Mobile Sensing Platform

Jeremy Johnson, Jiwon Nam, Shibo Tang, Zhao Wang
j.erik.johnson1994@gmail.com, skajw90@gmail.com, shibot.tom@gmail.com, wangzhaoppz@gmail.com
ECE 4710, University Of Utah
Department of Electrical and Computer Engineering
URL: https://www.group2.info/

Abstract— **In the world today, information is everything. We designed and constructed a mobile sensing platform that is easy to transport and operate. Our platform took the form of a remote-control car. The remote for our car was a custom iOS application. Communication between the car and app was done using a local Wi-Fi network. A user was able to control the cars movement, driving forward and backward. The car was equipped with a camera and mapping sensors and streamed information to the app. The work for our project was divided into several important sections: the iOS application, the communication protocols via Wi-Fi, the sensor/motor/power configurations for the car, the car frame/tire components, and the mapping algorithms. By combining all these together we had a complete, functional mobile sensing platform. We demonstrated full functionality of our iOS app to control the car driving anywhere within the same Wi-Fi network. You could drive out of sight using the video feed from the onboard camera, and we demonstrated successful collection of mapping information as the vehicle was driving around. The information was then compiled in real-time into a graphical map displayed on the application. Our project provides an avenue for collecting previously inaccessible information in a safe and effective way.**

I. Introduction

For our project we wanted to come up with a project that was not only challenging for us, but useful in the real world and fun for everyone on demo day. We wanted to incorporate an iOS application to control something physical. We decided on a car due to its applicability in the real world. Many jobs in the world can be dangerous to humans and the transition has already begun to move towards robots. Our goal was to provide a mobile sensing platform from which as much information as you could want about an environment could be obtained in a direct way without ever having to be there yourself.

Our information gathering was largely reliant upon a live video feed from the car. This allowed the user to guide the car to the locations they desired to get the data they needed. Besides gathering information via the video feed through which the user can see what is around, we also incorporated distance sensors on the sides of the car.

We then gave the user the option to map where the car went and what it saw on either side. This data was made immediately available to the user creating the opportunity for real-time information gathering without ever having to go to the location themselves.

Our project combined the technologies of Wi-Fi data transmission, embedded system design, iOS application development, live video transmission and room-mapping algorithms. Most of the current products available are for recreational use. Our design was intended to be used in industry. We separated our project into several major tasks:

- Driving controls
- Wi-Fi data transmission
- Live video feed
- Room mapping
- iOS application UI design
- Combine everything above together and test

Below is a figure representing what our overall design looked like:

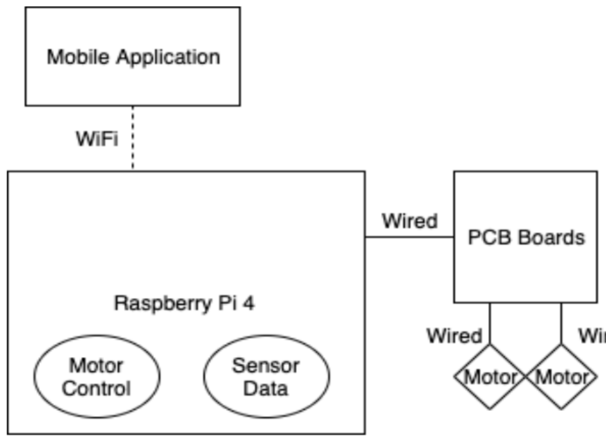


Fig. 1. Project Diagram

II. Hardware

A. Raspberry Pi 4 Model B

In our project proposal, we were planning on using STM32F072 Discovery boards to run the PCB control program and interface with the other sensors. Because we were planning to use Wi-Fi for communication between the mobile application and the STM32F072, we needed to add an additional Wi-Fi module, we chose an ESP8266. Moving forward in our project design we decided to find a solution that could more easily support the use of a camera module since that is such a vital part of our project. We decided to change to using a Raspberry Pi 4 as our new microcontroller. The raspberry pi had enough GPIO ports to interface all the sensors and the two motors (pcb). In addition, it has a built-in port to connect a camera module and has a built-in Wi-Fi module. The memory on the pi was more than enough for us to store our sensor data. This also turned out to be a good choice because of the later need to run multiple programs at the same time on the pi.

Microcomputer: Raspberry Pi 4 Model B

Manufacturer: Raspberry Pi Foundation

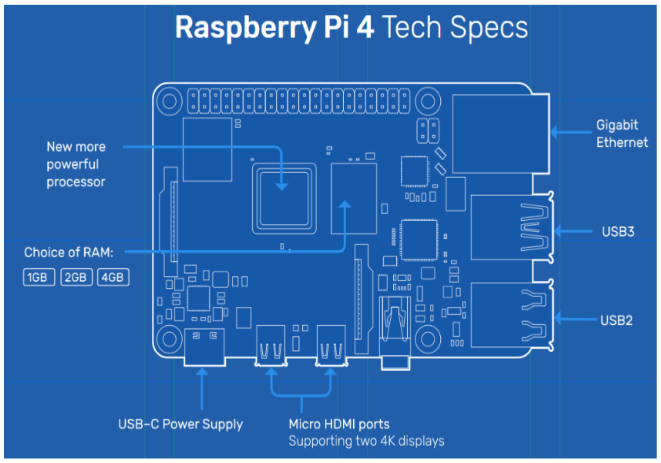


Fig. 2. Raspberry Pi 4 Microcomputer Tech Specs [1]

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit @ 1.5 GHz
- 4GB LPDDR4-3200 SDRAM
- 2.4 GHz and 5.0 GHz IEEE 802.11ac wireless
- Bluetooth 5.0
- 2 * USB 3.0 ports
- 2 * USB 2.0 ports
- 2 * micro-HDMI ports
- Raspberry Pi standard 40 pin GPIO header
- 2-lane MIPI CSI camera port
- 3.3V, 5V DC via GPIO header
- Micro-SD card slot for loading operating system and data storage

The Raspberry Pi became the central hub for our project. We used the pi to run the PCB motor drivers, camera and sensors, and create the server which is used to transfer the data between the car and mobile application. More specifics on software will be discussed later in this paper but we had 3 main programs running on the pi that gave our project life. We had the master controller which gathered data from the sensors and motors and controlled the motor speed. We ran a MQTT server which supported the communication between the application and the master controller. And we set up a webpage to stream the video to. All of these needed to be running for the project to be a success.

B. Motors

Motor: Pololu 2824 - 50:1 Metal Gearmotor 37Dx70L mm with 64 CPR Encoder Manufacturer: Pololu

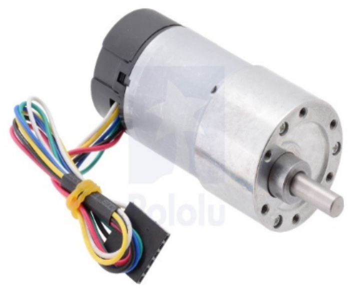


Fig. 3. Pololu 2824 Motor [2]

This gearmotor is a powerful 12V brushed DC motor with a 50:1 metal gearbox and an integrated quadrature encoder that provides a resolution of 64 counts per revolution of the motor shaft [2]. This will give us the proper torque to maneuver the car. One of the most important features of this motor is the integrated encoder.

The encoder played several vital roles in our project. First, we used the encoders in combination with a PID controller to get both motors to turn at the same speed. This is important because motors of the same variety have minute differences in their composition. This causes them to turn at slightly different speeds even when given the same amount of power.

By using the encoder values from the motors, we could tell exactly how fast the motor was turning and adjust the individual power to each motor to get them to turn at the same speed, thus allowing us to drive straight. The second importance behind the encoders was for our mapping functionality. We used the encoders to tell not only how far the vehicle had moved since the last sampling, but also used them to tell how far the vehicle had turned when one motor turned further than the other.

Fig. 4. and Fig. 5. shows how the motors quadrature encoder works, and how it can be used to get the rotation amount for speed control and mapping.

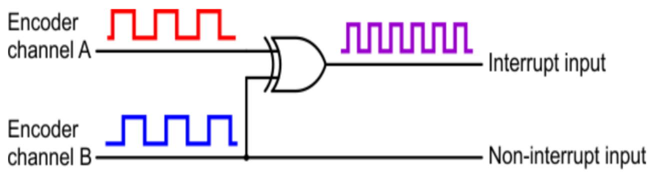


Fig. 4. Channel A and B in Encoder with XOR gate [3]

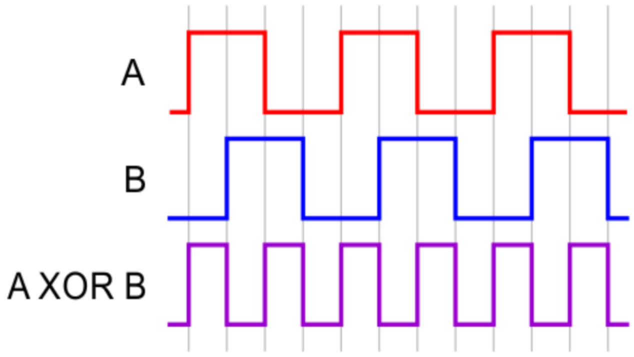


Fig. 5. Square Waveform of Channel A,B and A XOR B [3]

To monitor the speed of rotation, simply use the output or measure the frequency. The reason for having two outputs is that you can also determine the direction of shaft rotation by looking at the pattern of binary numbers generated by the two outputs. We used this to be able to measure the direction of the car as well as the distance.

C. Motor Driver

PCB motor driver: Custom Design

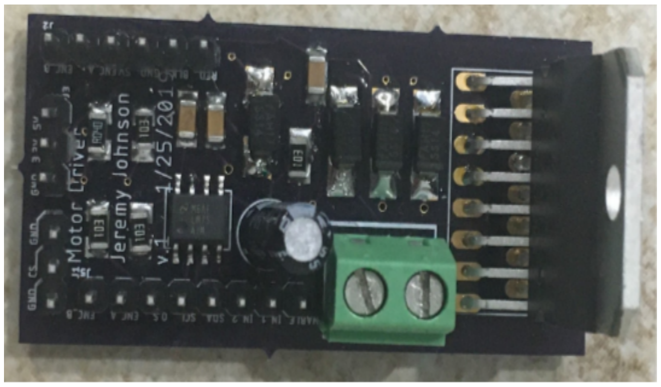


Fig. 6. Motor Driver Board

We designed a custom pcb motor driver that was equipped to drive the Pololu 50:1 metal gearmotor. It used a L289N H-bridge and had all the connections to be able to drive the motor, switch its direction and read back the encoder values. It took in 12V power to drive the motor and a 5V power supply to give and receive the proper control signals and encoder values from the motor.

D. Camera

Camera Module: Kuman Camera Module 5MP 1080p OV5647 Sensor HD Video Webcam

Manufacturer: Kuman

Fig. 7. Kuman for Raspberry Pi Camera Module [4]

- RPi Camera, Supports Raspberry Pi Model B/B+ A+ RPi 3 2 1 with the FFC Cable, it also support raspberry pi zero/ zero W with the FPC Cable
- 5-megapixel OV5647 sensor, Supports up to 2 infrared LED and/or fill flash

We selected this camera module because of its ease of use with the Raspberry Pi. The Pi we chose has a port that can directly interface with the camera, a ribbon runs from the camera into the port. The module transfers live video data to the Raspberry Pi. We mounted the camera to the front of the vehicle so that the user could use the feed to see where they were driving. The camera has a light sensor so that it can auto change between normal and infrared mode.

E. Distance Sensors

Ultrasonic distance sensor module: HC-SR04

Manufacturer: ElecRight

Fig. 8. HC-SR04 Ultrasonic Sensor Distance Module [5]

- Power Supply: 5V DC
- Effectual Angle: ¡15
- Detection Distance: 2 cm   500 cm
- Resolution: 0.3 cm

There are four HC-SR04 sensors used in this project: two for the left side and two for the right side of the car. This sensor sends out eight 40 kHz signals and detects whether it gets any pulse signal back. If it receives a signal in return, a high-level signal will be outputted by IO, and the duration of the signal is the time between the sent ultrasonic wave to when it received the return signal. The time difference is then converted to determine the distance an object is away from the sensor. To connect the sensor to Raspberry Pi board, it needs:

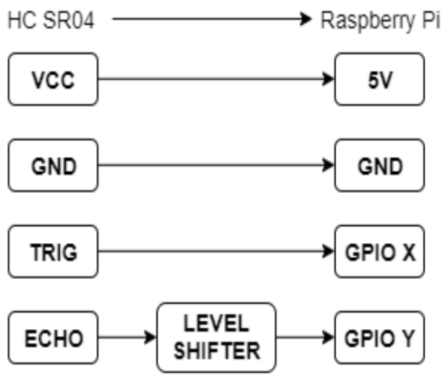


Fig. 9. Wire Connection Between HC-SR04 and Raspberry Pi

The level shifter is used to shift the 5V signal from echo to 3.3V because the GPIO pins on the Pi only tolerate a maximum of 3.3V. The connection between echo, level shifter and GPIO is shown below:

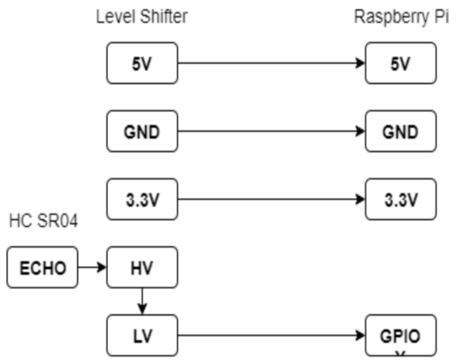


Fig. 10. Wire Connection Between HC-SR04, Level Shifter and Rasberry Pi

We installed two sensors on both sides of the car. The sensors were placed in a pattern with one above the other. This is because HC-SR04 is not the most stable type of ultrasonic sensor. Sometimes, it may give errors on feedback that would cause the mapping to fail. To solve this issue, we designed to use two sensors in combination with an "error fix" algorithm which solved most of the errors from the sensors. This algorithm is included in our data collecting program which can collect and compare the data from both sensors and fix the error.

F. Power Supply

Power Supplies:
The car utilizes three power supplies. One to drive the 12V for each motor, and one 5V supply for the Raspberry Pi:

- 5V-2.1A outputs 20000mAh for Raspberry Pi * 1
- 12V output for motors

Initially we were running everything off the same power bank. However, we noticed some inconsistencies with the motors when everything was connected to the same power source. The motors would consistently perform at different speeds and the increased power draw from one would affect the other. Due to these issues we opted to provide each motor with its own 12V power supply and have a third power bank provide the 5V for the Raspberry Pi.

III. Software

A. MQTT Server

For communication between the mobile application and the raspberry pi, we decided to use an MQTT (Message Queuing Telemetry Transport) protocol. It is a lightweight publish and subscribe system where different clients connect to a server and can each publish and receive messages. It is designed for constrained devices with low bandwidth. This was a perfect solution for our project. Using the Raspberry Pi, we spun up a MQTT server. The vehicle controller then connected to the server and subscribed to a variety of topics, these included "leftMotorForward", "leftMotorReverse" as well as topics regarding mapping. The iOS application also connected to the server as a client and subscribed to different topics. When we needed to communicate between the app and the car, they would send a message to the server under one of those topics. The server then looks at which of its clients care about said topic and sends it off. Below is a flowchart of the setup with some example topics:

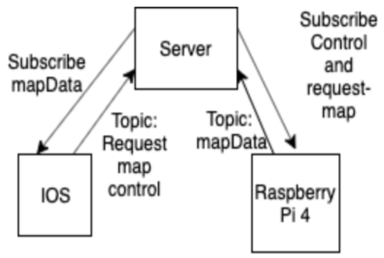


Fig. 11. MQTT data transmission

B. Car Controller

Our controller program on the Pi oversaw all of the hardware on the vehicle. The program was divided into 3 parts: server communication, motor control and mapping data collection.

As previously discussed, our communication server was built on an MQTT protocol. When the program was fired up it immediately attempted to connect to the MQTT server. Once connected it subscribed to a variety of topics in order to receive motor speed and direction commands, start mapping command, stop mapping command and a map received command. We designed handlers for each of these topics to respond accordingly.

Early in the project we ran into some issues with our motor control. We were initially just using a PWM to send power to the motors so we could go different speeds. The issue was that the motors we had turned at different speeds from each other even when given the same amount of power. To solve this issue, we designed a PID controller. Instead of telling the motor controller how much power to send the motors we told the controller what speed we wanted them to go. If the speed was too slow, the controller would add a little power. If it was too fast, it would cut some power. We had a PID controller for each motor, this enabled them to adjust themselves independent of the other and end up at the correct speeds allowing us to drive straight.

The final part to our program was the mapping data. When the user requested that a map be made, we began to sample all the sensors and encoders at a rate of 4 times a second. We elected to save the data in a text file that was comma and end line delimited. All the data from one snapshot was comma delimited and the snapshots were separated by an end line character. When the user requested to stop mapping, the information was immediately sent to the iOS application as one large string. The application could then easily parse through the data to begin creating the map.

C. Video Streaming

In our initial attempts at video streaming we tried to find ways to package up all the data and send it directly to the iOS application. Once there we would unpackage the data and reformat it into an image displaying to the user the live video stream. We ran into many issues with this and after coming up empty handed, we determined an easier way. Instead of packaging the data and sending it to the user directly, we spun up a simple web page and streamed the video there. Then, in the application we simply displayed that webpage as the background of the application, allowing the user to see everything that was going on. This wasn't an issue since our project was based on Wi-Fi communication, there was no problem with streaming the video to a web-page and displaying it from there, since there would always need to be Wi-Fi to run the project in the first place.

D. IOS Application

We used a MVC technique in the iOS application. There are four main controllers, an initial view controller, a main view controller, a map table view and a map view controller. All the controllers connect with their own view and model, and they used protocol to available passing data source and delegates without calling class. All view sizes are relative to iOS device screen size so that user can run this application on any iOS device with no difference.

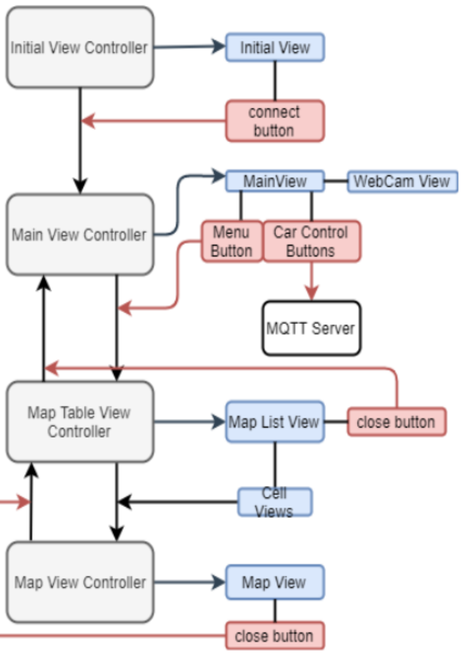


Fig. 12. IOS application overview

The initial view controller creates a view with a connect button so that the user can access the server. After the button is clicked, the controller pushes the main view controller to the front so that user can see main view.

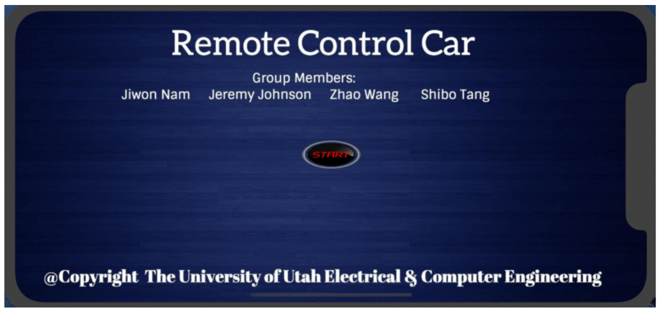


Fig. 13. Initial view controller of IOS Application

The main view controller has four car control buttons (left, right, acceleration and brake), speed labels, a drawing map button, and menu button. The car control buttons are analyzing user's touches and if they hold down the button, and then send commands to mqtt server. The speed labels' values are changed by accelerating or braking. The drawing map button sends a request to server to record data for a mapping, and the second click stop the mapping. The menu button will connect to map table view controller when it pressed. The map table view controller creates a list of all

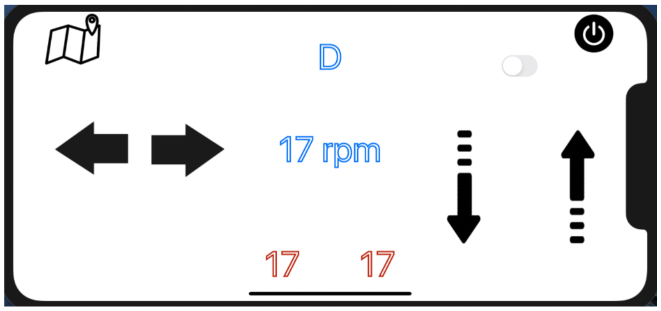
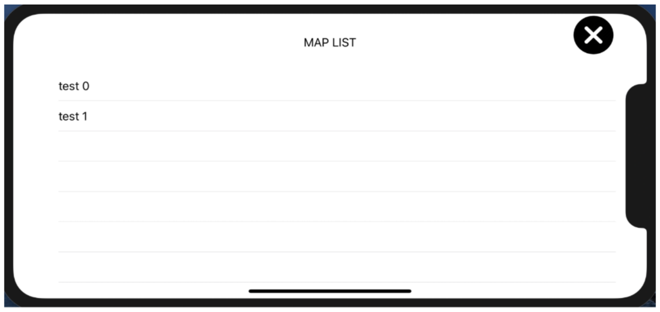


Fig. 14. Main view controller of IOS Application

map data that has been stored in the iOS device. Each cell is represented as a button so that when the cell is clicked the controller presents the map view controller to show the map. It also has exit button to terminate this controller. The map



MAP LIST

test 0

test 1

Fig. 15. Map list controller of IOS Application

view controller shows a map by using our mapping algorithm on an input data set. The car's movement history is drawn with white line, and obstacles detected by the sensors are drawn with a green line. It also has exit button to terminate this controller.

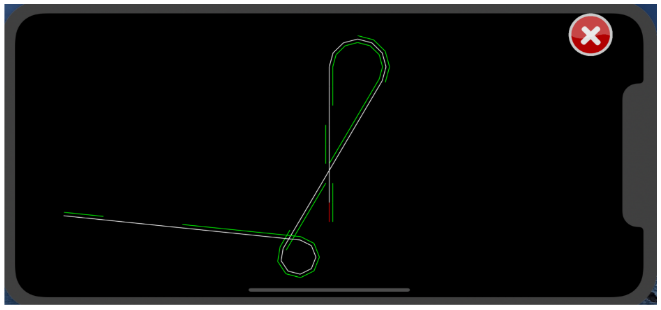


Fig. 16. Map view controller of IOS Application

E. Mapping Algorithm

Our mapping algorithm needed three different data inputs from the vehicle, the distance moved, the angle, and two detected walls some distance from the car. In this project, we measured each wheel's rotational distance and the two detected distances, then we drew a map using our mapping algorithm. The mapping algorithm has these steps: initiate position, set current angle of the car, set current position, resize map fit to phone screen, shift map fit to phone screen, and draw a map.

1) Initiate all coordination

At the beginning of mapping, it is supposed to be initialize all previous and current variables. To calculate the next position and angle, the program needs to set the first variable as first input in this section which is setting the current value as the previous value so that the program could use it on next iteration. The initial starting point is at the center of the phone screen, and angle to the north as 0 radius.

2) Set current angle and position

Since we couldn't find a reliable way to measure angle data, the program calculates it in this section by using the encoder values from the motors. The program already knows the previous points and current distances from the two wheels data; it needs an angular center point, a relative angle between two points, a previous center point and the current center point. Here is a graph of how this section is trying to find values.

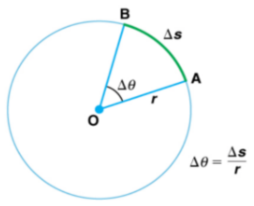


Fig. 17. Rotation angle and angular velocity [6]

Angular center could be determined by previous left wheel position (A) with angle ΔΘ. The current angle is sum of previous angle and ΔΘ.So, the result of angular center = $(prevCenter.x + r * cos(prevAngle), prevCenter.y - r * sin(prevAngle))$. From this, the program could get current car's center point which is $(angularCenter.x - r * curAngle, angularCenter.y + r * sin(curAngle))$.

After setting all current position of two wheels, the program sets detected objects which are orthogonal with car's direction vector. Therefore, it could get the point using a projection formula $u \cdot v = 0$ from left wheel and right wheel points.

3) **Resize map fit to phone screen**

The program then calculates the maximum ratio between phone screen's width / drawn map's maximum width and phone screen's height / drawn map's maximum height. The ratio will be adjusted to stored data array set and multiply the ratio to all points.

4) **Shift map fit to phone screen**

We then perform a check of the global maximum and minimum for x and y points, and if the global maximum or minimum exceed phone screen bounds, shift all points by the difference between maxima or minima and phone screen max point and phone screen min point. Now the entire map fits on the screen at one time.

5) **Draw map**

To draw the map, we connect between previous points and current points with lines. In the case of wall not detected which is -1 value of measured sensor distance, the program won't draw any lines.

IV. **Frame**

A. **Body**

Our frame design was based on three rules: 1. Easy to install, 2. stable, 3. Adapt to the components and wire connection. Therefore, we design a hollowed-polygonal frame which is shown below: As shown, the base was made of a piece of acrylic to provide stability for the rest of the frame and to hold the wheels and motors in place. It was designed to allow the wires from both motors to pass through the bottom and connect to the motor drivers inside the frame. The left and right sides are both screwed onto the base. There are three holes on each side with the same position. Two of these holes are used to hold the ultrasonic sensors and the other one is intended to facilitate airflow for cooling. The two holes at the front and the hollowed part at the bottom are also for airflow and cooling. We added these to the design because the Raspberry Pi heats up a lot when running all the programs necessary for our project.

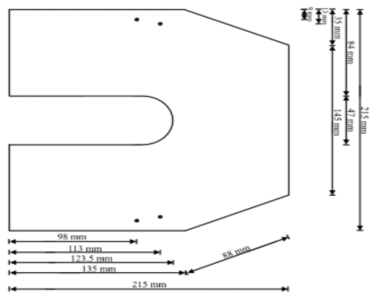


Fig. 18. Structure Chart of the Bottom (base)

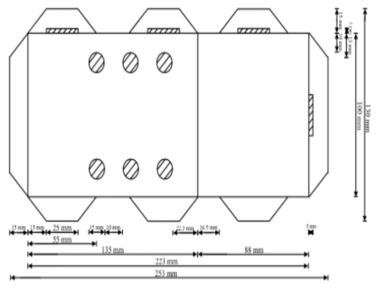


Fig. 19. Structure Chart of the sides

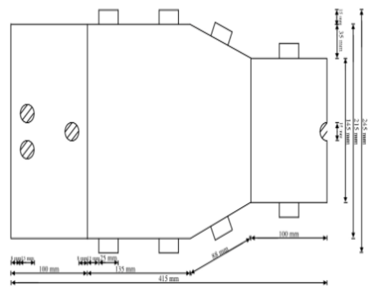


Fig. 20. Structure Chart of integrated front, top and back

For easy install and hardware adjustment (power banks charging), the front, top and back side are designed to be able to be put together via the slots on the left and right side. This allowed us to be able to remove the top quickly to check battery status, and wire connections easily.

B. **Motor and wheels**

For the motor and wheel parts, we used components from Pololu that were designed for our specific motors as well as some generic caster wheels to allow for easy turning:

- Pololu Universal Aluminum Mounting Hub for 6mm Shaft * 2
- Pololu Stamped Aluminum L-Bracket Pair for 37D mm Metal Gearmotors * 2
- Front Wheel: Pololu Wheel 9010mm Pair - Red * 2
- Rear Wheel: 1" Caster Wheels Swivel Plate w/Break Casters on Black Polyurethane Wheels * 2

Below is an image of the final product of the vehicle:

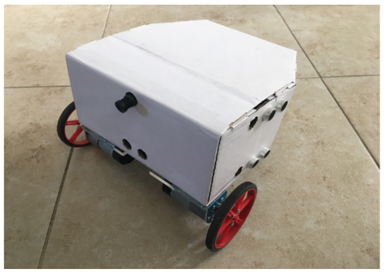


Fig. 21. Frame on finished project

V. **Conclusion**

The world today is built on information. Our project was designed around finding a way to gather information that would otherwise be unable to obtain. We did this through visual feedback with a camera as well as through a visual representation of the area the car was taken as a map. The project was very interesting as we worked through various issues and hiccups along the way. We ended up changing our microcontroller to a microcomputer and had to adjust and find different ways to do things along the way. It was great being able to incorporate our knowledge of both hardware and software together in one project. We were able to go through all the phases of project proposal, design, and implementation, and were able to create a successful project that met the requirements we had set forth in the beginning. This experience will help further our progress in this field and in our careers and future education.

References

[1] RaspberryPi, "Raspberry pi 4 microcomputer tech specs," 2019, https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/.

[2] pololu, "Pololu 2824 motor," 2019, https://www.pololu.com/product/2824.

[3] Pololu, "Pololu encoder," 2019, https://www.pololu.com/docs/0J63/3.4.

[4] Kuman, "Kuman for raspberry pi camera module," 2019, http://www.kumantech.com/.

[5] sparkfun, "Hc-sr04 ultrasonic sensor distance module," 2018, https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf.

[6] Unknown, "The radius of a circle is rotated through an angle . the arc length s is described on the circumference." 2016, https://courses.lumenlearning.com/physics/chapter/6-1-rotation-angle-and-angular-velocity/.