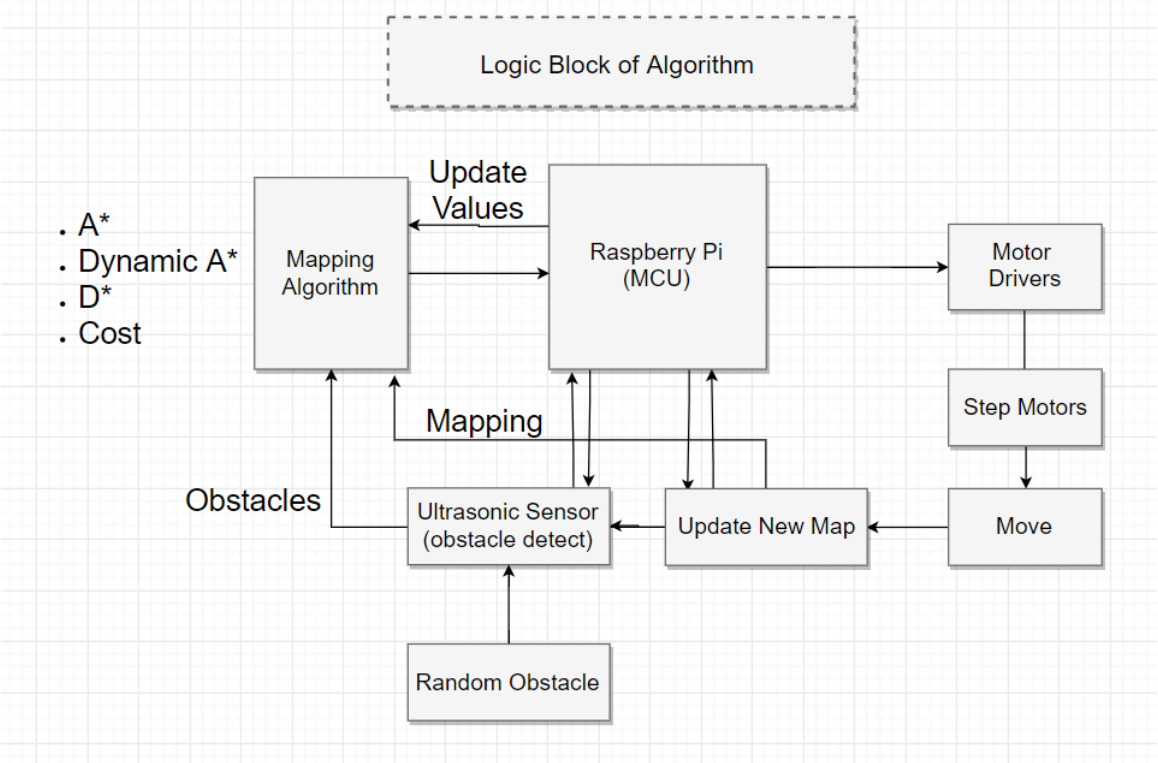


Figure 10: Final Algorithm Communication Block Diagram

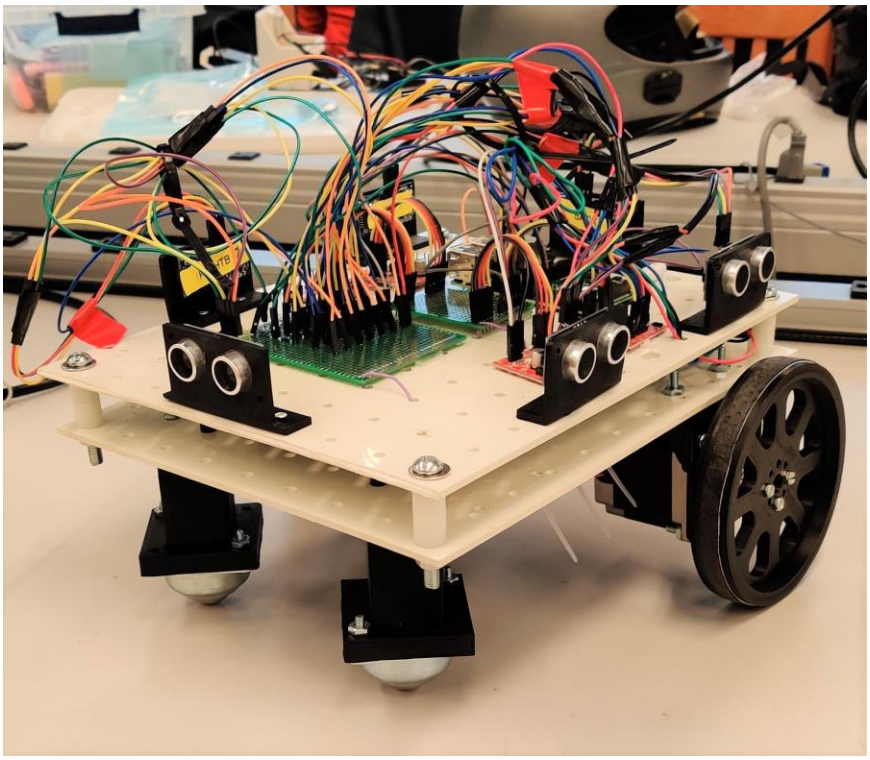


Figure 11: Final Robot Build

10. Schedule and Work Breakdown Structure

Each of the tasks that were needed to complete and reach milestones for this project was assigned to specific team members. These team members were responsible for taking the lead on these tasks, with the option of additional assistance from team members to push the deliverable. The task list along with its assigned work breakdown structure can be found in Appendix A. The team also developed a Gantt chart, which was used to track each of these tasks by the start and end date, along with the approximate amount of time each item would take to complete. This Gantt chart was updated through the process to reflect the team's current progress and adjust for project components that needed additional time. This chart can be found in Appendix A, following the Work Breakdown Structure.

11. Required Hardware, Equipment, and Facilities

This project required at least one empty room for testing. This testing room was then escalated to a room with obstacles that the robot had to avoid, and multiple exit paths. The team used empty classrooms and hallways in Dreese and Caldwell to test and troubleshoot the UGV. The team used a Raspberry Pi for feedback control and real-time decision making in mapping. The UGV required power management equipment to keep

a consistent power source to operate all the components for the required specification of 60 minutes. A laptop was used to communicate with the Raspberry Pi.

12. Budget

A number of key components needed to build this product were sourced from the supplies of previous capstone groups. This included the two DC stepper motors, seven to nine ultrasonic sensors, and a Raspberry Pi 4. Key items that the group purchased include the polypropylene board to construct a two-tier chassis, a battery pack for the Raspberry Pi, two gyroscopes, four motor drivers, four-set screw hubs, and miscellaneous wires and screws. Shown in Table 1, below, from the provided $500 budget, the team used $358.73, leaving $141.27 to spare. However, the team removed and replaced some of the design proposal items with alternatives to mitigate the challenges discovered in the testing phase. Additional parts were 3D-printed and used to hold the ultrasonic sensors in place. The group decided to move from the original four-wheel drive to two-wheel drive, eliminating weight concerns. In the final Bill of Materials, located in Section II of Appendix B, the amount of materials used to construct the final robot totaled to $221.74, or approximately $137 under the total budget spent.

Table 1: Budget Expenditure

Key Item	Amount From Budget
Stepper Motor (2)	$69.64
Stepper Motor Mounting Bracket (4)	$39.56
Polypropylene Chassis (2)	$41.83
Gyroscope (2)	$50.76
Motor Drivers (4)	$79.80
Power Bank (1)	$29.99
Set Screw Hub (4)	$19.96
Misc. (screws, spacers, etc.)	$27.19
Total Budget Spent	**$358.73**
Total Budget Remaining	**$141.27**

13. Conclusion and Recommendations

With the expanding market of autonomous vehicles and the sensationalism of drones, this product serves to combine some of the useful features of both. The development of an Unmanned Ground Vehicle (UGV) can be applicable to a variety of situations, whether it be through the defense sector, or a search and rescue mission. Where it might be unsafe for humans to venture, the design of this vehicle allows it to safely navigate areas with no prior knowledge of the scene it may be entering. Furthermore, this vehicle is capable of mapping out a room and determining the closest point of exit for personnel that may then head into the situation as a backup.

Since it is a primary prototype, there are numerous hardware enhancements worth pursuing for future work. Among these is optimizing the pairing of the motors and chassis material to be sturdy, but lightweight. While ultrasonic sensors were a useful first method of detection in the initial prototype, future groups should look toward using more robust methods of sensing, such as LIDAR. Cable management also played a huge role in the build of this prototype. Optimizing wire and cable management into the chassis design itself would prove more efficient for ease of use as well as assembling and disassembling. Overall, the budget played a limiting factor in the design concept and part choices the team decided on. Additional testing also showed that some parts were purchased that were unnecessary. With greater foresight, future groups could use these design and test challenges to optimize the budget available.

The software provided the real functionality of the robot without being limited by as many external factors, such as the budget. The software algorithm used could undergo further refinement to improve the efficacy and precision of the robot. Given improved consideration of possible ground turbulence or possible elevated surfaces built into the hardware, future software could work together with the hardware to optimize such features.

While the team made efforts to mitigate the in-aesthetic appearance caused by the lack of professional cable management, future work could be done to improve on the outward development of the robot. For example, a transparent dome-like fixture could be placed over the top of the robot, customized with the placement of the ultrasonic sensors. This would contain the cables while providing some protection from unexpected water or physical damage. In addition, the transparency feature could allow for visual feedback in the form of a color LED.

Overall, the team used its resources to develop the best prototype possible given the constraints at the time. While the robot was able to perform effectively, there are definitive measures that can be taken in the future to ensure a smoother design process that can allow for further optimizations and improvements.

Bibliography

1. Gray, Bobby J. “NFPA 70E - Proposed 2018 Edition.” *Nfpa.org*, 2017, www.nfpa.org/assets/files/AboutTheCodes/70E/Proposed_TIA_1265_NFPA_70E.pdf.
2. “Occupational Safety and Health Administration.” *1910.211 - Definitions.* | *Occupational Safety and Health Administration*, www.osha.gov/laws-regs/regulations/standardnumber/1910/1910.211.
3. “Polypropylene.” *The Association of Plastic Recyclers*, 2018, plasticsrecycling.org/pp.
4. Yoon, Soocheol, and Roger Bostelman. “Analysis of Automatic through Autonomous - Unmanned Ground Vehicles (A-UGVs) Towards Performance Standards.” *IEEE Xplore*, 2019, ieeexplore.ieee.org/document/8790421.
5. “A * Algorithm for Path Planning”, *Yunxi Community*, Alibaba Cloud, 9 Jan. 2019, https://yq.aliyun.com/articles/685477?utm_content=g_1000036267.
6. Koenig, Sven, and Maxim Likhachev. “D* Lite.” *Eighteenth National Conference on Artificial Intelligence*, 1 Aug. 2002, pp. 476–483.

Appendix A

Document Change Notice

Date	Change
10/8/2019	Initial Release
12/3/2019	Final Release (+): Team picture Ultrasonic sensor, motor driver and UGV schematics Updated design, budget, and schedule Testing process and results Design changes, Risks and Mitigations Logic for Mapping Algorithm Final BOM Python Code

Team Charter

Team name:
Go Go Power Rangers

The team mission is to successfully develop and design a device that provides a unique solution to a problem while keeping good engineering practice ethics. To make sure we meet all the requirements necessary for this project.

Team decision-making guidelines:
The initial line of decision will be made based on a majority vote with rating. In the event that there is a tie, each member of the team "scores" how strongly they vote for a particular design and the design with the higher score wins.

Meeting guidelines:
Each meeting will start with a predefined agenda. Each member of the group will contribute any deliverable updates since the last meeting and these objectives will be compared against the Gaant chart to measure progress. Members will discuss progress objectives for the upcoming week. After each meeting, a meeting minutes note will be sent out to recap over any discussions or resolutions. If a member wants to discuss a specific topic at a deeper level, the member will give a one-day notice to the other members of the group to ensure good time management.

Team roles:
Emily Kong: Analog power electronics
Devin Hensley: Digital, microcontroller
Yuan You: Control, MCU, DSP
Ryan Hackney: Microprocessor
Matt Stoner: Power and Control
Yoon Jae Lee: Control, Robotics, Avionics

Conflict resolution:
If any member(s) of the team have any conflicts with other member(s) of the team, a calm discussion will take place regarding team expectations and ways to solve the issue. Should the problem progress past these discussions to the point of hurting project objectives and going unresolved, the professor will be involved

Team member commitment:
As a member of team 4, I pledge to actively participate with the team in working toward completing all project goals and objectives by showing up to pre-scheduled meetings and completing all deliverables to facilitate product progress.

Signatures:
Emily Kong
Devin Hensley
Yuan You
Ryan Hackney
Matt Stoner

Work Breakdown Chart

TASK	ASSIGNED TO		PROGRESS	START	END
Milestones					
Robot Assembly Completed	All Team				
Navigation Trial Run					
Final Navigation Run					
Mapping Trial Run					
Preliminary Demonstration					
Finish Product					
Final Project Demonstration					
Physical Build					
Generate possible chassis designs and layout	Matt/Emily	Planned		9/17/19	9/25/19
		Actual	100%	9/17/19	10/8/19
Create BOM and Final Design Concept	Matt/Emily	Planned		9/25/19	9/26/19
		Actual	100%	10/8/19	10/15/19
Robot Assembly + Reconfiguration	Matt/Emily	Planned		10/11/19	10/18/19
		Actual	100%	10/22/19	11/19/19
Navigation + Mapping Code					
Choose Programming Language	Devin/Ryan (Lead) Matt/Emily	Planned		9/17/19	9/19/19
		Actual	100%	9/19/19	10/1/19
Research Navigation Processes	Devin/Ryan (Lead) Matt/Emily	Planned		9/20/19	10/10/19
		Actual	100%	9/20/19	10/10/19
Interface with 7 Sensors Simultaneously	Devin/Ryan (Lead) Matt/Emily	Planned		10/10/19	10/17/19
		Actual	100%	10/10/19	10/17/19
Create Preliminary Navigation Protocol	Devin/Ryan (Lead) Matt/Emily	Planned		10/11/19	10/18/19
		Actual	100%	10/17/19	11/5/19
Finish Navigation Protocol and Program Robot	Devin/Ryan (Lead) Matt/Emily	Planned		10/19/19	10/29/19
		Actual	100%	10/14/19	10/19/19
Optimize Navigation Protocol (if necessary)	Devin/Ryan (Lead) Matt/Emily	Planned		10/29/19	11/5/19
		Actual	100%	10/19/19	10/19/19
Program Algorithm to Find Optimal Path	Yuan/Yoon (Lead) Matt/Emily	Planned		10/18/19	12/1/19
		Actual	100%	10/2/19	10/27/19

Register Object/Wall Detection and Store in Memory	Yuan/Yoon (Lead) Matt/Emily	Planned		10/18/19	10/22/19
		Actual	100%	10/14/19	11/5/19
onvert Object Detectionto 2D Blueprint Drawing (machine readabl	Yuan/Yoon (Lead) Matt/Emily	Planned		10/22/19	10/27/19
		Actual	100%	10/18/19	11/10/19
Convert Machine-Readable Blueprint Drawing to User-Readable Blueprint Drawing	Yuan/Yoon (Lead) Matt/Emily	Planned		10/28/19	11/5/19
		Actual	100%	10/28/19	11/16/19
Optimize Mapping Protocol	Yuan/Yoon (Lead) Matt/Emily	Planned		11/6/19	12/7/19
		Actual	90%	11/6/19	
Documentation					
Schedule	All Team	Planned		8/20/19	9/12/19
		Actual	100%	8/20/19	9/10/19
PDR Presentation	All Team	Planned		9/27/19	10/4/19
		Actual	100%	9/27/19	10/4/19
Design Proposal and Planning Report	All Team	Planned		10/2/19	10/8/19
		Actual	100%	10/2/19	10/8/19
CDR Presentation	All Team	Planned		10/8/19	11/21/19
		Actual	100%	10/14/19	11/21/19
CDR Report	All Team	Planned		10/8/19	12/3/19
		Actual	100%	11/26/19	12/3/19

Gantt Chart

Project Planning Gantt Chart

Team 4
Team: Ryan Hackney, Devin Hensley, Emily Kong, Yoon Jae Lee, Matt Stoner, Yuan You

SIMPLE GANTT CHART by Vertex42.com
https://www.vertex42.com/ExcelTemplates/simple-gantt-chart.html

Project Start: Tue, 9/10/2019
Display Week: 1

TASK	ASSIGNED TO		PROGRESS	START	END
Milestones					
Robot Assembly Completed	All Team				
Navigation Trial Run					
Final Navigation Run					
Mapping Trial Run					
Preliminary Demonstration					
Finish Product					
Final Project Demonstration					
Physical Build					
Generate possible chassis designs and layout	Matt/Emily	Planned		9/17/19	9/25/19
		Actual	100%	9/17/19	10/8/19
Create BOM and Final Design Concept	Matt/Emily	Planned		9/25/19	9/26/19
		Actual	100%	10/8/19	10/15/19
Robot Assembly + Reconfiguration	Matt/Emily	Planned		10/11/19	10/18/19
		Actual	100%	10/22/19	11/19/19
Navigation + Mapping Code					
Choose Programming Language	Devin/Ryan (Lead) Matt/Emily	Planned		9/17/19	9/19/19
		Actual	100%	9/19/19	10/1/19
Research Navigation Processes	Devin/Ryan (Lead) Matt/Emily	Planned		9/20/19	10/10/19
		Actual	100%	9/20/19	10/10/19
Interface with 7 Sensors Simultaneously	Devin/Ryan (Lead) Matt/Emily	Planned		10/10/19	10/17/19
		Actual	100%	10/10/19	10/17/19
Create Preliminary Navigation Protocol	Devin/Ryan (Lead) Matt/Emily	Planned		10/11/19	10/18/19
		Actual	100%	10/17/19	11/5/19
Finish Navigation Protocol and Program Robot	Devin/Ryan (Lead) Matt/Emily	Planned		10/19/19	10/29/19
		Actual	100%	10/14/19	10/19/19
Optimize Navigation Protocol (if necessary)	Devin/Ryan (Lead) Matt/Emily	Planned		10/29/19	11/5/19
		Actual	100%	10/19/19	10/19/19
Program Algorithm to Find Optimal Path	Yuan/Yoon (Lead) Matt/Emily	Planned		10/18/19	12/1/19
		Actual	100%	10/2/19	10/27/19
Register Object/Wall Detection and Store in Memory	Yuan/Yoon (Lead) Matt/Emily	Planned		10/18/19	10/22/19
		Actual	100%	10/14/19	11/5/19
nvert Object Detectionto 2D Blueprint Drawing (machine read	Yuan/Yoon (Lead) Matt/Emily	Planned		10/22/19	10/27/19
		Actual	100%	10/18/19	11/10/19
Convert Machine-Readable Blueprint Drawing to User-Readable Blueprint Drawing	Yuan/Yoon (Lead) Matt/Emily	Planned		10/28/19	11/5/19
		Actual	100%	10/28/19	11/16/19
Optimize Mapping Protocol	Yuan/Yoon (Lead) Matt/Emily	Planned		11/6/19	12/7/19
		Actual	90%	11/6/19	
Documentation					
Schedule	All Team	Planned		8/20/19	9/12/19
		Actual	100%	8/20/19	9/10/19
PDR Presentation	All Team	Planned		9/27/19	10/4/19
		Actual	100%	9/27/19	10/4/19
Design Proposal and Planning Report	All Team	Planned		10/2/19	10/8/19
		Actual	100%	10/2/19	10/8/19
CDR Presentation	All Team	Planned		10/8/19	11/21/19
		Actual	100%	10/14/19	11/21/19
CDR Report	All Team	Planned		10/8/19	12/3/19
		Actual	100%	11/26/19	12/3/19

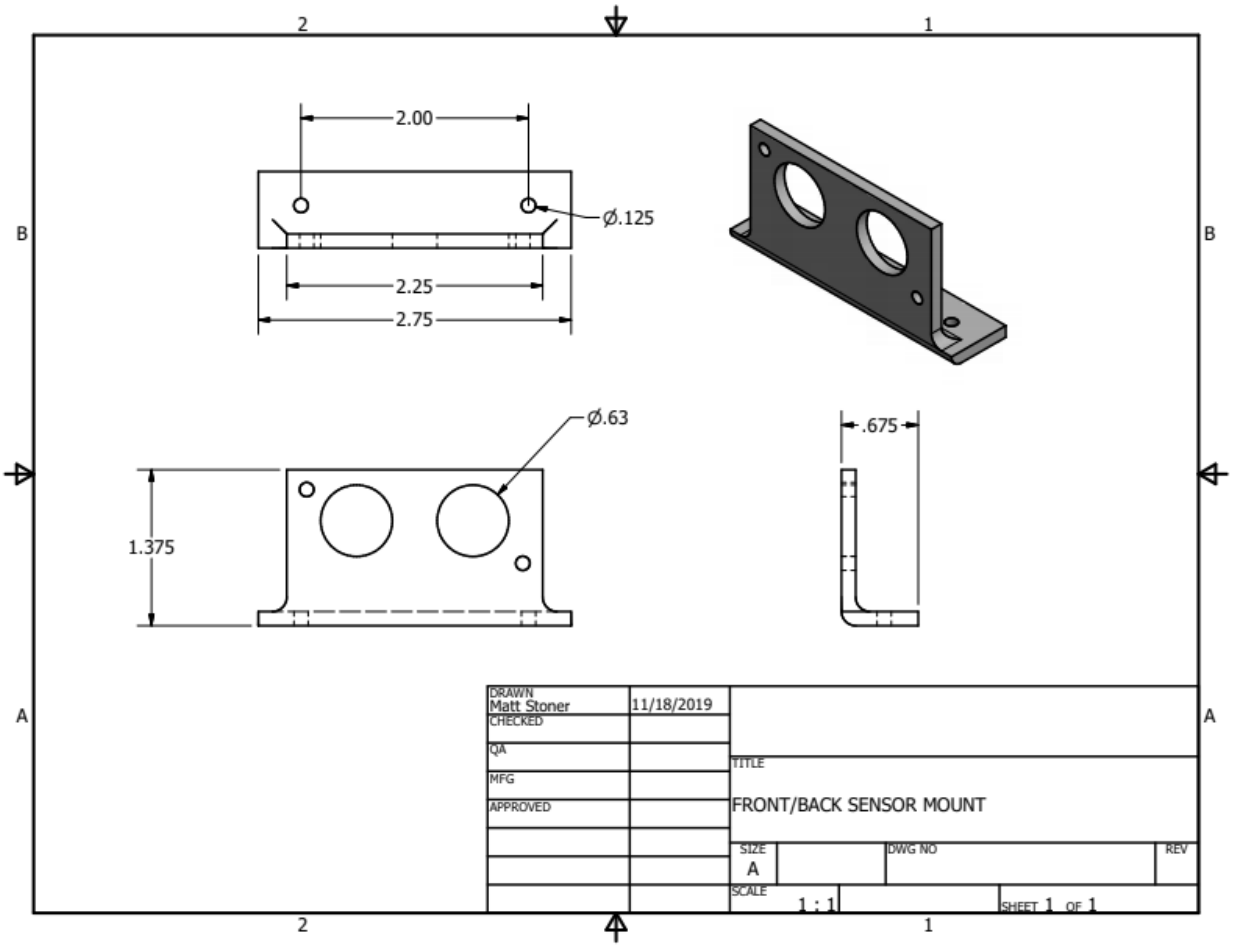
Appendix B

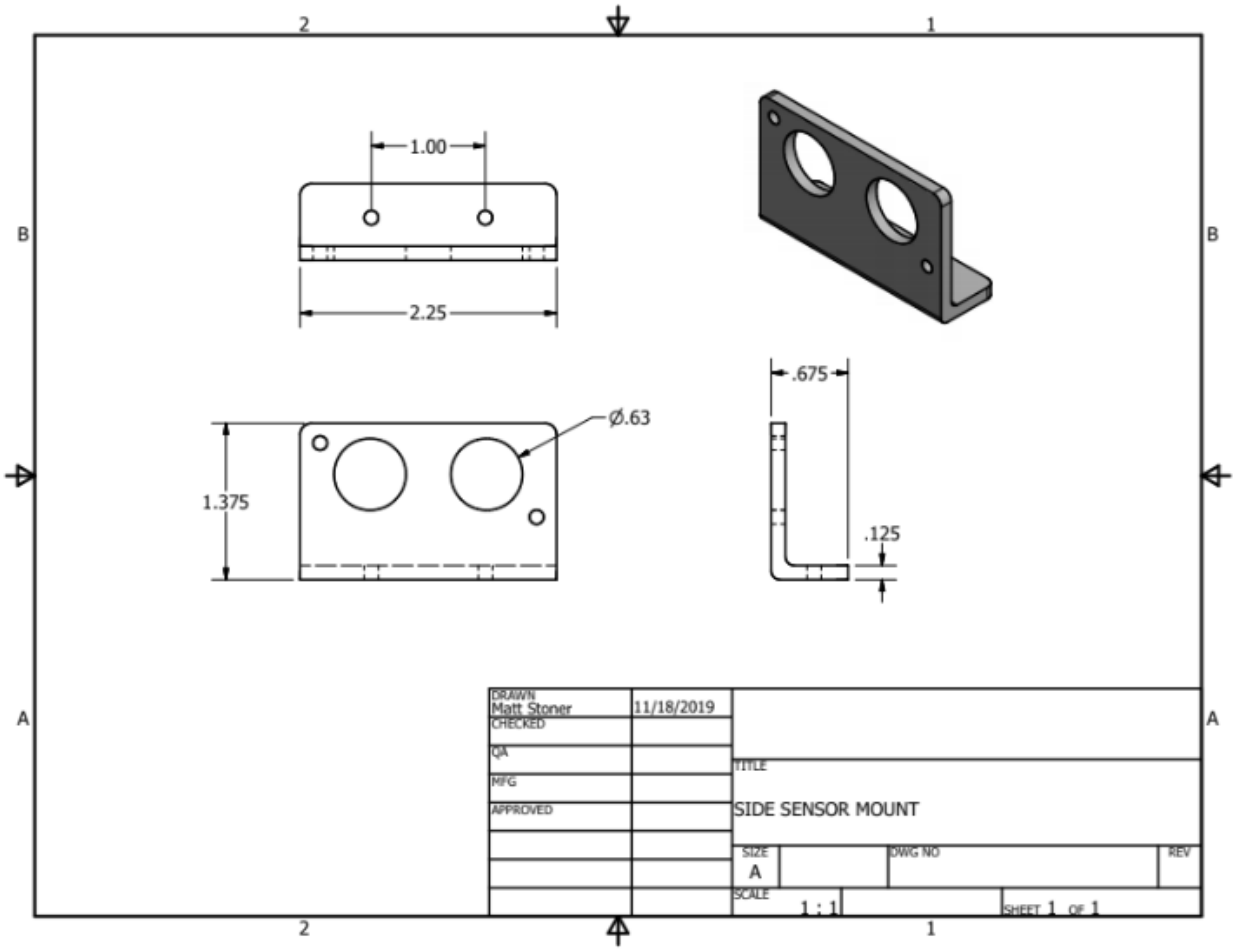
Section I: Physical Components

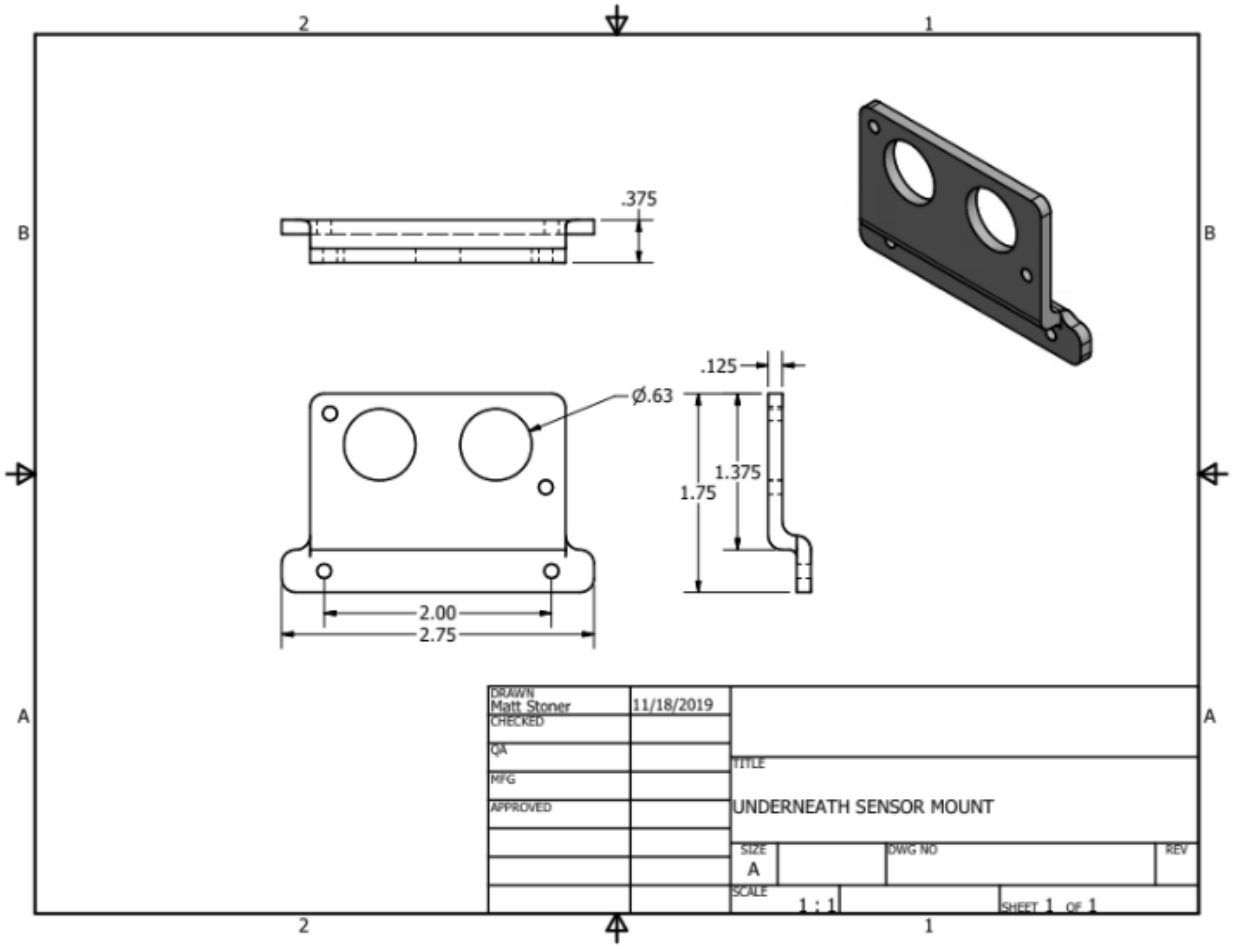
Bill of Materials

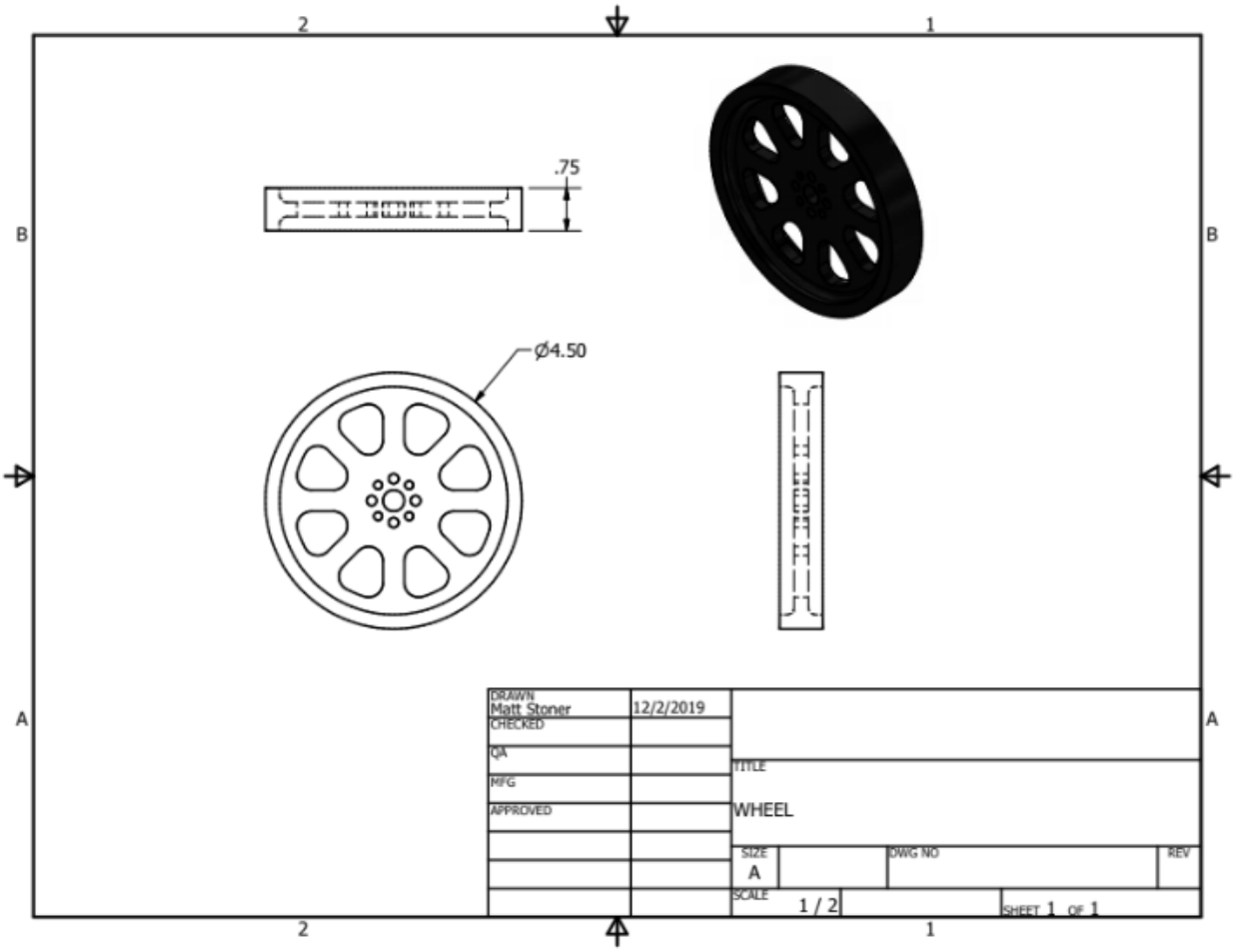
Part Description	Quantity	Price	Location Bought From
Polypropylene Pegboard (9"x11")	2	$41.83	Grainger
Stepper Motor NEMA 23	2	$38.69	Mouser
Stepper Motor Mounts	2	$19.78	Newegg
Motor Drivers	2	$39.90	SparkFun
1/2" Spacers	8	$1.79	Grainger
Ultrasonic Sensors	7	$0.00	Found in Lab
3D Printed Side Sensor Mount	4	$0.00	3D printed
3D Printed Front Sensor Mount	2	$0.00	3D printed
3D Printed Underneath Sensor Mount	1	$0.00	3D printed
Portable Bank, 5V	1	$29.99	Micro Center
Power Bank, 12V	1	$0.00	Found in Lab
3D Printed 4.5" Wheels	2	$0.00	3D printed
1/4" Wheel-to-Shaft Mounting Hub	2	$9.98	SparkFun
1/4"-28 Bolts, 2"	4	$2.20	ACE Hardware
#6-32 Bolts, 1/2"	16	$3.20	ACE Hardware
#8-32 Nuts and 3/4" Bolts	32	$5.98	Lowe's
#8 Washers	16	$4.98	Lowe's
Raspberry Pi	1	$0.00	Found in Lab
Prototype Board	2	$0.00	Found in Lab
Male Break Away Pins	2	$1.50	SparkFun
Small Heatsink	4	$7.80	SparkFun
#4-40 Bolts, 1/2"	100	$1.95	Grainger
#4-40 Nuts	100	$2.04	Grainger
Omni Wheel Mounts	**2**	$0.00	3D Printed
Roller Ball Bearing Casters	**2**	$10.13	Amazon
	Total	$221.74	

3D Printed Part Drawings




2
1
B
A
1.00
2.25
Ø.63
1.375
.675
.125
DRAWN
Matt Stoner
11/18/2019
CHECKED
QA
MFG
APPROVED
TITLE
SIDE SENSOR MOUNT
SIZE
A
DWG NO
REV
SCALE
1 : 1
SHEET 1 OF 1


2
1
B
A
.375
.125
Ø.63
1.375
1.75
2.00
2.75
DRAWN
Matt Stoner
11/18/2019
CHECKED
QA
MFG
APPROVED
TITLE
UNDERNEATH SENSOR MOUNT
SIZE
A
DWG NO
REV
SCALE
1 : 1
SHEET 1 OF 1



Electrical Schematics

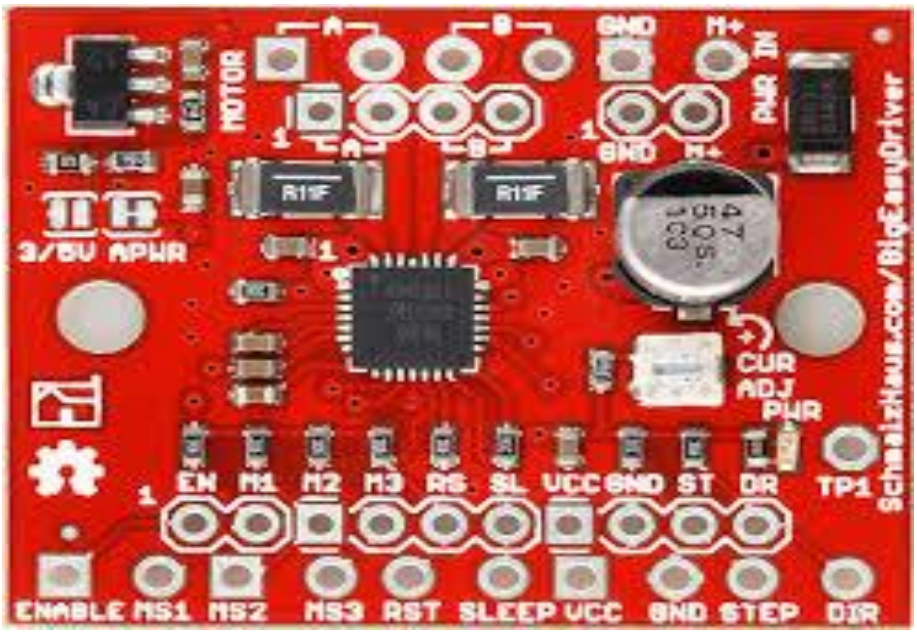


Figure B 1: Sparkfun Big Easy Driver

Figure B 2: Wantai Stepper Motor

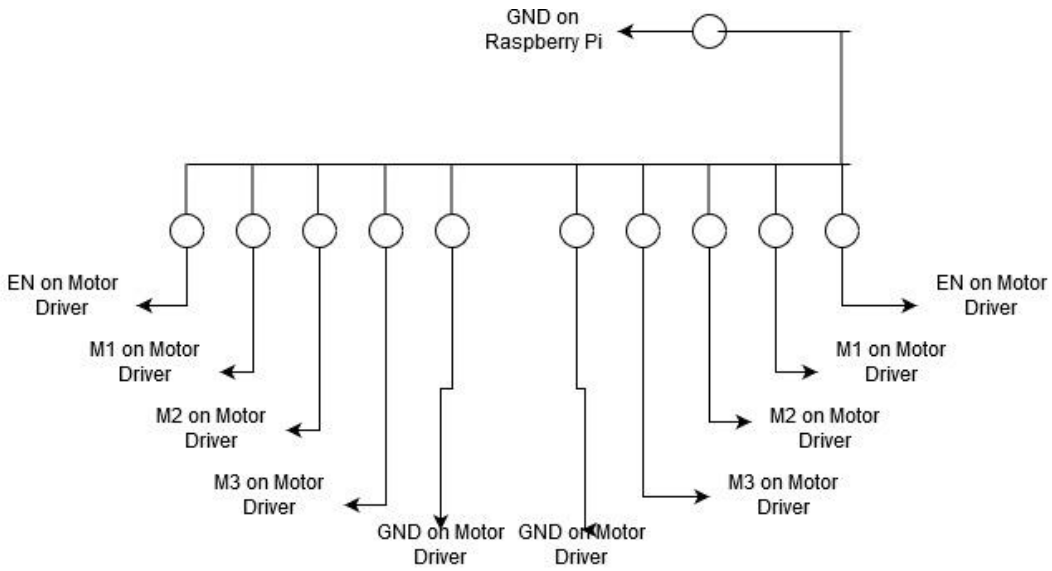


Figure B 3: Circuit for motor drivers

Figure B 4: HR SR04 Ultrasonic sensor

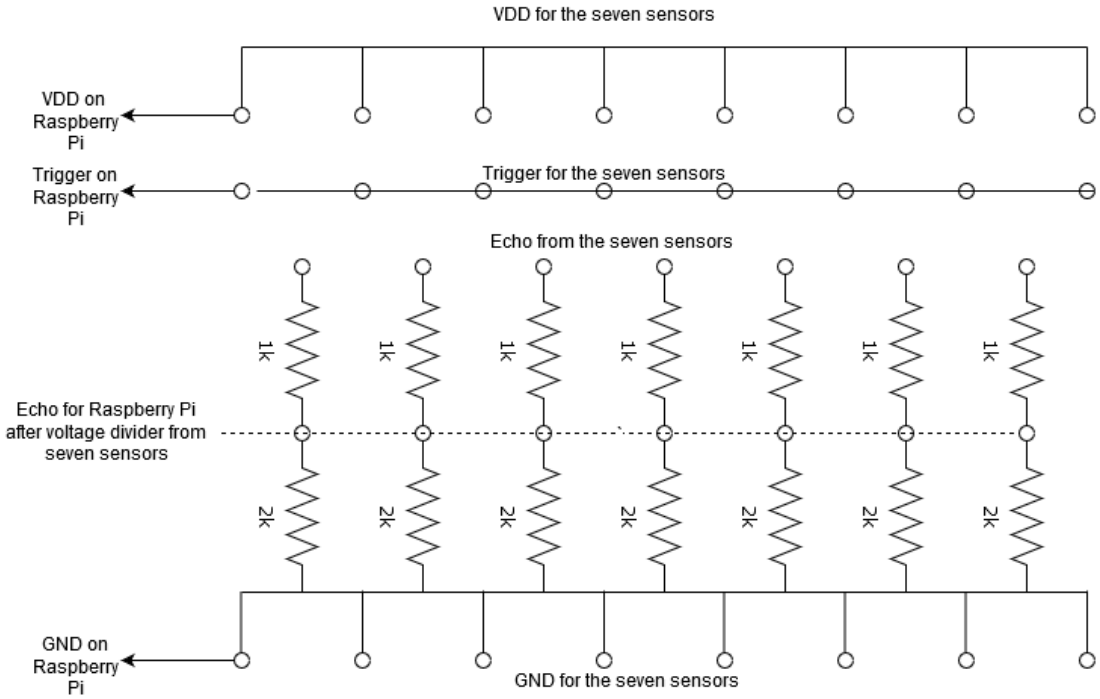


Figure B 5: Circuit for sensors

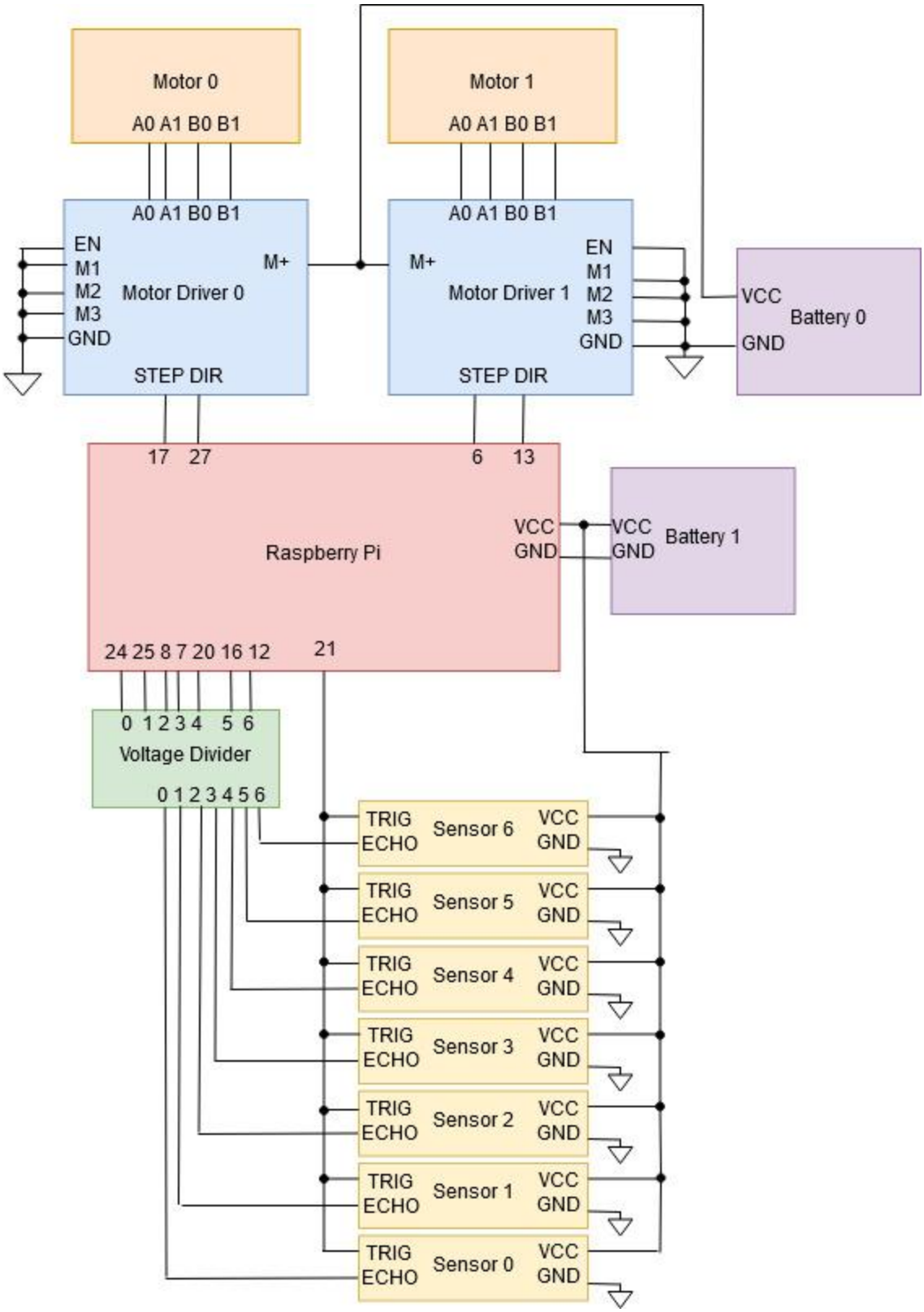


Figure B 6: UGV Schematic

Section II: Code

```python
#!/usr/bin/env python
# encoding: utf-8
from numpy import *

def H(a,b):
    D=1
    # # Manhattan Distance
    # return D*(abs(a[0,0]-b[0,0])+abs(a[0,1]-b[0,1]))

    # Eculidean Distance
    return D*(abs(a[0,0]-b[0,0])**2+abs(a[0,1]-b[0,1])**2)

if __name__ == '__main__':
    a=mat([[3,4,5]])
    b=mat([[4,5,6]])
    print("Unit Test - H cost: ", H(a,b))
```

```python
#!/usr/bin/env python
# encoding: utf-8
from numpy import *
from isSamePosition import isSamePosition

def isObstacle(m,obstacle):
    for index in range(0,len(obstacle[:,0])):
        if isSamePosition(obstacle[index,:],m[0:2]):
            flag=True
            return flag
    flag=False
    return flag

if __name__ == '__main__':
    obstacle=mat([[0,0],
                  [1,1],
                  [2,2],
                  [3,3],
                  [4,4],
                  [5,5],
                  [6,6]])
```

```
    e=mat([[6,6]])
    print("Unit Test - isObstacle: ", isObstacle(e,obstacle))
```

```
#!/usr/bin/env python
# encoding: utf-8
from numpy import *

def isSamePosition(a,b):
    result=False
    if a[0,0]==b[0,0] and a[0,1]==b[0,1]:
        result=True
    return result

if __name__ == '__main__':
    c=mat([[1,2]])
    d=mat([[1,2]])
    print("Unit Test - isSamePosition: ", isSamePosition(c,d))
```

```
#!/usr/bin/env python
# encoding: utf-8

import RPi.GPIO as gpio
import time

class led_flash:
    #initialize
    def __init__(self, green_led_pin, blue_led_pin):
        self.green_led_pin=green_led_pin
        self.blue_led_pin=blue_led_pin
        # gpio setup
        gpio.setmode(gpio.BCM) # Broadcom Mode, Index of Pin
        gpio.setup(self.green_led_pin, gpio.OUT)
        gpio.setup(self.blue_led_pin, gpio.OUT)

        gpio.output(self.green_led_pin,False)
        gpio.output(self.blue_led_pin,False)

    def toggle(self, led_pin, interval):
        # print(led_pin)
```

```
        while True:
            if led_pin=="GREEN":
                gpio.output(self.green_led_pin,True)
                time.sleep(interval)
                gpio.output(self.green_led_pin,False)
                time.sleep(interval)
            elif led_pin=="BLUE":
                gpio.output(self.blue_led_pin,True)
                time.sleep(interval)
                gpio.output(self.blue_led_pin,False)
                time.sleep(interval)

if __name__ == '__main__':
    print("LED MODULE SELF TESTING")
    gpio.cleanup()
    led=led_flash(17,27)
    led.toggle("GREEN", 1)
```

```
#!/usr/bin/env python
# encoding: utf-8
from numpy import *

def MotionModel():
    D=1             # x   y   cost
    next_move=mat([ [ 1,  0,  D*1],     # Move right
                    [ 0,  1,  D*1],     # Move up
                    [-1,  0,  D*1],     # Move left
                    [ 0, -1,  D*1]])    # Move down
                    # [ 1,  1,  D*1.414], # Move up-right
                    # [-1, -1,  D*1.414], # Move down-left
                    # [-1,  1,  D*1.414], # Move up-left
                    # [ 1, -1,  D*1.414]])# Move down-right
    return next_move

if __name__ == '__main__':
    print("Unit Test - MotionModel: ", print(MotionModel()))
```

```
#!/usr/bin/env python
# encoding: utf-8
from numpy import *
import time
from isObstacle import isObstacle
# from ultra3 import *

def Ultrasonic(path_map):
    left_flag=0
    right_flag=0
    up_flag=0
    down_flag=0



    # TRIG = 23                                   #Associate pin 23 to TRIG

    # FLOOR = 24                                   #Associate pin 24 to
ECHO
    # FRONT = 25                                  #Associate pin 24 to ECHO
    # RIGHT1 = 8
    # RIGHT2 = 7
    # LEFT1 = 20
    # LEFT2 = 16
    # BACK = 12
    # ultrasonic_echo_set=[FLOOR, FRONT, RIGHT1, RIGHT2, LEFT1, LEFT2,
BACK]
    # ultrasonic=UltraSonic_dev(TRIG,ultrasonic_echo_set)


    # # waiting for ultrasonic sensors
    # left_flag=ultrasonic.detect(direction='left')
    # right_flag=ultrasonic.detect(direction='right')
    # up_flag=ultrasonic.detect(direction='front')
    # down_flag=ultrasonic.detect(direction='back')


    # simulation
    left_position=mat([[path_map.current_position[0,0]-
1,path_map.current_position[0,1]]])
```

```
right_position=mat([[path_map.current_position[0,0]+1,path_map.current_pos
ition[0,1]]])

up_position=mat([[path_map.current_position[0,0],path_map.current_position
[0,1]+1]])

down_position=mat([[path_map.current_position[0,0],path_map.current_positi
on[0,1]-1]])
    if isObstacle(left_position,path_map.obstacle):
        left_flag=1
    if isObstacle(right_position,path_map.obstacle):
        right_flag=1
    if isObstacle(up_position,path_map.obstacle):
        up_flag=1
    if isObstacle(down_position,path_map.obstacle):
        down_flag=1


    return left_flag, right_flag, up_flag, down_flag
```

```
#!/usr/bin/env python
# encoding: utf-8
from numpy import *

def GetBoundary(map_size):
    boundary=mat([[0,0]])
    for i1 in range(1,map_size+2):
        boundary=vstack((boundary,[0,i1]))
    for i2 in range(1,map_size+2):
        boundary=vstack((boundary,[i2,0]))
    for i3 in range(1,map_size+2):
        boundary=vstack((boundary,[map_size+1,i3]))
    for i4 in range(1,map_size+1):
        boundary=vstack((boundary,[i4,map_size+1]))
    return boundary
```

```
if __name__ == '__main__':
    print("Unit Test - GetBoundary: \n", GetBoundary(5))
```

```
#!/usr/bin/env python
# encoding: utf-8
from numpy import *
from isSamePosition import isSamePosition

def FindList(m,open_list,close_list):
    if len(open_list):
        for index in range(0, len(open_list[:,0])):
            if isSamePosition(open_list[index,:],m[0:2]):
                flag=1
                return flag
    if len(close_list):
        for index in range(0, len(close_list[:,0])):
            if isSamePosition(close_list[index,:],m[0:2]):
                flag=2
                return flag
    flag=3
    return flag

if __name__ == '__main__':
    M=mat([[5,6,3,1,2]])
    OPEN_LIST=mat([[1,2,6],
                [6,6,6],
                [7,6,6],
                [7,6,6],
                [8,6,6]])
    CLOSE_LIST=mat([[3,4,6],
                [6,6,6],
                [7,6,6],
                [7,6,6],
                [5,6,6]])
    print("Unit Test - FindList: ", FindList(M,OPEN_LIST,CLOSE_LIST))
```

```
#!/usr/bin/env python
```

```
# encoding: utf-8
from numpy import *
from isSamePosition import isSamePosition
from GetBoundary import GetBoundary
from Ultrasonic import Ultrasonic


# WILL BE USED BY ULTRASONIC

def random_pick(some_list, probabilities):
    x = random.uniform(0,1)
    cumulative_probability = 0.0
    for item, item_probability in zip(some_list, probabilities):
         cumulative_probability += item_probability
         if x < cumulative_probability:
               break
    return item

def GetObstacle(path_map,mode):
    left_detect_flag=0
    right_detect_flag=0
    up_detect_flag=0
    down_detect_flag=0
    if mode=='random':
        # generate Obstacles

new_obstacle_cordinate=mat(random.randint(1,path_map.map_size+1,size=[path
_map.map_size*path_map.map_size,2]))
        # pick #num_of_obstacle of obstacles generated
        new_obstacle=new_obstacle_cordinate[0:path_map.num_of_obstacle,:]
        # remove Starting Point and Goal
        removed_list=[]
        for index in range(0,len(new_obstacle[:,0])):
            if
isSamePosition(new_obstacle[index,:],path_map.start_position) or
isSamePosition(new_obstacle[index,:],path_map.end_position):
                # Add Start/Goal to Remove List
                removed_list.append(index)
        # Remove the element in Remove List in row
        new_obstacle=delete(new_obstacle,removed_list,axis=0)
```

```
    elif mode=='detect':
        new_obstacle=mat([[0,0]])
        print("Ultrasonic is detecting...")

        # ultrasonic not created yet


left_detect_flag,right_detect_flag,up_detect_flag,down_detect_flag=Ultraso
nic(path_map)

        # simulation
        if left_detect_flag:
            temp_obstacle=mat([[path_map.current_position[0,0]-
1,path_map.current_position[0,1]]])
            new_obstacle=vstack((new_obstacle,temp_obstacle))
        if right_detect_flag:

temp_obstacle=mat([[path_map.current_position[0,0]+1,path_map.current_posi
tion[0,1]]])
            new_obstacle=vstack((new_obstacle,temp_obstacle))
        if up_detect_flag:

temp_obstacle=mat([[path_map.current_position[0,0],path_map.current_positi
on[0,1]+1]])
            new_obstacle=vstack((new_obstacle,temp_obstacle))
        if down_detect_flag:

temp_obstacle=mat([[path_map.current_position[0,0],path_map.current_positi
on[0,1]-1]])
            new_obstacle=vstack((new_obstacle,temp_obstacle))

        # # real situation
        # ultrasonic_detect_vector=mat(([left_detect_flag],
        #                               [right_detect_flag],
        #                               [up_detect_flag],
        #                               [down_detect_flag]))

        #
obstacle_in_map=path_map.trasnmit_matrix*ultrasonic_detect_vector
```

```
        # if obstacle_in_map[0,0]:
        #     temp_obstacle=mat([[path_map.current_position[0,0]-
1,path_map.current_position[0,1]]])
        #     new_obstacle=vstack((new_obstacle,temp_obstacle))
        # if obstacle_in_map[1,0]:
        #
temp_obstacle=mat([[path_map.current_position[0,0]+1,path_map.current_posi
tion[0,1]]])
        #     new_obstacle=vstack((new_obstacle,temp_obstacle))
        # if obstacle_in_map[2,0]:
        #
temp_obstacle=mat([[path_map.current_position[0,0],path_map.current_positi
on[0,1]+1]])
        #     new_obstacle=vstack((new_obstacle,temp_obstacle))
        # if obstacle_in_map[3,0]:
        #
temp_obstacle=mat([[path_map.current_position[0,0],path_map.current_positi
on[0,1]-1]])
        #     new_obstacle=vstack((new_obstacle,temp_obstacle))


        # remove first all-zero row
        new_obstacle=delete(new_obstacle,0,axis=0)
        # print(new_obstacle)

    return new_obstacle

if __name__ == '__main__':
    from PATHPLANNING import pathplanning
    map_size=5
    start_position=mat([[1,1]])
    end_position=mat([[4,4]])
    path_map=pathplanning(start_position,end_position,map_size)
    path_map.current_position=path_map.start_position
    path_map.start_position=mat([[1,1]])
    path_map.end_position=mat([[4,4]])
    path_map.obstacle=mat([[0,0],
                           [1,0],
                           [2,0],
                           [3,0],
```

```
                            [1,2],
                            [2,1],
                            [0,1]])
    path_map.num_of_obstacle=5
    print("Unit Test - GetObstacle: \n",
GetObstacle(path_map,mode='detect'))
```

```
#!/usr/bin/env python
# encoding: utf-8
from numpy import *
import time
from isSamePosition import isSamePosition

def GetPath(close_list,start):
    path=mat([[0,0]]) # Create an empty path, zero will be removed after
OPTIMAL PATH found
    index=0
    while True:
        path=vstack((path,close_list[index,0:2]))
        if isSamePosition(close_list[index,0:2],start):
            break

        for i in range(0,len(close_list[:,0])):
            if isSamePosition(close_list[i,0:2],close_list[index,3:5]):
                index=i
                break
    # remove first all-zero row from OPTIMAL PATH
    path=delete(path,0,axis=0)
    return path
```

```
#!/usr/bin/env python
# encoding: utf-8
import RPi.GPIO as gpio
import time

class stepper:
```

```
    def __init__(self, FRONT_LEFT, FRONT_RIGHT, BACK_LEFT, BACK_RIGHT,
ms1_pin, ms2_pin, ms3_pin, enable_pin, mode, step_time=0.0045):
        self.front_left_step_pin = FRONT_LEFT[0]
        self.front_left_dir_pin = FRONT_LEFT[1]
        self.front_right_step_pin = FRONT_RIGHT[0]
        self.front_right_dir_pin = FRONT_RIGHT[1]
        #self.back_left_step_pin = BACK_LEFT[0]
        #self.back_left_dir_pin = BACK_LEFT[1]
        #self.back_right_step_pin = BACK_RIGHT[0]
        #self.back_right_dir_pin = BACK_RIGHT[1]
        self.ms1=ms1_pin
        self.ms2=ms2_pin
        self.ms3=ms3_pin
        self.enable=enable_pin
        self.mode=mode

        # gpio setup
        gpio.setmode(gpio.BCM) # Broadcom Mode, Index of Pin
        gpio.setup(self.front_left_step_pin, gpio.OUT)
        gpio.setup(self.front_left_dir_pin, gpio.OUT)
        gpio.setup(self.front_right_step_pin, gpio.OUT)
        gpio.setup(self.front_right_dir_pin, gpio.OUT)
        #gpio.setup(self.back_left_step_pin, gpio.OUT)
        #gpio.setup(self.back_left_dir_pin, gpio.OUT)
        #gpio.setup(self.back_right_step_pin, gpio.OUT)
        #gpio.setup(self.back_right_dir_pin, gpio.OUT)
        gpio.setup(self.ms1, gpio.OUT)
        gpio.setup(self.ms2, gpio.OUT)
        gpio.setup(self.ms3, gpio.OUT)
        gpio.setup(self.enable, gpio.OUT)

        # initial
        gpio.output(self.front_left_step_pin, False)
        gpio.output(self.front_left_dir_pin, False)
        gpio.output(self.front_right_step_pin, False)
        gpio.output(self.front_right_dir_pin, False)
        #gpio.output(self.back_left_step_pin, False)
        #gpio.output(self.back_left_dir_pin, False)
        #gpio.output(self.back_right_step_pin, False)
        #gpio.output(self.back_right_dir_pin, False)
```

```
        gpio.output(self.ms1, False)
        gpio.output(self.ms2, False)
        gpio.output(self.ms3, False)
        gpio.output(self.enable, False)


        self.step_time = step_time
        self.steps_per_rev = 1600
        self.current_position = 0


    # def steps(self, step_count=1):
    #     print("Moving Forward")
    #   #当step_count为正数的时候, 设置dir引脚为低电平。否则为高电平。
    #     if step_count > 0:
    #         print("Moving Forward")
    #         gpio.output(self.dir, False)
    #     else:
    #         print("Moving Backward")
    #         gpio.output(self.dir, True)

    #     for i in range(abs(step_count)):
    #         gpio.output(self.step, True)
    #         time.sleep(self.step_time)
    #         gpio.output(self.step, False)
    #         time.sleep(self.step_time)
    #     self.current_position += step_count

    def move_F(self, step_count=200):#260
        print("FFFFFFFFFFFF")
        # DIRECTION
        # right side motor
        gpio.output(self.front_right_dir_pin, False)
        #gpio.output(self.back_right_dir_pin, True)
        # left side motor
        gpio.output(self.front_left_dir_pin, True)
        #gpio.output(self.back_left_dir_pin, False)

        # STEP
        for i in range(abs(step_count)):
```

```
            gpio.output(self.front_left_step_pin, True)
            gpio.output(self.front_right_step_pin, True)
            #gpio.output(self.back_left_step_pin, True)
            #gpio.output(self.back_right_step_pin, True)
            time.sleep(self.step_time)
            gpio.output(self.front_left_step_pin, False)
            gpio.output(self.front_right_step_pin, False)
            #gpio.output(self.back_left_step_pin, False)
            #gpio.output(self.back_right_step_pin, False)
            time.sleep(self.step_time)

    def move_L(self, step_count=130):
        print("LLLLLLLLLLLLL")
        # DIRECTION
        # right side motor
        gpio.output(self.front_right_dir_pin, True)
        #gpio.output(self.back_right_dir_pin, True)
        # left side motor
        gpio.output(self.front_left_dir_pin, True)
        #gpio.output(self.back_left_dir_pin, True)

        # STEP
        for i in range(abs(step_count)):
            gpio.output(self.front_left_step_pin, True)
            gpio.output(self.front_right_step_pin, True)
            #gpio.output(self.back_left_step_pin, True)
            #gpio.output(self.back_right_step_pin, True)
            time.sleep(self.step_time)
            gpio.output(self.front_left_step_pin, False)
            gpio.output(self.front_right_step_pin, False)
            #gpio.output(self.back_left_step_pin, False)
            #gpio.output(self.back_right_step_pin, False)
            time.sleep(self.step_time)

    def move_R(self, step_count=130):
        print("FRRRRRRRRRRRR")
        # DIRECTION
        # right side motor
        gpio.output(self.front_right_dir_pin, False)
        #gpio.output(self.back_right_dir_pin, True)
```

```
        # left side motor
        gpio.output(self.front_left_dir_pin, False)
        #gpio.output(self.back_left_dir_pin, True)

        # STEP
        for i in range(abs(step_count)):
            gpio.output(self.front_left_step_pin, True)
            gpio.output(self.front_right_step_pin, True)
            #gpio.output(self.back_left_step_pin, True)
            #gpio.output(self.back_right_step_pin, True)
            time.sleep(self.step_time)
            gpio.output(self.front_left_step_pin, False)
            gpio.output(self.front_right_step_pin, False)
            #gpio.output(self.back_left_step_pin, False)
            #gpio.output(self.back_right_step_pin, False)
            time.sleep(self.step_time)

    def move_U(self, step_count=259):
        print("UUUUUUUUU")
        # DIRECTION
        # right side motor
        gpio.output(self.front_right_dir_pin, False)
        #gpio.output(self.back_right_dir_pin, True)
        # left side motor
        gpio.output(self.front_left_dir_pin, False)
        #gpio.output(self.back_left_dir_pin, True)

        # STEP
        for i in range(abs(step_count)):
            gpio.output(self.front_left_step_pin, True)
            gpio.output(self.front_right_step_pin, True)
            #gpio.output(self.back_left_step_pin, True)
            #gpio.output(self.back_right_step_pin, True)
            time.sleep(self.step_time)
            gpio.output(self.front_left_step_pin, False)
            gpio.output(self.front_right_step_pin, False)
            #gpio.output(self.back_left_step_pin, False)
            #gpio.output(self.back_right_step_pin, False)
            time.sleep(self.step_time)
    # def relative_angle(self, angle):
```

```
    # def absolute_angle(self, angle):

if __name__ == '__main__':
    print("STEPPER MODULE SELF TESTING")
    #gpio.cleanup()
    ms1_pin=14
    ms2_pin=14
    ms3_pin=14
    enable_pin=14
    front_left_step_pin=26
    front_left_dir_pin=19

    back_left_step_pin=6
    back_left_dir_pin=13

    front_right_step_pin=2
    front_right_dir_pin=3

    back_right_step_pin=17
    back_right_dir_pin=27

    FRONT_LEFT=[front_left_step_pin, front_left_dir_pin]
    FRONT_RIGHT=[front_right_step_pin, front_right_dir_pin]
    BACK_LEFT=[back_left_step_pin, back_left_dir_pin]
    BAKC_RIGHT=[back_right_step_pin, back_right_dir_pin]


stepper=stepper(FRONT_LEFT,FRONT_RIGHT,BACK_LEFT,BAKC_RIGHT,ms1_pin,ms2_pi
n,ms3_pin,enable_pin,0)
    # right side backward
    # stepper.steps(100)
    # left side front
    stepper.move_U()
    time.sleep(0.75)
    stepper.move_F()
```

```
#!/usr/bin/env python
# encoding: utf-8
```

```
from numpy import *
from STEPPER import stepper

def Move(last_direction_vector,current_position,next_position):
    move_direction=""
    # # if last_direction_vector is no exist yet (first step), default UP
    # try:
    #     last_direction_vector
    # except NameError:
    #     last_direction_vector = mat([[0,1]])
    # else:
    #     pass

    # print("last direction,", last_direction_vector)

    if current_position[0,0]<next_position[0,0] and
current_position[0,1]==next_position[0,1]:

        direction_str="RIGHT"
        # direction vector
        direction_vector=mat([[1,0]])
        trasnmit_matrix=mat([[0,0,0,1],
                             [0,0,1,0],
                             [1,0,0,0],
                             [0,1,0,0]])
    elif current_position[0,0]>next_position[0,0] and
current_position[0,1]==next_position[0,1]:
        direction_str="LEFT"
        # direction vector
        direction_vector=mat([[-1,0]])
        trasnmit_matrix=mat([[0,0,1,0],
                             [0,0,0,1],
                             [0,1,0,0],
                             [1,0,0,0]])
    elif current_position[0,0]==next_position[0,0] and
current_position[0,1]<next_position[0,1]:
        direction_str="UP"
        # direction vector
        direction_vector=mat([[0,1]])
        trasnmit_matrix=mat([[1,0,0,0],
```

```
                            [0,1,0,0],
                            [0,0,1,0],
                            [0,0,0,1]])
    elif current_position[0,0]==next_position[0,0] and
current_position[0,1]>next_position[0,1]:
        direction_str="DOWN"
        # direction_vector vector
        direction_vector=mat([[0,-1]])
        trasnmit_matrix=mat([[0,1,0,0],
                            [1,0,0,0],
                            [0,0,0,1],
                            [0,0,1,0]])


    # print("now direction", direction_vector)




    # calculate relative angle by dot product
    cos_theta=(last_direction_vector-
mat([[0.02,0.01]]))*(direction_vector+mat([[0.01,0.02]])).transpose()

    if cos_theta == -1.0304 or cos_theta == -0.9703999999999999:
        move_direction="U-turn"

        transmit_vector=mat([[0,-1]])
        Stepper.move_U()
        Stepper.move_F()
    elif cos_theta == 1.0096 or cos_theta == 0.9896:
        move_direction="Forward"
        transmit_vector=mat([[0,1]])
        Stepper.move_F()
    elif cos_theta == -0.0204 or cos_theta == -0.0003999999999999976 or
cos_theta == 0.0196:
        move_direction="Rightward"
        transmit_vector=mat([[1,0]])
        Stepper.move_R()
        Stepper.move_F()
```

```
    elif cos_theta == -0.00040000000000000105 or cos_theta == 0.0396 or
cos_theta == -0.0404:
        move_direction="Leftward"
        transmit_vector=mat([[-1,0]])
        Stepper.move_L()
        Stepper.move_F()


    last_direction_vector=direction_vector
    print("MOVE "+move_direction)
    return current_position, trasnmit_matrix, last_direction_vector

if __name__ == "__main__":
    current_grid=mat([[12,12]])
    next_grid=mat([[12,11]])
    last_dir=mat([[1,0]])
    Move(last_dir,current_grid,next_grid)
```

```
#!/usr/bin/env python
# encoding: utf-8
import RPi.GPIO as GPIO                     #Import GPIO library
import time                                 #Import time library
GPIO.setmode(GPIO.BCM)                      #Set GPIO pin numbering

class UltraSonic_dev:
    FLOOR_BOOL = False                                  #Associate pin 24
to ECHO
    FRONT_BOOL = False                                  #Associate pin 24
to ECHO
    RIGHT1_BOOL = False
    RIGHT2_BOOL = False
    BACK_BOOL = False
    LEFT2_BOOL = False
    LEFT1_BOOL = False


    # Initialize
    def __init__(self, TRIG, ultrasonic_echo_set):
        # setup
        self.TRIG=TRIG
```

```
        self.ultrasonic_echo_set=ultrasonic_echo_set
        self.distance=[99999,99999]
        GPIO.setup(self.TRIG,GPIO.OUT)                  #Set pin as GPIO
out
        for echo_pin in self.ultrasonic_echo_set:
            GPIO.setup(echo_pin,GPIO.IN)                    #Set pin as
GPIO in

    def detecting_process(self, side_set):
        pulse_start=[0,0]
        pulse_end=[0,0]
        pulse_duration=[0,0]

        i=0
        for echo_pin in side_set:
            # Trig the ultrasonic soundwave
            GPIO.output(self.TRIG, False)                 #Set TRIG as LOW
            time.sleep(0.01)                             #Delay of 2
seconds
            GPIO.output(self.TRIG, True)                  #Set TRIG as
HIGH
            time.sleep(0.00001)                         #Delay of 0.00001
seconds
            GPIO.output(self.TRIG, False)                 #Set TRIG as LOW

            while GPIO.input(echo_pin)==0:               #Check whether
the ECHO is LOW
                pulse_start[i] = time.time()              #Saves the last
known time of LOW pulse
            while GPIO.input(echo_pin)==1:               #Check whether
the ECHO is HIGH
                pulse_end[i] = time.time()                #Saves the last
known time of HIGH pulse

            pulse_duration[i] = pulse_end[i] - pulse_start[i] #Get pulse
duration to a variable

            self.distance[i] = pulse_duration[i] * 17150        #Multiply
pulse duration by 17150 to get distance
```

```
            self.distance[i] = round(self.distance[i], 2)
#Round to two decimal points
            i=i+1

#         print('First:{:g},
Second:{:g}'.format(self.distance[0],self.distance[1]))
        time.sleep(0.001)

    def detect(self, direction=None):
        if direction=='left':
            print("Left Ultrasonic is detecting")
            left_ultrasonic_set=self.ultrasonic_echo_set[4:5+1]
#             while True:
#                 self.detecting_process(left_ultrasonic_set)
            k=0
            avg_distance=[]
            while k<10:
                self.detecting_process(left_ultrasonic_set)
                avg_distance.append(self.distance[0])
                k=k+1
            return sum(avg_distance)/10.0
        if direction=='right':
            print("Right Ultrasonic is detecting")
            right_ultrasonic_set=self.ultrasonic_echo_set[2:3+1]
#             while True:
#                 self.detecting_process(right_ultrasonic_set)
            k=0
            avg_distance=[]
            while k<10:
                self.detecting_process(right_ultrasonic_set)
                avg_distance.append(self.distance[0])
                k=k+1
            return sum(avg_distance)/10.0
        if direction=='front':
            print("Front Ultrasonic is detecting")
            front_ultrasonic_set=self.ultrasonic_echo_set[1:1+1]
#             while True:
#                 self.detecting_process(front_ultrasonic_set)
            k=0
            avg_distance=[]
```

```
            while k<10:
                self.detecting_process(front_ultrasonic_set)
                avg_distance.append(self.distance[0])
                k=k+1
            return sum(avg_distance)/10.0
        if direction=='back':
            print("Back Ultrasonic is detecting")
            back_ultrasonic_set=self.ultrasonic_echo_set[6:6+1]
#             while True:
#                 self.detecting_process(back_ultrasonic_set)
            k=0
            avg_distance=[]
            while k<10:
                self.detecting_process(back_ultrasonic_set)
                avg_distance.append(self.distance[0])
                k=k+1
            return sum(avg_distance)/10.0

if __name__=='__main__':
    print("Unit Test: Ultrasonic")
#     GPIO.cleanup()
    TRIG = 21
    # 20 16 12 7 8 23 24
    FLOOR = 20
    FRONT = 23
    RIGHT1 = 12
    RIGHT2 = 8
    LEFT1 = 7
    LEFT2 = 16
    BACK = 24
    ultrasonic_echo_set=[FLOOR, FRONT, RIGHT1, RIGHT2, LEFT1, LEFT2, BACK]
    ultrasonic=UltraSonic_dev(TRIG,ultrasonic_echo_set)
    flag=ultrasonic.detect(direction='back')
    print(flag)
```

```
#!/usr/bin/env python
# encoding: utf-8
from numpy import *
```

```
from H import H
from isSamePosition import isSamePosition
from MotionModel import MotionModel
from FindList import FindList
from GetBoundary import GetBoundary
from GetObstacle import GetObstacle
from isObstacle import isObstacle
from GetPath import GetPath

def Astar(obstacle,start,goal):
    G=0 # NEXT MOVE cost
    path=mat([[0,0]]) # Create an empty path, zero will be removed after
OPTIMAL PATH found
    open_list=mat([[start[0,0]],                # current position x
                   [start[0,1]],                # current position y
                   [G+H(start,goal)],           # total cost F=G+H, 当前点到
终点的距离
                   [start[0,0]],                # last position x, parent
set
                   [start[0,1]]]).transpose()   # last position y, parent
set
    # initialize CLOSE LIST, this all-zero row will be removed after
OPTIMAL PATH found
    close_list=mat([[0,0,0,0,0]])
    # open_list=mat([[0,0]])
    # open_list=delete(open_list,0,axis=0)
    next_move=MotionModel() # set NEXT position motion model
    findFlag=False # flag to determine whether the path can be found
    # print(open_list)

    while findFlag==False:
        # first column of OPEN LIST is empty
        if len(open_list)==0:
            print("No path to GOAL.")
            return

        # sorting open list based on total cost
        open_list=open_list[lexsort((open_list.view(ndarray)[:,2],))]
        # print(open_list)
```

```
        # compare the current position to GOAL position
        if isSamePosition(open_list[0,0:2],goal):
            print("Optimal path found.")
            # put first row of OPEN LIST into CLOSE LIST
            close_list=vstack((open_list[0,:],close_list))
            # remove the least cost MOVE from OPEN LIST
            open_list=delete(open_list,0,axis=0)
            findFlag=True
            break

        # calculate the cost in NEXT MOTION MODEL
        for index in range(0,len(next_move[:,0])):
            m=mat([[open_list[0,0]+next_move[index,0]], # pick NEXT MOVE
position x
                [open_list[0,1]+next_move[index,1]], # pick NEXT MOVE
position y
                [0]]).transpose()
            G=next_move[index,2]+H(m[0:2],goal) # NEXT MOVE cost G
            m[0,2]=G
            # print("m =",m)

            # skip if current position is OBSTACLE
            if isObstacle(m,obstacle):
                # print("[{}, {}] is OBSTACLE".format(m[0,0],m[0,1]))
                continue

            # check that whether the next movement choice is in OPEN LIST
or CLOSE LIST
            list_flag=FindList(m,open_list,close_list)
            # print("list flag =",list_flag)
            # if it is in OPEN LIST or CLOSE LIST, skip
            if list_flag==1: # in OPEN LIST
                # print("[{}, {}] is in OPEN LIST".format(m[0,0],m[0,1]))
                continue
            elif list_flag==2: # in CLOSE LIST
                # print("[{}, {}] is in CLOSE LIST".format(m[0,0],m[0,1]))
                continue
            else:
                # append this MOVE and current position into OPEN LIST
```

```
                # print("[{}, {}] has appended into OPEN
LIST".format(m[0,0],m[0,1]))
                temp=hstack((m,[[open_list[0,0]]],[[open_list[0,1]]]))
                open_list=vstack((open_list,temp))
                # print(open_list)

        # if the NEXT MOVE is neither in OPEN LIST nor CLOSE LIST
        if findFlag==False:
            # append the least cost MOVE into CLOSE LIST, as moved
            close_list=vstack((open_list[0,:],close_list))
            # print("[{}, {}] has added into
PATH".format(close_list[0,0],close_list[0,1]))
            # remove the least cost MOVE from OPEN LIST
            open_list=delete(open_list,0,axis=0)

    # remove the last all-zero row in CLOSE LIST
    close_list=delete(close_list,-1,axis=0)
    # print(close_list)
    # generate OPTIMAL PATH
    path=GetPath(close_list,start)
    return path

if __name__ == '__main__':
    start_point=mat([[1,1]])
    end_point=mat([[4,4]])
    obstacle=GetBoundary(5)
    print("Unit Test - Astar: ", Astar(obstacle,start_point,end_point))
```

```
#!/usr/bin/env python
# encoding: utf-8
import time, RPi.GPIO, threading
from LED import led_flash
from STEPPER import stepper
# pin statement
#green_led_pin=17
#blue_led_pin=27

ms1_pin=14
ms2_pin=14
```

```
ms3_pin=14
enable_pin=14
front_left_step_pin=26
front_left_dir_pin=19
back_left_dir_pin=13
back_left_step_pin=6
front_right_dir_pin=3
front_right_step_pin=2
back_right_dir_pin=27
back_right_step_pin=17
# initialize
#led=led_flash(green_led_pin,blue_led_pin)
front_left_stepper=stepper(front_left_step_pin,front_left_dir_pin,ms1_pin,
ms2_pin,ms3_pin,enable_pin,0)
back_left_stepper=stepper(back_left_step_pin,back_left_dir_pin,ms1_pin,ms2
_pin,ms3_pin,enable_pin,0)
front_right_stepper=stepper(front_right_step_pin,front_right_dir_pin,ms1_p
in,ms2_pin,ms3_pin,enable_pin,0)
back_right_stepper=stepper(back_right_step_pin,back_right_dir_pin,ms1_pin,
ms2_pin,ms3_pin,enable_pin,0)

threads=[]
task1=threading.Thread(target=front_left_stepper.steps, args=(500,))
threads.append(task1)
task2=threading.Thread(target=back_left_stepper.steps, args=(500,))
threads.append(task2)
task3=threading.Thread(target=front_right_stepper.steps, args=(-500,))
threads.append(task3)
task4=threading.Thread(target=back_right_stepper.steps, args=(-500,))
threads.append(task4)

if __name__ == '__main__':
    print("RUNNING...")
    try:
        for t in threads:
            t.setDaemon(True)
            t.start()
        t.join()
    except KeyboardInterrupt:
        RPi.GPIO.cleanup()
```

Section III: Mapping Algorithms, Theory

Dynamic Programming Algorithm:

$$x_{t+1} = f_t(x_t, u_t) = x_t + u_t$$

$$J(\gamma_{t:T-1}) = g_T(x_T) + \sum_{t=0}^{T-1} g_t(x_t, \gamma_t)$$

$$\gamma_t^*(x_t) = \underset{u_t}{\operatorname{argmin}}\, g_t(x_t, u_t) + V_{t+1}(f_t(x_t, u_t))$$

$$V_t(x_t) = \min_{u_t} g_t(x_t, u_t) + V_{t+1}(f_t(x_t, u_t))$$

where
xt-: state, trajectory
ut: action
γt*: optimal policy
J: total operation cost
Vt: optimal cost-to-go
gt: operation cost

Dijkstra's Algorithm:

$$V_t = 0$$

$$\gamma_t^*(x_t) = \underset{u_t}{\operatorname{argmin}}\, g_t(x_t, u_t), \qquad t = 0,1, \dots, T-1$$

$$F_n = G_n$$

where n is the index of next state,
G is the operation cost function as the total actual cost from the start point to the next state,
F is the total evaluation cost, also known as priority.

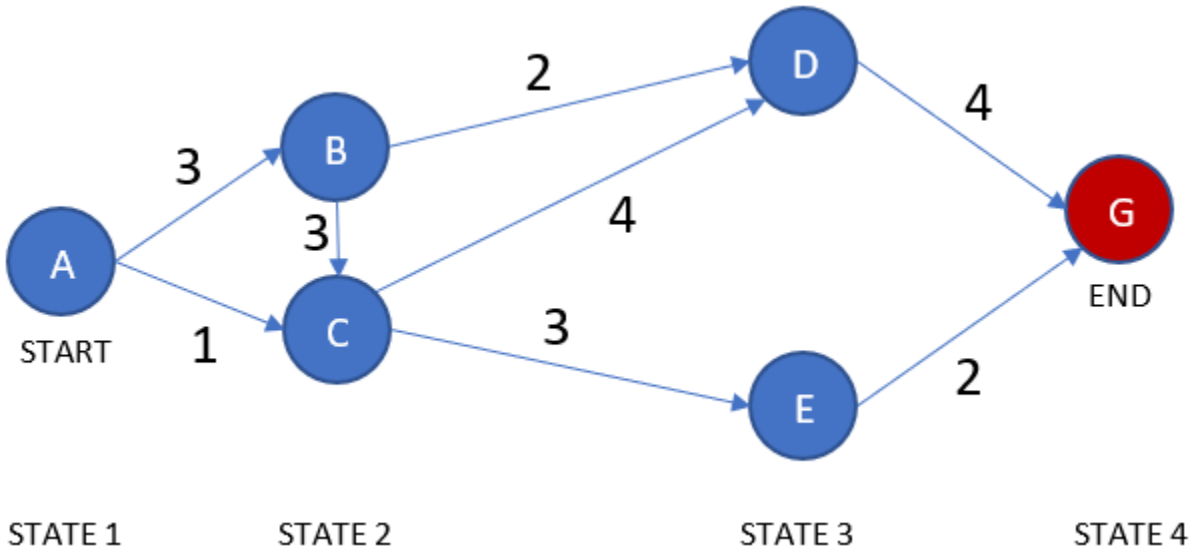


Figure B 7: Example of Dijkstra's Algorithm

Table B 1: The process of Dijkstra's Algorithm

t	x_t	x_{t+1}	g_T	$\sum_t^{T-1} g_t(x_t, \gamma_t)$	J	u_t	γ_t^*
1	A	B	3	0	3	A-B	
1	A	C	1	0	1	A-C	A-C
2	B	C	3	3	6	B-C	
2	B	D	2	3	4	B-D	B-D
2	C	D	4	1	5	C-D	
2	C	E	3	1	4	C-E	C-E
3	D	G	4	5	9	D-G	
3	E	G	2	4	6	E-G	E-G

By backpropagating to reach endpoint G, the optimal policy is E to G, and C to E and A to C.

Best-First Search Algorithm:

$$g_t = 0$$

$$\gamma_t^*(x_t) = \underset{u_t}{\operatorname{argmin}} V_{t+1}(f_t(x_t, u_t)), \qquad t = 0,1, \dots, T-1$$

$$F_n = H_n$$

where H is the heuristic estimated cost-to-go, estimated distance from next state n to the goal.

Best-First Search Algorithm greatly speeded up the search but sacrificed the optimality. Many times, the strategy is not optimal apparently.

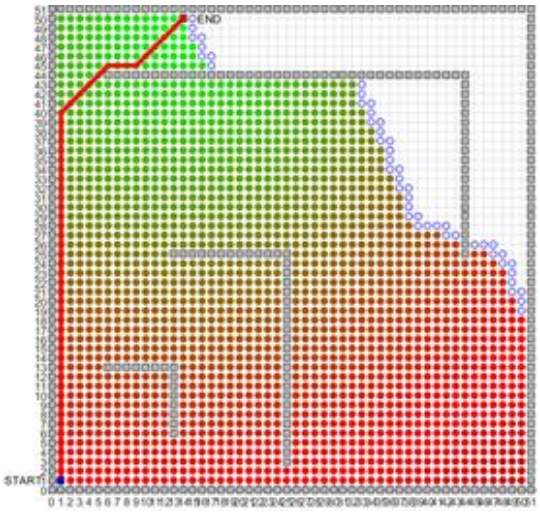


Figure B 8: Dijkstra's Algorithm, cost 56.14

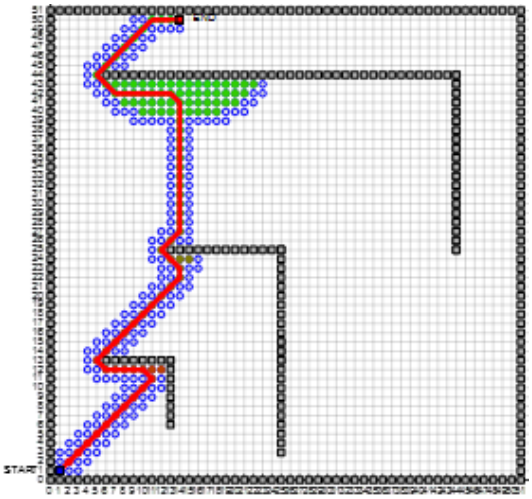


Figure B 9: Best-First Search Algorithm, cost 76.08

A* Algorithm, Dynamic Programming

In 1968, Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute (now SRI International) first published A* (A star) algorithm, which is a Best-First or informed search idea and using heuristics to guide its search, and it can be seen as a combination with Dijkstra's algorithm and Best-First Search Algorithm.

A* Algorithm:

$$F_n = G_n + H_n$$

For each step, A* algorithm tries to find the minimum F in all the possible next searching choices, and by minimizing the operation cost

$$J = g_T(x_T) + \sum_{t=0}^{T-1} g_t(x_t, \gamma_t)$$

to find out the optimal one until the goal is reached. After that, using backpropagation, from goal to start, it backtracks these choices to generate the policy γ*, which is the optimal path.

Appendix C

Meeting Minutes:

September 05, 2019

Meeting with the Professor Fought

Team 4: Go ~~Go~~ Power Rangers

Date: September 5, 2019

Attendance: Ryan Hackney, Devin Hensley, Emily Kong, Yoon Jae Lee, Matt Stoner, Yuan You, Professor Fought

- Matt starts by explaining the idea of the fire fighting UGV
- Fought points out the specific functions robot will have to accomplish for that idea
    - Small autonomous vehicle can't hold enough extinguisher
- Decided to account for navigation and mapping only
- Optimize the scale to something we can manage while also holding all components needed
- A problem statement needs to be determined that will work with the project
- Stretch goals will be added to the project if extra time is there
- Project will have 3 different parts: Robot Build, Navigation, and Mapping

September 12, 2019

Interim Meeting with the Professor Fought

Team 4: Go Go Power Rangers

Date: September 12, 2019

Attendance: Ryan Hackney, Devin Hensley, Emily Kong, Yoon Jae Lee, Matt Stoner, Yuan You, Professor Fought

- Emily explains UGV to map out room and show back to user
- Professor Fought says keep story consistent throughout report
- Don’t include prototype ideas until the design report
- Plan for mapping hallways in Caldwell, Baker Systems, and Dreese
- System block diagram is a representation of the product
- Work Breakdown Structure (WBS) is a list of tasks along with who is assigned those tasks
- Milestones are events, not tasks
    - A task is research which software to use, milestone is a software is selected

September 17, 2019

Meeting with the Professor Fought

Team 4: Go Go Power Rangers

Date: September 17, 2019

Attendance: Ryan Hackney, Emily Kong, Yoon Jae Lee, Yuan You, Professor Fought

- Discussed schedule to create design concepts for next week
- Took a look at motors and sensors in storage for potential use
- Plan to meet and pull out specific storage parts on Thursday
- Confirmed each section of the team (Navigation, Mapping, Robots)

October 03, 2019

Meeting with the Professor Fought

Team 4: Go Go Power Rangers

Presentation Meeting:
Date: October 3, 2019

Attendance: Ryan Hackney, Devin Hensley, Emily Kong, Yoon Jae Lee, Matt Stoner, Yuan You, Professor Fought

- Team 4 presents problem statement, design concepts and technical evaluation
- Professor Fought says
- Team 4 presents moving mechanism and algorithm
- Professor Fought says algorithm should be clear and precise.
- Team 4 presents testing plans, task, milestones and schedules

October 23, 2019

Meeting with the Professor Fought

Team 4: Go Go Power Rangers

Parts Receiving and New purchase order Meeting
Date: October 22, 2019

Attendance: Emily Kong, Yoon Jae Lee, Matt Stoner, Professor Fought

- Received ordered parts and aware of shipping delay for some parts.
- Team 4 discussed power supply and battery charging
- Finalize two batteries and one battery for Pi. Checked eligibility and charging conditions.
- Team 4 announced a new purchase order to Professor Fought
- Professor Fought informs how to pickup order in person
- Plan to meet Thursday for assembly more with 3D printed wheels.

November 12, 2019

Meeting with the Professor Fought

Team 4: Go Go Power Rangers

Date: November 12, 2019

Attendance: Ryan Hackney, Devin Hensley, Emily Kong, Yoon Jae Lee, Matt Stoner, Yuan You, Professor Fought

- Demonstrated the assembled prototype to Professor Fought
- Discussed optimized size and locating.
- Professor Fought gave advice about sensor operation and the idea of board connection.
- The team discussed circuit wiring
- Plan for operating in hallways of Caldwell and Dreese with updated code.

November 21, 2019

Meeting with the Professor Fought

Team 4: Go Go Power Rangers

Date: November 21, 2019

Attendance: Ryan Hackney, Devin Hensley, Emily Kong, Yoon Jae Lee, Matt Stoner, Yuan You, Professor Fought

- Demonstrated presentation about Critical Design Review
- Professor Fought gave advice about finessing our projects
- Plan to update the document in more detail about electrical connections and software.

November 27, 2019

Meeting with the Professor Fought

Date: November 26, 2019

Attendance: Yoon Jae Lee, Yuan You, Professor Fought

- Discussed about rotating issue.
- Professor Fought gave advice about friction and wheels.
- Plan to update some parts for effective and constant moving mechanism.