Road Interference Mapper (The RIM)

Team 35- Euiho Jung, Minh Le, Zhiyuan Zheng

ECE 445 Final Report - Fall 2019

TA: Kristina Miller

Abstract

The project, the Road Interference Mapper (The RIM), was designed to reduce evitable damages to vehicles or accidents by notifying drivers with the number of approaching interferences on the road. The information on the designed street was collected by an ultrasonic sensor and the noise of the interferences was filtered. The data was calculated in the microcontroller unit. The smartphone app was developed to convey the collected information to drivers. The RIM successfully detected the interferences at least 0.4m deep within +/-2 and alerted on the developed smartphone app. This system has the potential to raise awareness and to reduce the rate of vehicle damages and accidents caused by interferences on the streets.

Contents

1. Introduction

1.1 Purpose

In many places, there are multiple roads one can take to get to one's desired destination within a similar amount of time. Especially in many metropolitan cities where the road layout is a grid, there are countless paths to a given destination. However, in many places, certain roads are littered with potholes and debris. These roads with many interferences would ideally be avoided but there is currently no way to tell whether or not a road contains a large number of potholes and debris. Even government surveyors have no real way to find and count potholes besides manually counting them.

To allow for better driving path planning and pothole detection, our product allows government road surveyors to attach a device to their vehicle that will detect the number of interferences on the road and send that data to a server. Consumers can then view this data on their own smartphones using our app. Ultrasonic sensors periodically sense the average distance to the road and any deviation in this distance within a threshold signifies the presence of an interference. A microcontroller unit (MCU) periodically processes the distance data, and upon detection of an obstacle, sends a signal to a smartphone app notifying the app of the detected obstacle. The signal differs depending on if a pothole or a piece of debris was detected. The smartphone app then increments the number of potholes or debris, depending on the signal received, the surveyor has detected on the current road. This information is then sent to a server where the displayed number of potholes and debris on each nearby road is an average of all of the surveyor data on that road. All consumers using this app are able to see the average number of interferences on each road that the surveyors detected. However, we believed the server portion of this system was out of the scope of this class.

1.2 Functionality

Our product has three main requirements in order to function properly. The first requirement is that it must be able to detect potholes and debris of at least .25 m in width, .25 m in length, and .04 m in depth. This makes sure our system does not fail to detect interferences of a notable size. We chose these dimensions based on United Kingdom road data [1] and the average size of a tire [2]. Secondly, our product must be able to count the number of potholes and debris on a road within +/- 2 potholes and debris and display this data on a smartphone app. This is to ensure we get an accurate count on each road so that the data we collect and broadcast is useful data. Finally, the product must be able to detect potholes and debris while moving at speeds of up to 9.5 mph. This speed was chosen to allow the surveyors to move at a decent speed while also allowing the ultrasonic sensors to collect enough data to determine the existence of interferences.

1.3 Subsystem Overview

The RIM requires five main electrical components for operation as shown in figure 1.3. First, a 9 V battery is needed to supply power to a processing module at all times, Second, an ultrasonic sensor is required to detect potholes and debris on streets at speeds within 9.5 mph. Third, we

need a MCU to power the ultrasonic sensor and the Bluetooth module at 5 V, and to process all data from the sensor and the Bluetooth module. Fourth, the Bluetooth module will be

a data bridge to process between the MCU and a smartphone. Fifth, a smartphone/software is needed to exchange data with the Bluetooth module and count the number of potholes and debris for users.

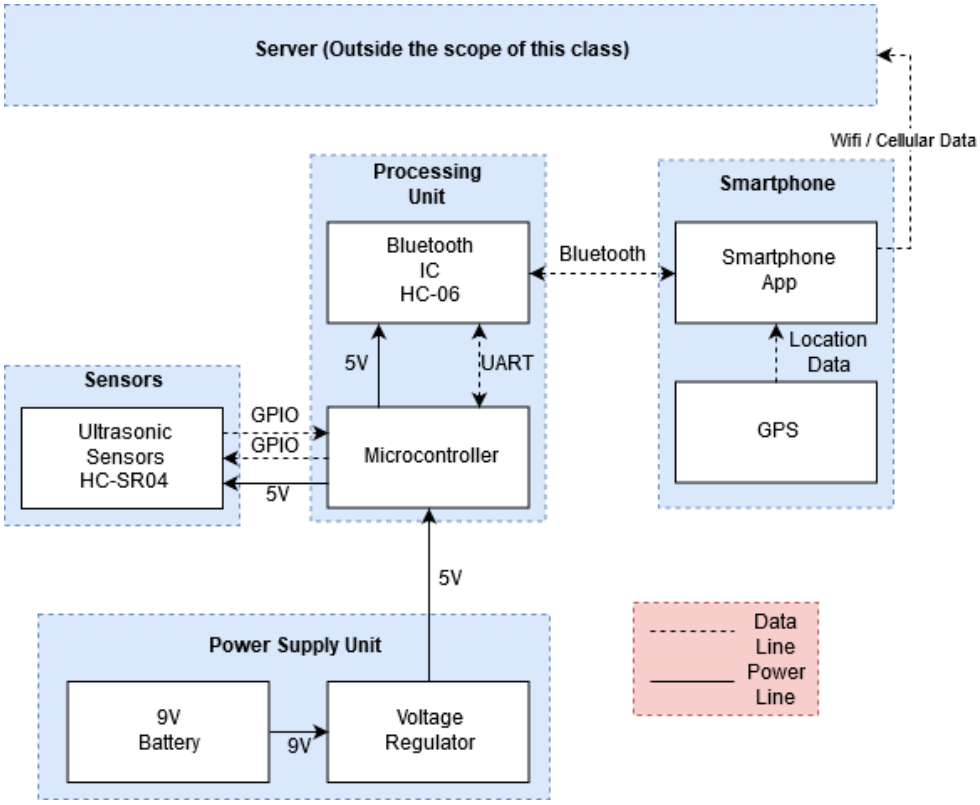


Figure 1.3 Overall Block Diagram

2. Design

2.1 Power Supply Module

9 V battery supplies power to the processing unit regulating 9 V to 5 V by a voltage regulator, which in turn supplies power to the Bluetooth module and an ultrasonic sensor. As the voltage regulator is a linear regulator, it converts a 9 V input power to a 5 V output which is supplied to the input voltage on the MCU; all components and modules in our system require 5 V power supply.

2.2 Processing Module

2.2.1 Processing Module Design Procedure

The design decision for MCU of the system was to add complexity in hardware to our project, instead of just using Arduino Nano. At the beginning of consideration for our project, it was decided that we use Arduino Nano as our MCU. However, this would simply lead our project to just connecting and wiring modules to each other and programing the sensor, the Bluetooth module, and the smartphone app. Also, there were some unnecessary output pins to be removed as analog pins were not used. By designing our own MCU, only necessary pins were drawn to be connected to our sensor and Bluetooth module. The MCU was built on Printed Circuit Board (PCB). On the PCB, through hole components, such as resistors, capacitors, and crystal oscillator, are used, as they were easily found in our lab. Even if the size of through hole components are much bigger than the size of surface mount components, they were still suitable in scale for the PCB because not too many components were placed on the PCB; the size of the PCB is 76 mm x 67 mm in length and width, so that even with the through hole components, the PCB still has spacious surface. On the PCB, bypass capacitor is used for the power supply to short AC noises, so only DC signal can be delivered. The Bluetooth module (HC-06) was used as its operating voltage is 5 V. However, voltage divider is connected to the data-in pin on the Bluetooth module as RX pin can only receive 3.3 V, but the MCU pin supplies 5 V through the pin; as the MCU pin can read 3.3 V as HIGH, the TX pin on the Bluetooth module can directly connected to the MCU. The first decision of the mode of the Bluetooth module was that we intend to turn it on and off with the smartphone app. However, it would add complexity to our system in software portion and there was no necessity as the system is turned on and off manually by connecting and disconnecting the power source for our demonstration; so the system can only be operated whenever in use. The bootloader (FTDI chip - LM1117IMPX-5.0) allows the MCU to communicate with a computer through the mini-B USB port, so the MCU can be coded to make calculations with data from the sensor and the Bluetooth module. The MCU (ATMEGA328P-AU) is chosen to receive data from the sensor, to make calculations of the data and to communicate with the smartphone block through the Bluetooth module. This chip is equipped with enough digital pins required for our project.

2.2.2 Processing Module Design Details

As shown in Appendix A and Appendix B**,** the processing module consists of a MCU (ATMEGA328P-AU), a bootloader chip (FT232RL), a Bluetooth module(HC-06), and a mini-B USB port. The processing module performs critical calculations with data collected by the ultrasonic sensor in order to determine whether or not interferences are detected; the noisy data is filtered out. The information of the interference calculated in this module is sent to the smartphone block to be displayed on the smartphone app for users. The Bluetooth module operates with the Bluetooth 2.0 protocol at an operating frequency of 2.4 GHz in the ISM frequency band [3]. As shown in Appendix B, The MCU is to communicate with ultrasonic sensors and the Bluetooth module. The crystal oscillator connected to pin 7 and pin 8 on MCU generates the 16MHz clock signal. The left three connectors are to connect sensors to the MCU

and the Bluetooth pin connector is to connect the Bluetooth module to the MCU for data transmission with the smartphone block via digital pins on MCU.

2.3 Sensor Module

2.3.1 Sensor Design Procedure

The sensors of The RIM were chosen to achieve one objective: detect change in distances. In the preliminary stages of design, we considered infrared, LIDAR, and ultrasonic sensors to detect change in distances; we chose ultrasonic sensors. Although ultrasonic sensors don't measure distances perfectly, the low cost of the unit combined with the fact that sound is less prone to noise than light led us to choose ultrasonic sensors over the more expensive LIDAR and noisier infrared sensors.

With our choice of sensors narrowed down to ultrasonic, we then shifted our focus to the placement of the sensors, the number of sensors we were going to use, the operating speed of the sensors, and how we were going to process the sensor readings to detect change in distances. To find the optimal placement and number of sensors, we studied the sensor's range behavior and spread behavior described by equations 2.1 and 2.2 respectively. To find the maximum speed at which a vehicle can move for the sensors to operate correctly, we studied the sensor's sampling rate governed by equation 2.3. Given a sampling rate, we can then calculate the maximum operating speed of the sensors using equation 2.4. To figure out how to use the sensor's readings to discern changes in distances, we designed a detection algorithm mapped out in figure 2.1

$$Time\ Delay(\mu s) = Time_2(\mu s) - Time_1(\mu s)$$

Equation 2.1 Travel time of ultrasonic sensor pulse

$$w(cm) = 2sin(\frac{\theta}{2})\frac{d(cm)}{cos(\frac{\theta}{2})}$$

Equation 2.2 Ultrasonic sensor spread

$$Sampling\ Rate(\frac{1}{s}) = \frac{Number\ of\ samples\ collected}{Time(s)}$$

Equation 2.3 Ultrasonic sensor sampling rate

$$OperatingVelocity_{max}(\frac{m}{s}) = SamplingInterval_{min}(m) \cdot SamplingRate(\frac{1}{s})$$

Equation 2.4 Maximum Ultrasonic Sensor Operating Speed

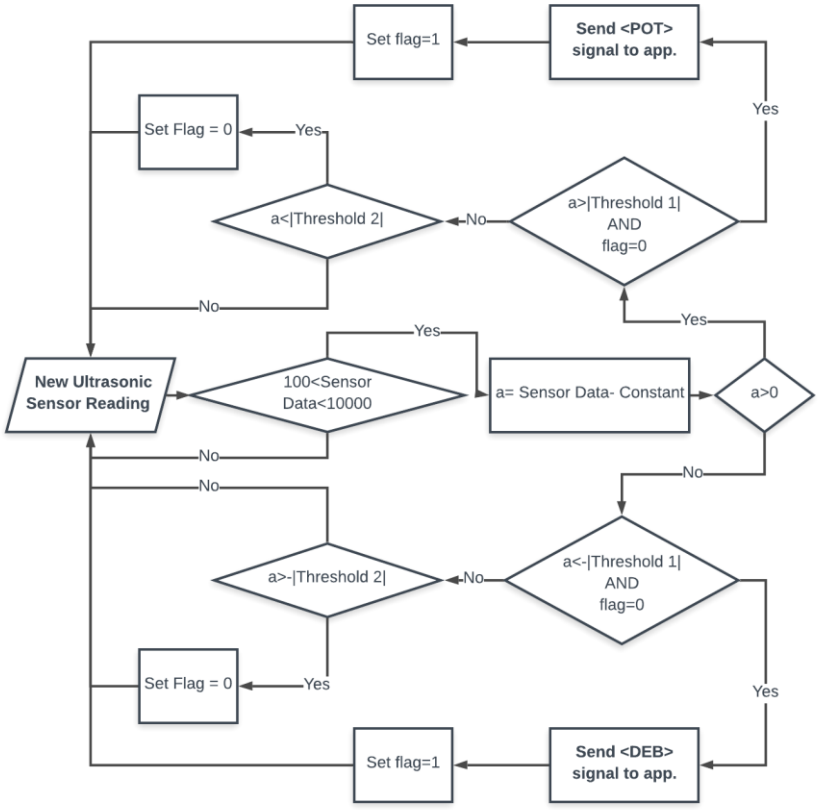


Figure 2.1 Interference detection algorithm

2.3.2 Sensor Design Details

2.3.2.1 Sensor Physical Design Details

First, to determine the optimal height to mount the ultrasonic sensors, we must understand how ultrasonic sensors use sound waves to measure distances. Sound waves produced by an ultrasonic sensor leave through the transmitter, bounce off the closest object in front of the sensors, and return to the ultrasonic sensor's receiver; this process is portrayed in figure 2.3. Time one in equation 2.1 denotes the time at which a sound wave leaves the sensor's transmitter and time two denotes the time at which the sound wave returns to the receiver. The delay between time two and one gives us a sense of how far an object is from the sensors. Ideally, the further an object is from the sensor the greater this time delay will be and the closer an object is from the sensor the shorter this time delay will be. This time delay depends directly on the trajectory of the sound wave being transmitted and received by the ultrasonic sensor. In turn, trajectory is affected by the geometry of the object the sound waves are hitting, the distance the object is from the sensor, and external sound interferences.

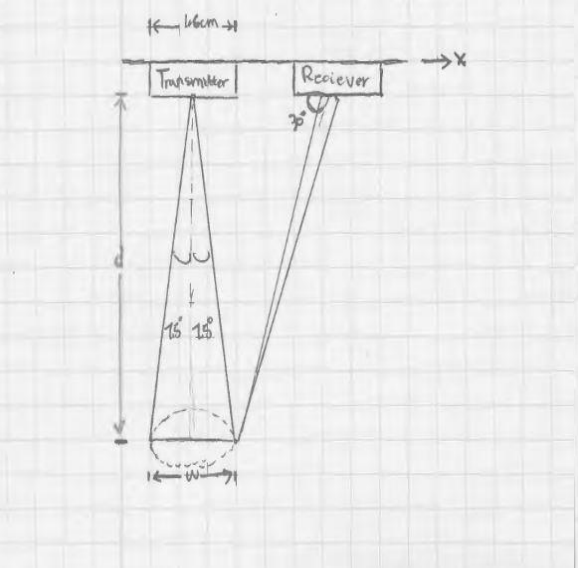


Figure 2.2 Ultrasonic sensor side view

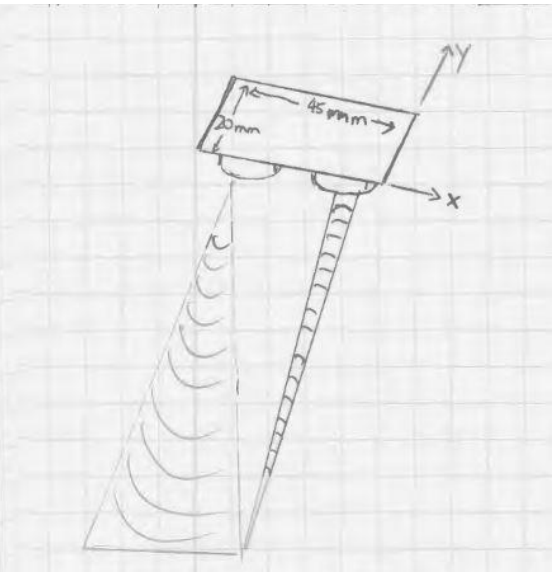


Figure 2.3 Ultrasonic sensor top view

According to the sensor's manufacturer's documentation, the sensors should accurately measure distances between 2 cm and 400 cm [4] so we initially chose to mount the sensors 40 cm above ground level corresponding to the height of the chair seat on which we tested our device. After empirical tests explained in section 3.3.1, we found that the sensors doesn't operate below 40 cm and so we raised the sensors to a height of 90 cm above ground level for the final implementation.

Next, in order to determine the optimal number of sensors to use, we applied equation 2.2 which computes the width (w) of the floor an ultrasonic sensor would be able to probe when placed at height d above the floor given that the sensor's measuring angle is Θ. These parameters are illustrated in figure 2.2. Using the initial height we decided to mount the sensors at (d=40 cm), we found that a single ultrasonic sensor unit will be able to probe a horizontal width of w=10cm of the ground underneath it. With the goal of probing the entire width underneath the test vehicle in mind, we initially decided to use four sensor units.

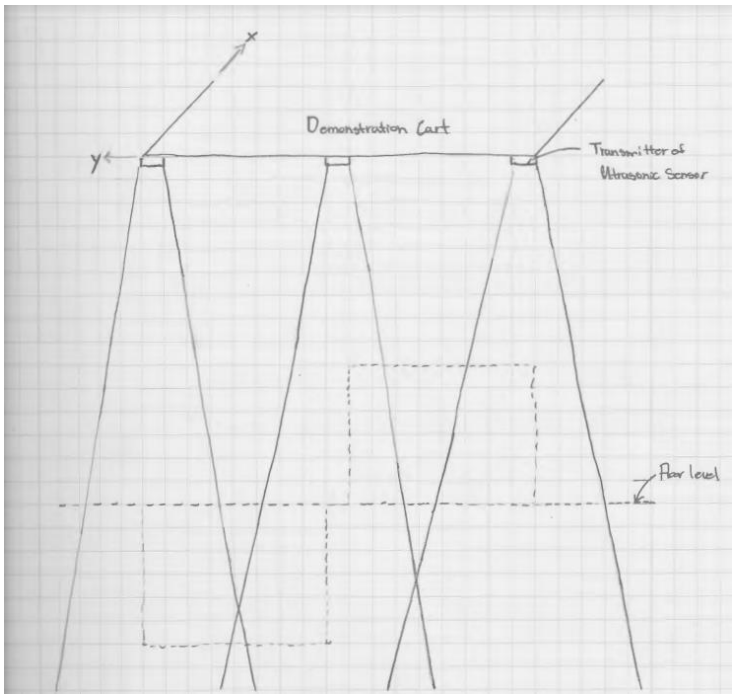


Figure 2.4 Realization Illustration

After taking into account the geometry of the interferences we are trying to detect and performing tests on the ultrasonic spread behavior described in section 3.3.2 , we realized that only one center sensor is needed for our device to function. This realization is illustrated in figure 2.4.

Finally, to determine the maximum velocity that the sensors can travel at for it to function correctly, we applied equation 2.4. This equation calculates a maximum operating velocity based on the sampling rate of the sensors and the minimum distance interval between which we want to get samples. Using the HC-SR04's suggested measurement cycle of 60 ms the sensors should theoretically exhibit a sampling frequency of 17 Hz [4]. If we aim to obtain a ground-distance reading every 0.25m, equation 2.4 tells us an ultrasonic sensor sampling at 17hz will only be able to detect interferences under a car moving up to a velocity of 4.25 m/s. This result assumes that every single sample the ultrasonic sensor returns is accurate. Due to noise, the effective sampling rate is lower than 17 Hz and thus the actual operating velocity of The Rim is below 4.25m/s.

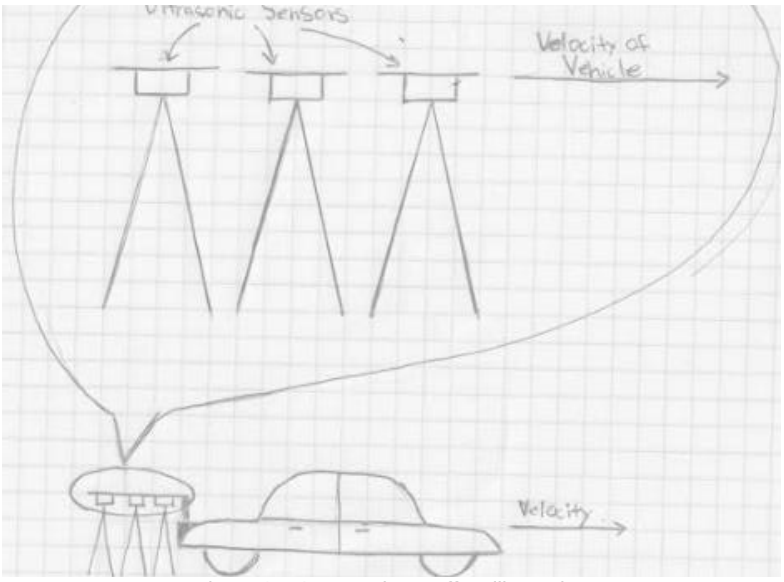


Figure 2.5 Sensor phase offset illustration

To compensate for this sampling rate reduction, we could've placed multiple ultrasonic sensors adjacently one behind the other (shown in figure 2.5) with a phase offset introduced between sensors. This phase offset sensor configuration would multiply the effective sampling rate achieved with one sensor by the new number of sensors; this solution was not implementable due to the limitation of having only one functioning digital pulse width modulation pin on our printed circuit board. We could've also switched to LIDAR sensors which inherently have much higher sampling rates than ultrasonic sensors. In the end, due to monetary and time restraints, we had to make due with one ultrasonic sensor. Consequently, The RIM in its current state has to move at a slower speed to detect interferences. This speed is dependent on the number of accurate data samples the ultrasonic sensor is collecting in any given timer interval.

2.3.2.1 Sensor Data Processing Design Details

After the physical design ultrasonic sensors were taken into account, we then created a detection algorithm mapped out in figure 2.1 that would use the ultrasonic sensor's data to detect change in distances caused by potholes and road debris. The function of our final detection algorithm depends on 4 main variables: constant, threshold 1, threshold 2, and flag. Constant is set equal to 4770 µs and describes the average time delay returned by the ultrasonic sensors when The RIM is over level ground. Threshold 2 is set equal to 100 µs and describes the average deviation from the constant in an ultrasonic sensor's reading when The RIM is over level ground. Threshold 1 is set equal to 200 µs microseconds which describes the minimum difference in sensor readings from the constant when it is 4 cm below ground level. Flag is a binary variable that prevents double counting; when an interference is detected, flag gets raised to 1 and isn't lowered again to 0 until The RIM detects that it is once again over level ground.

As The RIM travels along a road, it's ultrasonic sensors constantly probes the ground and returns data points in the form of a time delay. The detection algorithm subtracts the variable constant from the data point and compares it to threshold 1 to decide if an interference of at least 4 cm in thickness was present when the data point was taken. If an interference wasn't detected, the algorithm then compares this same difference to threshold 2 to determine if it is back on level ground.

2.4 Smartphone Module

2.4.1 Smartphone Design Procedure

The main design decisions for the smartphone block of the system revolve around the GPS and location mapping portions of the system. The first decision was which GPS unit to use to get the current longitude and latitude coordinates of the user. We could have either used the smartphone GPS capabilities or a separate GPS chip. Using a separate GPS chip would provide a more accurate location data, however, it would have added significant complexity to the system as well as introduce latency from the GPS chip's necessity to communicate data to the smartphone. We decided it would be better to use the smartphone GPS as the accuracy was believed to be sufficient and the data would be available almost instantly. The smartphone GPS would provide the location data necessary to map each interference detected on an onscreen map. However, after testing the accuracy of the smartphone GPS with a public Google play app [5], as shown in table 2.1**,** it was found that the smartphone GPS would not be accurate enough to get a good mapping of the interferences detected. Also, we failed to consider the fact that since our system would be on a moving vehicle, the location data would be even more inaccurate as the system would move away from the location of the interference as data propagated to mark the interference. Due to the inaccuracy of the smartphone GPS, we decided to change the scope of the project. Instead of mapping the exact locations of each interference, our system was changed to count the number of interferences on each road. This change relaxed the accuracy requirements of our GPS system to require accuracy to the closest road instead of to a specific point. It also helped to keep the complexity of this project within the scope of our class.

After changing the scope of the project, we had to determine the best way to get the current road based on the users current latitude and longitude coordinates. This process of converting latitude and longitude coordinates to road names is called reverse geocoding. There were many reverse geocoding APIs available and we narrowed it down to four, Nominatim [6], Google Nearest Road [7] , Geonames [8], and Mapbox [9]. The Nominatim reverse geocoding API offered a fast response time and accurate results because of the expansiveness of the Open Source Maps database. The responses, however, were fairly large. The Google Nearest Road API had the largest database of the reverse geocoding options and had compact responses. However, this was a paid service and was prone to slowdowns due to needing verification for each request. Geonames had a smaller database than the other two

aforementioned APIs, however, it had customizable response parameters which allowed for simple and easy to parse responses. Finally, Mapbox also seemed to provide accurate results and easy to parse responses. However, the responses contained a large amount of extraneous data that made responses take a long time. The responses for the same input in all of these APIs are located in Appendix G**.** In the end, we decided to use the Nominatim reverse geocoding API for its accuracy and speed of responses. These two factors were the most important for ensuring the system was detecting the correct road and would change the current road quickly if the user changed roads.

Location	Moving?	Error (m)
Residential House	X	3.2
ECEB	✓	4.3
Outside	X	4.9
Outside	✓	6.0 ~ 8.0

Table 2.1 Smartphone GPS Accuracy Table

2.4.2 Smartphone Design Details

We used Java to design the smartphone app in Android Studio. The smartphone app consists of two main components, the Bluetooth communication module, and the location tracking module. The Bluetooth communication module allows the smartphone to communicate with the Processing module to update the pothole and debris counts on the current road upon the reception of a Bluetooth signal. While the location tracking module keeps track of the current road the user is on and displays the correct pothole and debris counts according to that current road. The source code for the smartphone app is found in Appendix E. Figure 2.6 shows the overall flowchart for the smartphone app.

The Bluetooth communication module uses the Client Bluetooth Library [10]. This library allowed the smartphone to connect with the Bluetooth module in the processing unit and communicate with the module as needed. This library allowed for the synchronous reception of a Bluetooth signal in the form of a text string. Upon receiving a text string, the smartphone app will check whether the signal corresponds to a pothole or a piece of debris. Upon receiving the text string “<DEB>” or “<POT>”, the app will increment the count of debris or potholes respectively for the current street.

The location tracking module utilizes a pipeline of Google Location Manager [11], Volley [12], Nominatim, and back to Volley. First, the app uses Google Location Manager to get the

current latitude and longitude coordinates of the user. Google Location Manager uses either the network or the GPS of the smartphone get these coordinates depending on which one is more accurate. Outside the GPS will be more accurate, and inside a building, the network will be more accurate. Then, we use the Volley library to construct and send a Javascript Object Notation, or JSON, request to the Nominatim API through the internet. The Volley library properly formats the latitude and longitude coordinates received in the first step into a format the Nominatim API can parse. After this request is sent, the app must wait for the reverse geocoding response from Nominatim. Upon receiving a response, the Volley library will parse the JSON response for the street name. With this information, the app can update the current street and the current pothole and debris counts displayed to the user. The app repeats this process periodically to ensure the user has not changed roads. The workflow for the location tracking feature of the smartphone is found in figure 2.7**.**

These modules allow the smartphone app to display and update the number of potholes and debris on the current road upon receiving each Bluetooth signal sent from the processing module.

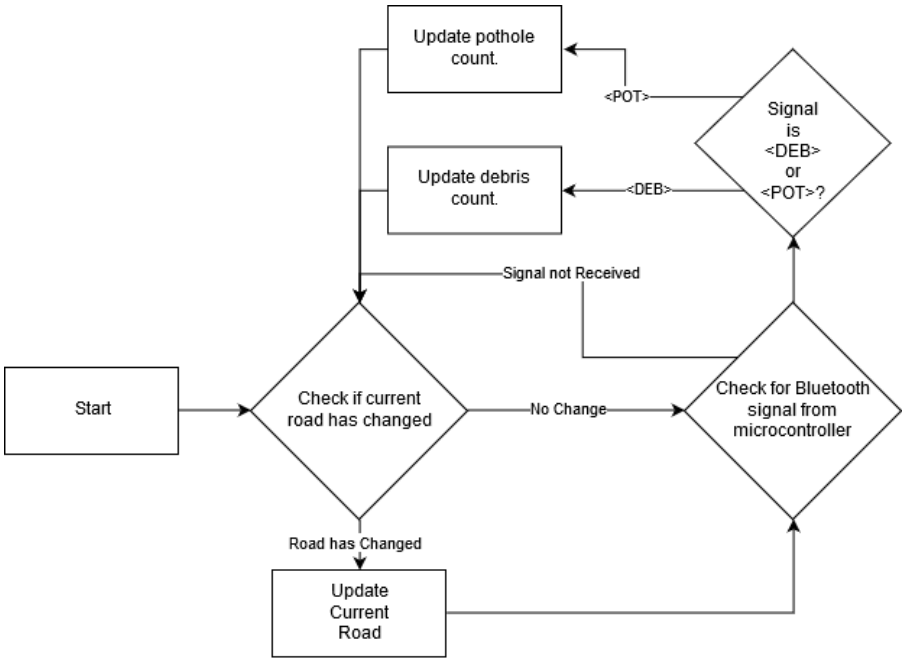


Figure 2.6 Smartphone App Flowchart

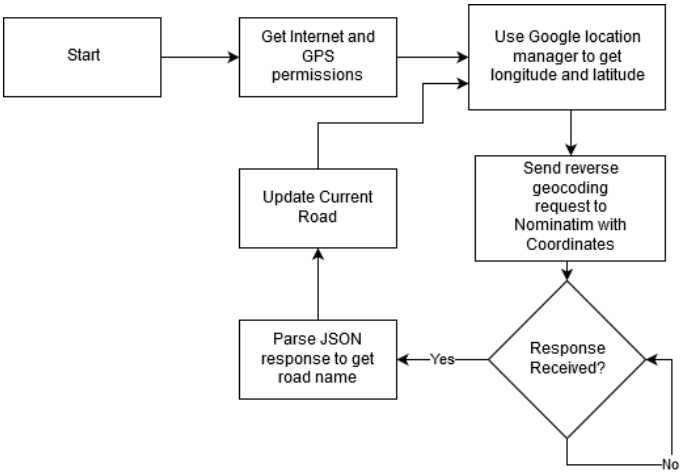


Figure 2.7 Smartphone Location Tracking Flowchart

3. Design Verification

3.1 Power Module

For a successful project, constantly supplying power would be the most critical process to be done. As shown in figure 3.1, the top red led on the right hand side lights up when the power is supplied to the PCB after regulation through the voltage regulator. Another method to check if the power is supplied is to simply connect the Bluetooth module or the ultrasonic sensor; if the Bluetooth module blink the light on itself or if the MCU receives data from the sensor, it means that the power is being supplied.

Figure 3.1 Top view of the PCB

This voltage regulator has a low efficiency which is about 46.9% by using the equation 3.1.

$$V_{in} = 8.0V\ I_{in} = 1.2A\ V_{out} = 5.0V\ I_{out} = 0.9A$$

$$Efficiency = \frac{output\ power}{Input\ power} \times 100 = \frac{V_{out} \times I_{out}}{V_{in} \times I_{in}} \times 100$$

Equation 3.1 Efficiency of the Voltage Regulator

3.2 Processing Module

In order for this processing module to perform its role in the system, all the requirements should be met. First, the distance data from the ultrasonic sensor should be collected in the MCU. Second, the interference detection algorithm on the MCU should detect the number of the interference on a given street within +/-2 interference. Third, The Bluetooth chip should receive all signals from the MCU and send them to the smartphone app within 2 second. Fourth, The Bluetooth chip should receive all signals from the smartphone app and send them to MCU within 2 seconds. In other words, the communications between the MCU and the sensor, and between the MCU and the smartphone through the Bluetooth module. To verify this, the system should be actually run and ultimately should be able to display the number of potholes and debris respectively, as shown in figure 3.2.

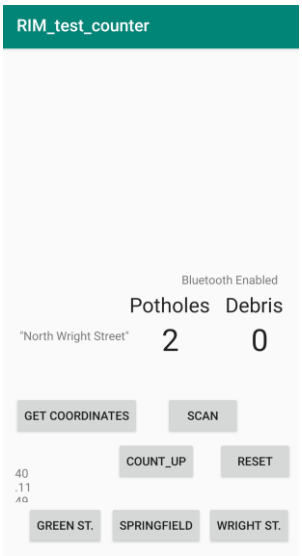


Figure 3.2 Picture of Smartphone App Display

3.3 Sensor Module

3.3.1 Ultrasonic Sensor Range Behavior

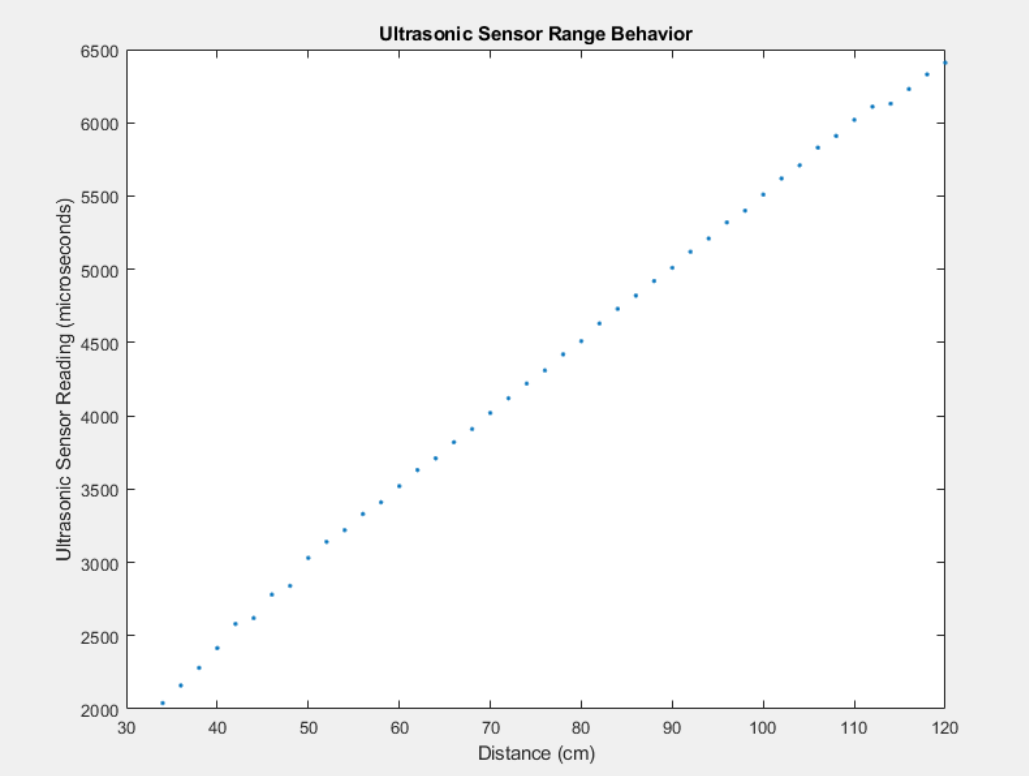


Figure 3.3 Ultrasonic Sensor Range Behavior

To test the ultrasonic sensor's range behavior, an ultrasonic sensor was moved from 1 cm to 120 cm away from the ground in 2 cm intervals. The echo duration output (difference between the time the transmitter sent a signal and the time the receiver collected an echo) was recorded at each of these distance intervals and plotted in figure 3.3. From the sensor module's requirement and verification table, we originally required the ultrasonic sensors to function between a range of 2-400cm. However, after empirical tests, the sensors were found to be noisy to the point of being unusable below ranges of 34 cm. In addition, the ultrasonic sensors readings had a margin of error of up to +/- 100 μs at all distances which limits the sensor's accuracy as well as resolution.

3.3.2 Ultrasonic Sensor Spread Behavior

Edge of Book (cm)	Sensor 1 (µs)	Sensor 2 (µs)	Sensor 3 (µs)
30	4770	4800	4990
31	4800	4820	5010
32	4780	4810	5010
33	4790	4790	5000
34	4810	4800	4810
35	4820	4770	4810
36	4790	4800	4820
37	4810	4810	4790
38	5000	4800	4810
39	4950	4820	4810
40	5020	4790	4800
41	5010	4810	4800

Table 3.1 Ultrasonic sensor spread behavior test results

To test the ultrasonic sensor's spread behavior, we placed three ultrasonic sensor units along the edge of a table as they would be mounted on The RIM. We then moved a styrofoam block 2"x4"x12" across the width of the bottom of the chair in 1 cm intervals and recorded the sensor's reading for each position as illustrated in figure 3.3 which can be found in Appendix D. The results are shown in table 3.1. Based on the data in table 3.1, no matter what position the interference is at, two sensors are able to detect its presence. Furthermore, the center sensor

will be able to detect the interference at all positions if the interference is fully underneath the test vehicle.

3.3.3 Ultrasonic Sensor Sampling Rate

Time interval number:	Samples Collected
1	180
2	183
3	162
4	179
5	170
6	186
7	165
8	167
9	179
10	183
Average number of samples/ second	17.54

Table 3.2 Ultrasonic sensor sampling rate test results

We test the ultrasonic sensor's sampling rate using a counter that kept track of the number of readings the Arduino UNO received from the HC-SR04 ultrasonic sensors in 10 seconds. We recorded the number of readings collected over ten, ten second intervals in table 3.2. Based on the data in table 3.2, we found that the sensors delivered, on average, 17.54 readings every 10 seconds. In the sensor modules requirement and verification table, we required that the sensors achieve a sampling rate of 17 Hz so that any vehicle The RIM is attached to can travel up to 4.25 m/s. Even though the ultrasonic sensors achieved a sampling rate of 17 Hz, not all the samples were accurate data points that could be used for interference detection which implies the effective sampling rate of the ultrasonic sensors is actually below 17Hz.

3.4 Smartphone Module

In order for the system to function, the smartphone app had to meet requirements on the two main modules, the Bluetooth communication module and the location tracking module. These requirements and verifications are shown in Appendix F. The Bluetooth communication module had to receive signals quickly and without mistakes. We tested this with a test program loaded onto an Arduino Uno, shown in Appendix H**.** We sent multiple text strings of different lengths to the smartphone through the Bluetooth module connected to the Arduino and observed the time it took to receive the signal. With each string, the time to receive the signal was near instant, less than 0.5 seconds. With this response time, the Bluetooth communication module on the smartphone app is effective at vehicle speeds much greater than required. This means the Bluetooth communication will not be a hindrance to the efficacy of the project.

We also needed the location tracking module to identify the current road the user was on with a high degree of accuracy. To test this, we added a modification to the app that allowed us to update the displayed current road after pushing a button. This is the “GET COORDINATES” button shown in the smartphone UI, located in figure 3.2**.** We then tested the app at various locations and observed the response time and the accuracy of the current street. Table 3.3 shows the results of our testing. The results show that the location tracking module is accurate to the closest street almost every time. However, when testing on smaller streets, it was found that the location tracking would display a nearby major street rather than the small street we were on from time to time. Since our system works for all other cases a vast majority of the time, we feel that this accuracy is suitable for the project. To improve this accuracy we would have to implement a separate GPS module with a higher accuracy and use a more premium reverse geocoding API. At that point, it would be more beneficial to implement the original goal of the project, mapping the exact locations of each pothole and debris. Finally, we found the response time for the location tracking module to be far under our requirement.

With all requirements for the smartphone block of our system met, the smartphone portion of the system works flawlessly when implemented with the rest of the system. When we test our system over our constructed environment, the app counts each pothole and debris when given a Bluetooth signal from the processing unit. The app is also able to display the current road the user is on and update the number of potholes and debris displayed depending on the current road.

	Location Tracking Testing	
True Location	Displayed Location	Response Time (s)
Green St.	Green St.	2.6
Springfield Ave.	Springfield Ave.	2.4

S. Busey Ave.	S. Busey Ave.	3.0
Wright St.	Wright St.	2.5
W. High St.	W. High St.	2.7
1st St.	Green St.	2.1
2nd St.	2nd St.	3.1

Table 3.3 Smartphone Location Tracking Verification

4. Cost

4.1 Parts

Description	Manufacturer	Part #	Quantity	Unit Price($)
MCU Unit	Microchip Technology	ATMEGA328P-AU	1	2.14/ea
Bootloader	FTDI	FT232RL-REEL	1	4.50/ea
Bluetooth Module	DSD Tech	HC-06	1	7.99/ea
Ultrasonic Sensor	Adafruit Industries LLC	HC-SR04 (4007)	4	3.95/ea
PCB	PCBway		1	5.00/10 items
Adhesive	3M		1	9.29/ea
9V Battery	Energizer		1	3.10/ea
9V Battery clip	Gonioa	2.1mmx5.5mm	1	0.58/ea
9V Battery Adapter	Duttek		1	6.99/ea

Mini-B USB port	CUI	490-UJ2-MBH-1-SMT	1	0.49/ea
Speaker Grill Cloth	The Wire Zone		1(6ftx3ft)	10.95/ea
Plastic Box	Saim		1	1.80/ea
Voltage Regulator	Texas Instruments	926-LM1117IMPX50NOPB	1	1.10/ea
Schottky Diode	Vishay Semiconductor Diodes Division	SS1P3L-M3/84AGICT-ND	1	0.45/ea
Resonator	Murata Electronics	CSTCR6M00G53Z-R0	1	0.46/ea
1k Ohm Resistor	EDGELEC	E10P004	1	0.80/10 items
100nF Capacitor	BOJACK	B07X5	1	0.80/10 items
1uF Capacitor	LATTECH	B074LZWRV5	1	0.70/5 items
Green LED	Kingbright	APT2012SGC	2	0.37/ea
Yellow LED	Kingbright	APT2012YC	1	0.37/ea
Red LED	Kingbright	APT2012SRCPRV	1	0.41/ea
Total Cost				**74.09**

Table 4.1 Cost Analysis of the Components

4.2 Labor

Team Member	**Hourly Wage**	**Weekly Hours**	**Number of Weeks**	**Cost per member**
Michael	$35	15 hours	14 weeks	$7,350

Minh	$35	15 hours	14 weeks	$7,350
Ethan	$35	15 hours	14 weeks	$7,350
Total Cost				**$22,050**

Table 4.2 Cost Analysis of the Labor

4.3 Schedule

Week	Michael's task	Minh's task	Ethan's Task
10/7	Test ultrasonic sensor spread and sampling rate	Created rough draft of detection algorithm	Create schematic and board for PCB
10/14	Calibrate sensors; Research how to interface sensors with an app we will create	Interfaced Bluetooth with microcontroller	Research how to interface the phone's gps with an Android app we will create
10/21	Mount hardware on PCB	Interfaced app with Bluetooth module	Create demonstration environment
10/28	Test power and sensor modules	Tested smartphone app Bluetooth capabilities to count interferences	Test processing module
11/4	Interface hardware with software	Begin smartphone app location tracking	Interface hardware with software
11/11	Test product and make necessary adjustments	Implemented current road software on smartphone app	Test product and make necessary adjustments
11/18	Make adjustments based on mock demo feedback	Interfaced hardware with software	Make adjustments based on mock demo feedback

Table 4.3 Schedule of Team Members

5. Conclusion

5.1 Accomplishments

As shown in the previous sections, most of our modules met all of our requirements. The power module functions as designed and adequately provides power to all of the other modules without damaging any components. The smartphone module receives all incoming Bluetooth signals from the processing module and updates the pothole or debris count corresponding to the type of Bluetooth signal received. The smartphone module is also able to identify the current road the user is on with a high degree of accuracy for major roads. The app also properly updates the number of potholes and debris displayed when the current road changes. The processing module PCB works perfectly. The processing moduleis able to receive all incoming sensor data, use the sensor data in the detection algorithm to determine if there is a pothole or debris present ,and send Bluetooth signals to the smartphone when an interference is detected. The detection algorithm maintains our required accuracy of +/- 2 potholes and debris when counting interferences of at least .25 m in length, .25 m in width, and .04 m in depth. The sensors are able to get semi-accurate distance data, however, the effective sample rate is much lower than expected. The sensors still give enough reliable data to detect potholes and debris within our size requirement and only affect the speed at which we can detect interferences.

The full system is able to detect potholes and debris of our required size when moving at 4 mph or less on our test environment. The final count of potholes and debris on each run over our test environment is always within the +/- 2 potholes and debris bound, barring outliers. Overall, our product will accurately count the number of potholes and debris on the current road when the system is moving at speeds of up to 4 mph.

5.2 Uncertainties

The biggest uncertainty surrounding The RIM is the behavior of its ultrasonic sensors. The RIM depends on its detection algorithm to detect interferences. The detection algorithm needs accurate data from the sensors to correctly indicate whether a pothole or road interference is present. The ultrasonic sensor’s limited resolution, periodic inaccuracy, and limited sampling rate described in section 3.3 are all sources of uncertainty that prevent The RIM from operating consistently and quickly.

5.3 Ethical Considerations

There are a few ethical and safety concerns with our project. The sensor mount could fall off of the vehicle and potentially damage the vehicle or other vehicles if it becomes completely unattached. Extensive stress testing and a strong adhesive or mountain apparatus will remedy this potential hazard. Also, in accordance with Illinois state laws, the sensor mount will not at all obstruct the view of the front license plate [13]**.**

First, as with all batteries, the 9V battery used to power the system could overheat or melt due to an electrical short. First, the circuit was carefully designed as to prevent shorts. Then every component of the circuit was extensively tested to ensure there are no potential shorts in the circuitry of the system.

Secondly, our app could be a potential distraction for the driver and may cause accidents. To mitigate this, our app reminds the users to pay attention to their surroundings while using the app. This will dissuade users from putting too much focus on the app when driving.

Finally, if given to consumers, they could use the app to generate false interferences and deter others from using those roads. This would also give a false flag for necessary repairs on roads that may not need them. This would go against #9 of the IEEE code of ethics [14]. To solve this we only allow government surveyors to input data into the app. This way, malicious persons will not be able to tamper with the data.

Our solutions to potential safety and ethics concerns follow #1 of the IEEE code of ethics, “to hold paramount the safety, health, and welfare of the public, to strive to comply with ethical design and sustainable development practices, and to disclose promptly factors that might endanger the public or the environment;” [14]. There are many potential risks in a product that is to be used while operating an automobile, but we feel that our mitigations allow the benefits to outweigh the potential problems with such a system.

5.4 Future Work

The sampling rate and noisiness of the ultrasonic sensors are a significant detriment in the efficacy of our project. The sensors output unreliable data a large amount of the time which renders most samples useless. This lowers the effective sampling rate to a point that the product becomes ineffective at high speeds. In order to improve this we have two options. The first option is to continue using the ultrasonic sensors and instead put them in an array. If we set up the sensors out of phase with each other we should be able to improve our effective sampling rate. This could allow the product to function at higher speeds. However, this would also increase the complexity of the detection algorithm. The increase in hardware complexity and incoming data could slow down the detection algorithm and the system as a whole. The other option is to use a more expensive and effective sensor, for example a LIDAR sensors. This would allow us to keep the number of sensors and incoming data to the detection algorithm low while increasing the sampling rate and accuracy of our sensor module. This would also significantly increase the speed our system could function at. However, this would make the product much more expensive.

Another improvement to the project would be to implement the original goal of mapping the exact location of each pothole and debris and displaying those locations to the user. In order to achieve this we would need to use a better GPS chip. We could use a more accurate GPS chip to get more accurate latitude and longitude coordinates and perform the calculations required to pinpoint the location of the interference. This would add a significant amount of complexity to the project, but it would be a huge improvement in the usefulness of our product.

Finally, we would have to scale up our project for commercial use. This would require widening the array of sensors and adjusting the detection algorithm to take the data of multiple sensors as input. If the above improvements are implemented, the current microcontroller may be able to support all the necessary components, however, we may have to replace

microcontroller with one with more pins to support the addition of new components and sensors.

Also, with the increase in input data, we may need to utilize a microcontroller with a faster computing speed in order to keep the system fast. Lastly, it would be beneficial to upgrade the power system to support all the new components. We could replace the 9 V battery with a connection straight to the vehicles power source. Then we would replace the linear voltage regulator with a more efficient option to match the upgraded power source.

With these improvements, our product will be commercially viable and a valuable resource for drivers and government agencies looking to improve road conditions.

References

[1] “Council response times to potholes,” *RAC Foundation RSS2*, 18-Jan-2019. [Online]. Available: https://www.racfoundation.org/research/mobility/council-response-times

[2] https://tireguides.com, 2019. [Online]. Available: https://tireguides.com/TireTips/TireDocument/11 [Accessed: 12- Oct- 2019] -to-potholes [Accessed: 13-Oct-2019].

[3] *Olimex.com*, 2019. [Online]. Available: https://www.olimex.com/Products/Components/RF/BLUETOOTH-SERIAL-HC-06/resources/hc06.pdf. [Accessed: 19- Sep- 2019].

[4]*Cdn.sparkfun.com*, 2019. [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf. [Accessed: 19- Sep- 2019].

[5] https://play.google.com, “GPS Status & Toolbox”, 2019. [Online]. Available: https://play.google.com/store/apps/details?id=com.eclipsim.gpsstatus2 [Accessed: 29- Sep- 2019].

[6] https://nominatim.openstreetmap.org, “Nominatim”, 2019. [Online]. Available: https://nominatim.openstreetmap.org/ [Accessed: 12- Nov- 2019].

[7] https://developers.google.com, “Nearest Roads”, 2019. [Online]. Available: https://developers.google.com/maps/documentation/roads/nearest [Accessed: 12- Nov- 2019].

[8] https://www.geonames.org/, “Geonames”, 2019. [Online]. Available: https://www.geonames.org/ [Accessed: 12- Nov- 2019].

[9] https://www.mapbox.com/, “Mapbox”, 2019. [Online]. Available: https://docs.mapbox.com/api/search/ [Accessed: 12- Nov- 2019].

[10] https://github.com, “Client Bluetooth Library”, 2019. [Online]. Available: https://github.com/OmarAflak/Bluetooth-Library [Accessed: 22- Oct- 2019].

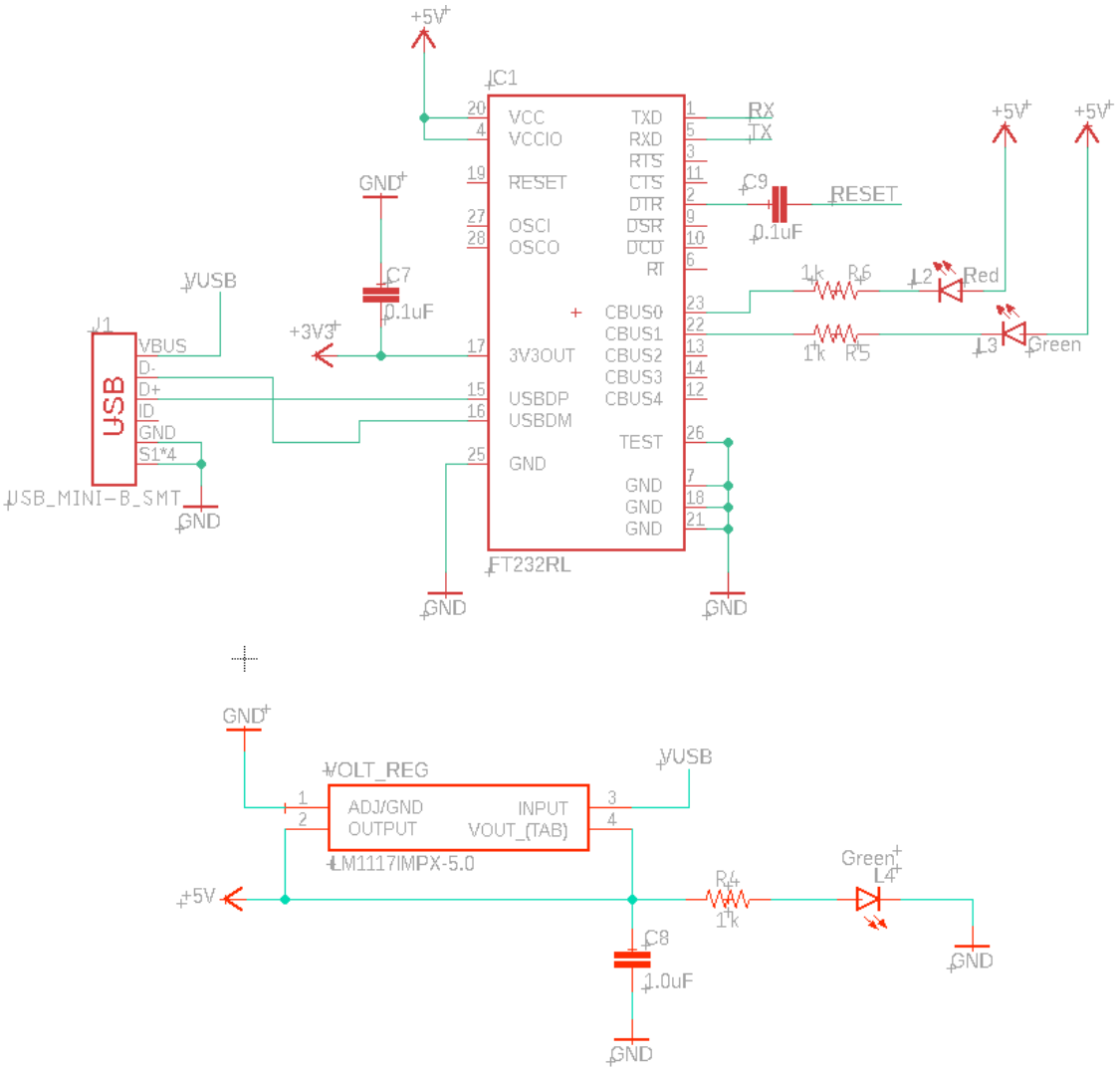
[11] https://developer.android.com, “Google Location Services”, 2019. [Online]. Available: https://developer.android.com/reference/android/location/LocationManager [Accessed: 12- Nov- 2019].

[12] https://github.com, “Volley”, 2019. [Online]. Available: https://github.com/google/volley [Accessed: 12- Nov- 2019].

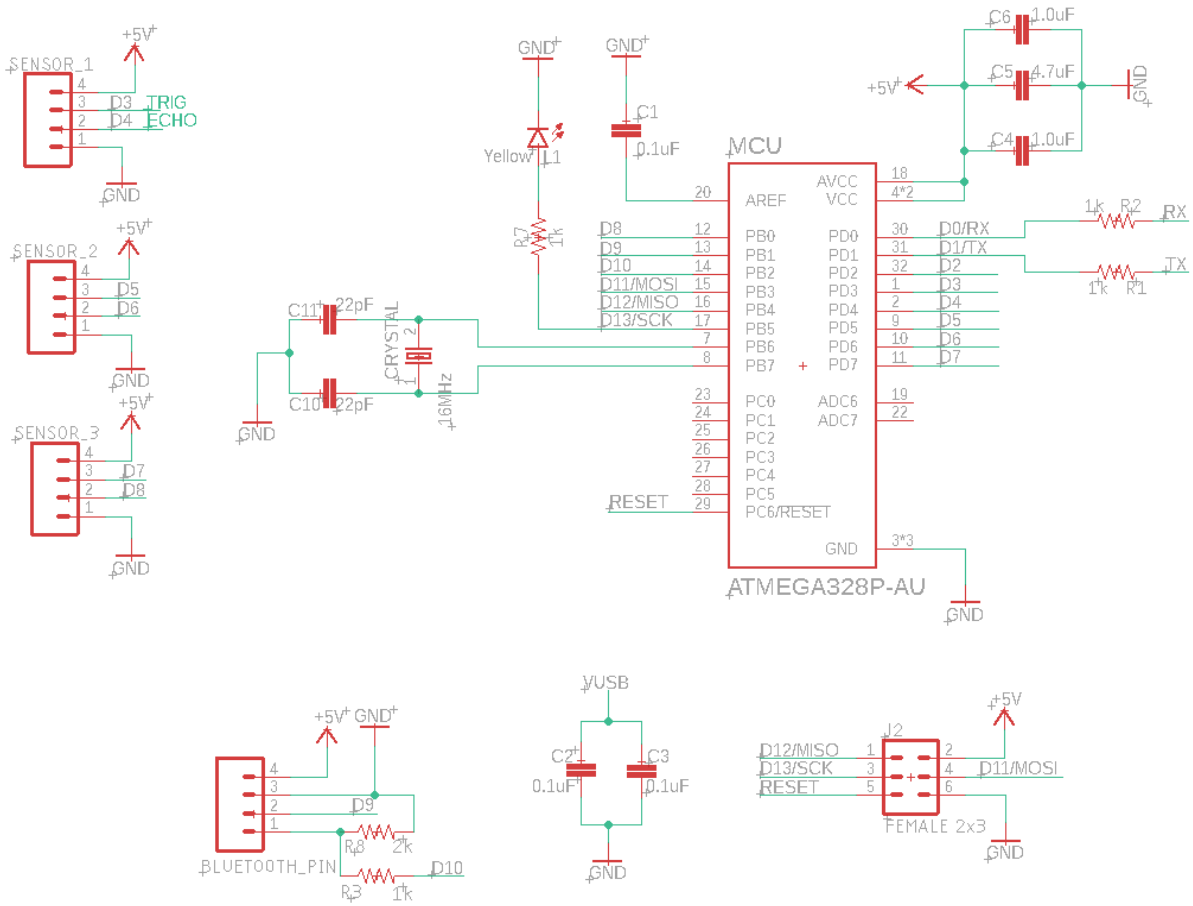
[13] https://www.cyberdriveillinois.com, 2019. [Online]. Available: https://www.cyberdriveillinois.com/publications/pdf_publications/vsd7001.pdf [Accessed: 06- Oct- 2019].

[14] Ieee.org, "IEEE IEEE Code of Ethics", 2016. [Online]. Available:http://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 19- Sep- 2019].

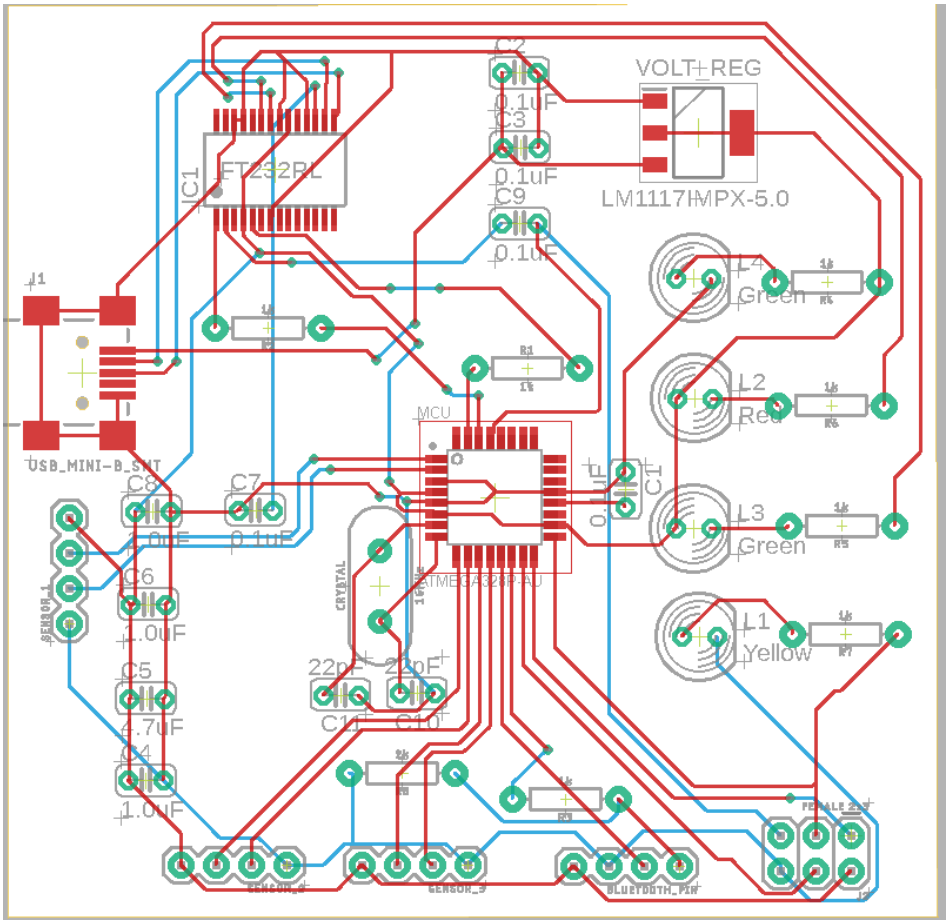
Appendix A Schematics of the PCB - 1



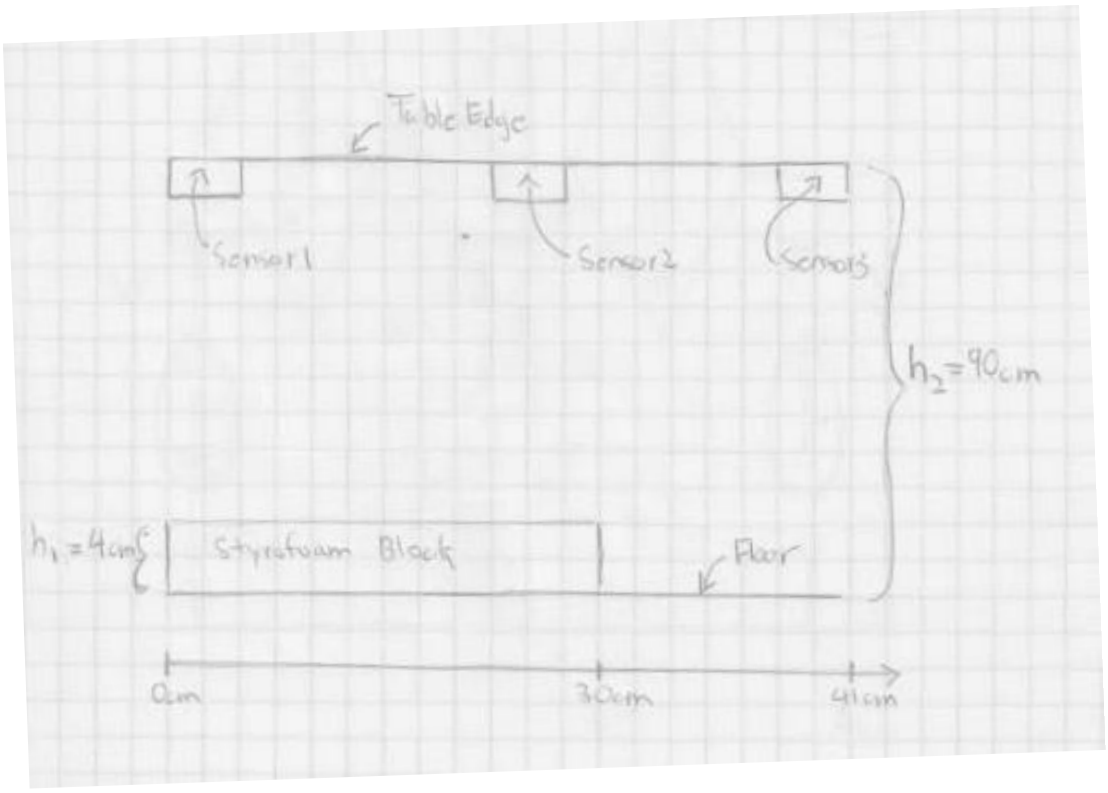
Appendix B Schematics of the PCB - 2



Appendix C Board of the PCB



Appendix D Ultrasonic Sensor Spread Behavior



Appendix E Smartphone Source Code

```
package com.example.rim_test_counter;

import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;

import android.Manifest;
import android.content.Context;
import android.content.pm.PackageManager;
import android.location.Criteria;
import android.os.Bundle;
import android.util.Log;
import android.util.Pair;
import android.widget.TextView;
import android.widget.Button;
import android.view.View;
import android.content.Intent;
import android.widget.AdapterView;
import android.location.Location;
import android.location.LocationManager;
import android.location.LocationListener;

import android.bluetooth.BluetoothDevice;
import me.aflak.bluetooth.Bluetooth;
import me.aflak.bluetooth.interfaces.BluetoothCallback;
import me.aflak.bluetooth.interfaces.DiscoveryCallback;
import me.aflak.bluetooth.interfaces.DeviceCallback;

import java.util.ArrayList;
import java.util.List;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
import com.android.volley.toolbox.Volley;
import com.google.gson.Gson;
import com.google.gson.JsonObject;
```

```
import com.google.gson.JsonParser;

import java.util.HashMap;

import org.json.*;
public class MainActivity extends AppCompatActivity {
	// Local Variables
	public TextView counter;
	public TextView streetText;
	public TextView counterStreet2;
	public TextView counterStreet;
	public Button counter_button;
	public Button scan_button;
	public Button reset_button;
	public Button green;
	public Button spring;
	public Button wright;
	public Button gps;
	public TextView coordinates;
	private Bluetooth bluetooth;
	private ArrayList<BluetoothDevice> scannedDevices;
	public boolean scanning = false;
	int tempCount;
	public ListView listView;
	HashMap<String, Pair<Integer,Integer>> streetCounts = new HashMap<>();

	Location currLoc;
	String currentStreet = "Green St.";
	String msg;
	ArrayAdapter adapter;
	String url = "www.google.com";

	// Instantiate the RequestQueue.
	RequestQueue queue;
	// Request a string response from the provided URL.
	StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
	new Response.Listener<String>() {
		@Override
		public void onResponse(String response) {
		// Display the first 500 characters of the response string.
		streetText.setText("Response is: "+ response.substring(0,500));
		}
	}, new Response.ErrorListener() {
```

```
@Override
public void onErrorResponse(VolleyError error) {
streetText.setText("That didn't work!");
}
});

@Override
protected void onCreate(Bundle savedInstanceState) {
queue = Volley.newRequestQueue(this);
streetCounts.put("Green St.", Pair.create(0,0));
streetCounts.put("Springfield", Pair.create(0,0));
streetCounts.put("Wright St.", Pair.create(0,0));
// Ask for permissions
int MY_PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION = 1;
ActivityCompat.requestPermissions(this,
        new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
        MY_PERMISSIONS_REQUEST_ACCESS_FINE_LOCATION);

int MY_PERMISSIONS_REQUEST_ACCESS_INTERNET = 1;
ActivityCompat.requestPermissions(this,
        new String[]{Manifest.permission.ACCESS_WIFI_STATE},
        MY_PERMISSIONS_REQUEST_ACCESS_INTERNET);

// Setup Bluetooth Library
setContentView(R.layout.activity_main);
bluetooth = new Bluetooth(this);
bluetooth.setBluetoothCallback(bluetoothCallback);
bluetooth.setDiscoveryCallback(discoveryCallback);
bluetooth.setDeviceCallback(devicecallBack);

// Setup list of devices
listView = findViewById(R.id.device_list);
adapter = new ArrayAdapter<String>(this,
        R.layout.activity_listview, new ArrayList<String>());
if (listView != null) {
listView.setAdapter(adapter);
listView.setOnItemClickListener(onScanListItemClick);
}
super.onCreate(savedInstanceState);
// Initialize counter button and textview
counter = findViewById(R.id.textView);
counterStreet2 = findViewById(R.id.counterStreet2);
counterStreet = findViewById(R.id.counterStreet);
```

```
        counter_button = findViewById(R.id.btn_count);
        scan_button = findViewById(R.id.scan_btn);
        counterStreet.setText(String.valueOf(0));
        // Set up counter button
        counter_button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
                tempCount = streetCounts.get(currentStreet).first;
                tempCount += 1;
                streetCounts.put(currentStreet, Pair.create(tempCount,
streetCounts.get(currentStreet).second));
                counterStreet.setText(String.valueOf(tempCount));
        }
        });

        // Setup Bluetooth Scan Button
        scan_button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
                bluetooth.startScanning();
                scanning = true;
        }
        });
        // Setup Reset Button
        reset_button = findViewById(R.id.resetBTN);
        reset_button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
                for (String name : streetCounts.keySet())
                streetCounts.put(name, Pair.create(0,0));

                counterStreet.setText(String.valueOf(0));
                counterStreet2.setText(String.valueOf(0));
        }
        });

        // Setup current street text
        streetText = findViewById(R.id.streetText);
        green = findViewById(R.id.greenSt);
        spring = findViewById(R.id.Springfield);
        wright = findViewById(R.id.wright);

        green.setOnClickListener(new View.OnClickListener() {
```

```
        @Override
        public void onClick(View view) {
                currentStreet = "Green St.";
                tempCount = streetCounts.get(currentStreet).first;
                counterStreet.setText(String.valueOf(tempCount));
                tempCount = streetCounts.get(currentStreet).second;
                counterStreet2.setText(String.valueOf(tempCount));
                streetText.setText(currentStreet);
        }
        });

        spring.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
                currentStreet = "Springfield";
                tempCount = streetCounts.get(currentStreet).first;
                counterStreet.setText(String.valueOf(tempCount));
                tempCount = streetCounts.get(currentStreet).second;
                counterStreet2.setText(String.valueOf(tempCount));
                streetText.setText(currentStreet);
        }
        });

        wright.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
                currentStreet = "Wright St.";
                tempCount = streetCounts.get(currentStreet).first;
                counterStreet.setText(String.valueOf(tempCount));
                tempCount = streetCounts.get(currentStreet).second;
                counterStreet2.setText(String.valueOf(tempCount));
                streetText.setText(currentStreet);
        }
        });



        gps = findViewById(R.id.gps);
        coordinates = findViewById(R.id.coordinates);

        gps.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
```

```
                currLoc = getLocationWithCheckNetworkAndGPS(view.getContext());
                String Lat, Long;
                Lat = String.valueOf(currLoc.getLatitude());
                Long = String.valueOf(currLoc.getLongitude());
                coordinates.setText(Lat+","+Long);

                url =
"https://nominatim.openstreetmap.org/reverse?&format=jsonv2&lat="+Lat+"&lon="+Long;
                StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
                new Response.Listener<String>() {
                        @Override
                        public void onResponse(String response) {
                        // Display the first 500 characters of the response string.
                        JsonObject jsonObject = new
JsonParser().parse(response).getAsJsonObject();
                        jsonObject = jsonObject.get("address").getAsJsonObject();
                        streetText.setText(""+ jsonObject.get("road"));
                        currentStreet = jsonObject.get("road").getAsString();
                        if (!streetCounts.containsKey(currentStreet)){
                        streetCounts.put(currentStreet, Pair.create(0,0));
                        }
                        tempCount = streetCounts.get(currentStreet).first;
                        counterStreet.setText(String.valueOf(tempCount));
                        tempCount = streetCounts.get(currentStreet).second;
                        counterStreet2.setText(String.valueOf(tempCount));
                        }
                }, new Response.ErrorListener() {
                @Override
                public void onErrorResponse(VolleyError error) {

                streetText.setText(error.getMessage());
                }
                });
                queue.add(stringRequest);
        }
        });


        }

        @Override
        protected void onStart() {
        super.onStart();
```

```
bluetooth.onStart();
if(bluetooth.isEnabled()){
String message = "Bluetooth Enabled";
counter.setText(message);
} else {
bluetooth.enable();
}
}

@Override
protected void onStop() {
super.onStop();
bluetooth.onStop();
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
super.onActivityResult(requestCode, resultCode, data);
bluetooth.onActivityResult(requestCode, resultCode);
}


private BluetoothCallback bluetoothCallback = new BluetoothCallback() {
@Override public void onBluetoothTurningOn() {}
@Override public void onBluetoothTurningOff() {}
@Override public void onBluetoothOff() {}

@Override
public void onBluetoothOn() {
// doStuffWhenBluetoothOn() ...
}

@Override
public void onUserDeniedActivation() {
// handle activation denial...
}
};

private DiscoveryCallback discoveryCallback = new DiscoveryCallback() {
public void onDiscoveryStarted() {
scannedDevices = new ArrayList<>();
}
@Override public void onDiscoveryFinished() {}
```

```
        @Override public void onDeviceFound(BluetoothDevice device) {
        String message = "Device Found";
        counter.setText(message);
        scannedDevices.add(device);
        adapter.add(device.getAddress()+" : "+device.getName());
        }
        @Override public void onDevicePaired(BluetoothDevice device) {}
        @Override public void onDeviceUnpaired(BluetoothDevice device) {}
        @Override public void onError(int errorCode) {

        }
        };

        private DeviceCallback devicecallBack = new DeviceCallback() {
        @Override public void onDeviceConnected(BluetoothDevice device) {
        String message = "Device Connected";
        }
        @Override public void onDeviceDisconnected(BluetoothDevice device, String
message) {}
        @Override public void onMessage(byte[] message) {
        msg = new String(message);
        Log.d("myTag", msg);
        if (msg.equals("<POT>")){
                tempCount = streetCounts.get(currentStreet).first;
                tempCount += 1;
                streetCounts.put(currentStreet, Pair.create(tempCount,
streetCounts.get(currentStreet).second));
        }
        else if (msg.equals("<DEB>")){
                tempCount = streetCounts.get(currentStreet).second;
                tempCount += 1;
                streetCounts.put(currentStreet,
Pair.create(streetCounts.get(currentStreet).first, tempCount));
        }
        runOnUiThread(new Runnable() {

                @Override
                public void run() {
                setStatus(msg);
                counterStreet.setText(String.valueOf(streetCounts.get(currentStreet).first));

        counterStreet2.setText(String.valueOf(streetCounts.get(currentStreet).second));
                }
```

```
        });
        }
        @Override public void onError(int errorCode) {}
        @Override public void onConnectError(BluetoothDevice device, String message) {}
        };

        private void setStatus(String message){
        counter.setText(message);
        }

        private AdapterView.OnItemClickListener onScanListItemClick = new
AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        if (scanning) {
                bluetooth.stopScanning();
        }
        // Pair
        bluetooth.pair(scannedDevices.get(i));
        String message = "Device Paired";
        setStatus(message);
        bluetooth.connectToDevice(scannedDevices.get(i));
        message = "Device Connected";
        setStatus(message);
        listView.setVisibility(View.GONE);
        }
        };


        // Location Get
        public static Location getLocationWithCheckNetworkAndGPS(Context mContext) {
        /*
        LocationManager lm = (LocationManager)
                mContext.getSystemService(Context.LOCATION_SERVICE);
        assert lm != null;
        boolean isGpsEnabled = lm.isProviderEnabled(LocationManager.GPS_PROVIDER);
        boolean isNetworkLocationEnabled =
lm.isProviderEnabled(LocationManager.NETWORK_PROVIDER);

        Location networkLocation = null, gpsLocation = null, finalLoc = null;
        if (isGpsEnabled)
        if (ActivityCompat.checkSelfPermission(mContext,
Manifest.permission.ACCESS_FINE_LOCATION) !=
```

```
PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(mContext,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {

                return null;
        }gpsLocation = lm.getLastKnownLocation(LocationManager.GPS_PROVIDER);
        if (isNetworkLocationEnabled)
        networkLocation =
lm.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);

        if (gpsLocation != null && networkLocation != null) {

        //smaller the number more accurate result will
        if (gpsLocation.getAccuracy() > networkLocation.getAccuracy())
                finalLoc = networkLocation;
        else
                finalLoc = gpsLocation;

        } else {

        if (gpsLocation != null) {
                finalLoc = gpsLocation;
        } else if (networkLocation != null) {
                finalLoc = networkLocation;
        }
        }
        */
        LocationManager lm =
(LocationManager)mContext.getSystemService(Context.LOCATION_SERVICE);
        assert lm != null;
        if (true)
        if (ActivityCompat.checkSelfPermission(mContext,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(mContext,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {

                return null;
        }
        Location loc = lm.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
        return loc;
```

```
        }
};
```

Appendix F Smartphone Module R&V table

Requirement	Verification	Verified?
1) The smartphone app should process signals and update the number of interferences on the current road within 4 seconds of receiving the signal. 2) The smartphone app should receive and recognize the Bluetooth signal from the MCU within 1 second. 3) The smartphone app should update the current road within 3 seconds of changing roads at speeds of up to 10 mph. 4) The GPS data must be able to tell the current road and nearby roads within 100m.	1) Create a test app that marks the location of an interference without using a Bluetooth signal. Measure the time from start to the updating of the interference count with a stopwatch. Verify the time is less than 4 seconds. 2) After implementing all of the parts of the main loop of the app, display which box in the flowchart we are currently in. Then, send a bluetooth signal from the microcontroller and measure how long it takes to go to the signal detected state. Verify the time is less than 1 second. 3) Create a test app that displays the current road. Drive around and change roads with the app open. Have a passenger measure the time it takes for the current road to change with a	1) Success 2) Success 3) Success 4) Success (Only current road detected)

	stopwatch and ensure the time is below 3 seconds. 4) Create app to tell the name of the current road. Verify the results are correct.	

Appendix G Reverse Geocoding Responses

Geonames


```
streetSegment:
  0:
      adminCode2:    "019"
      adminCode1:    "IL"
      distance:      "0"
    line:            "-88.217755 40.112804,-88.218925 40.112808,-88.219226 40.112813,-88.219337 40.11281"
      mtfcc:         "S1400"
      placename:     "Urbana"
      adminName2:    "Champaign"
      fraddl:        "801"
      postalcode:    "61801"
      countryCode:   "US"
      toaddr:        "898"
      name:          "Springfield Ave"
      adminName1:    "Illinois"
      toaddl:        "899"
      fraddr:        "800"
```


Google Nearest Road


```
snappedPoints:
  0:
    location:
        latitude:      40.112809132023465
        longitude:     -88.2186964485933
    originalIndex:     0
    placeId:           "ChIJ8QDGG23XDIgR1qhg1oH0XY0"
```

Mapbox


```
query:
    0:                  -88.2186964524825
    1:                  40.1128072390702
features:
  0:
      id:               "address.2039157604394352"
      type:             "Feature"
    place_type:
        0:              "address"
      relevance:        1
    properties:
        accuracy:       "rooftop"
      text:             "West Springfield Avenue"
    place_name:         "808 West Springfield Avenue, Urbana, Illinois 61801, United States"
```


Nominatim

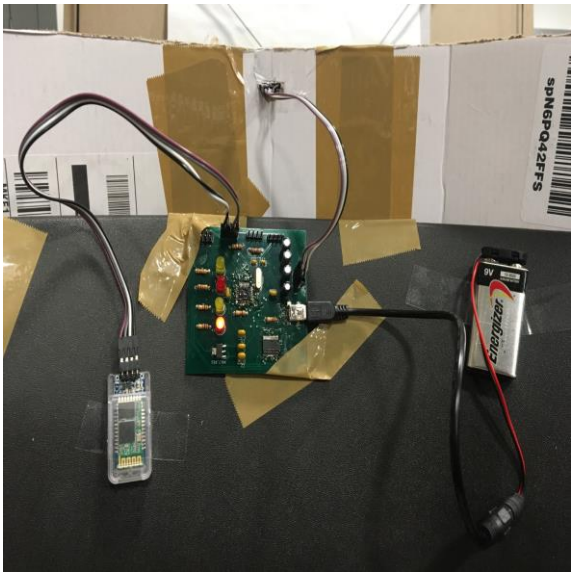

```
  place_id:         247064427
licence:            "Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright"
  osm_type:         "way"
  osm_id:           5339010
  lat:              "40.1128072390702"
  lon:              "-88.2186964524825"
  place_rank:       26
  category:         "highway"
  type:             "tertiary"
  importance:       0.1
  addresstype:      "road"
  name:             "West Springfield Avenue"
display_name:       "West Springfield Avenue, Downtown Urbana, Urbana, Champaign County, Illinois, 61801, United States of America"
```

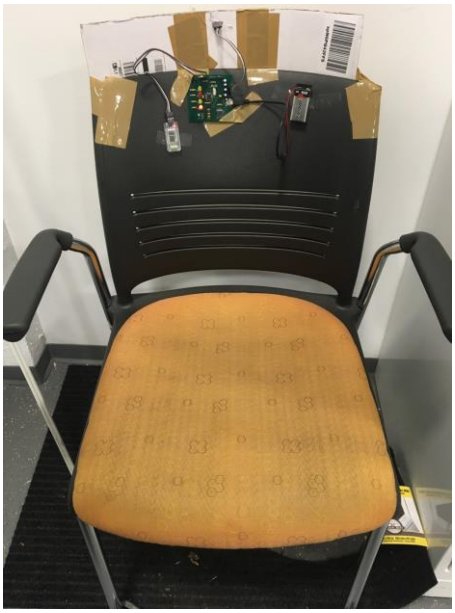
Appendix H Bluetooth Testing Source Code

```
#include <SoftwareSerial.h>
SoftwareSerial BTserial(2, 3); // RX | TX
// Connect the HC-06 TX to the Arduino RX on pin 2.
// Connect the HC-06 RX to the Arduino TX on pin 3 through a voltage divider.
bool start_cmd = false;
bool end_cmd = false;
char cmd[8];
int cmd_idx = 0;
void setup()
{
        Serial.begin(9600);
        Serial.println("Enter AT commands:");

        // HC-06 default serial speed is 9600
        BTserial.begin(9600);
}

void loop()
{

        // Keep reading from HC-06 and send to Arduino Serial Monitor
        if (BTserial.available())
        {
        type_command();
        if(end_cmd){
        parse_command();
        }

        }

        // Keep reading from Arduino Serial Monitor and send to HC-06
        if (Serial.available())
        {
        BTserial.write("<DET>\n");
        }

}

void type_command(){
  char current_rec;
```

```
  if (BTserial.available()){
          current_rec = BTserial.read();
          if (current_rec == '<'){
          start_cmd = true;
          }
          else if (current_rec == '>'){
          end_cmd = true;
          }

          if (start_cmd){
          cmd[cmd_idx] = current_rec;
          cmd_idx++;
          }
          else{
          Serial.write(current_rec);
          Serial.write('\n');
          }
  }
}

void parse_command(){
  if (strcmp(cmd, "<hello>") == 0){
          Serial.write("Goodbye\n");
  }
  else if (strcmp(cmd, "<green>") == 0){
          Serial.write("Blue\n");
  }
  else{
          Serial.write("Unknown Command\n");
  }
  start_cmd = false;
  end_cmd = false;
  cmd_idx = 0;
}
```

Appendix I Demonstration Environment





Appendix J Power Module R&V Table

Requirements	Verifications	Verified?
1) The input of the LDO voltage regulator must sink 9 V+/-67%. 2) The output of the LDO voltage regulator must source 5 V +/-1%.	1) Using a digital multimeter: a) Connect the red port of the dmm to the input pin of the voltage regulator. b) Connect the black port of the dmm to the ground pin of the voltage regulator. c) Check the voltage reading is 9 V +/- 67%. 2) Using a digital multimeter: a) Connect the red port of the dmm to the output pin of the voltage regulator. b) Connect the black port of the dmm to ground pin of the voltage regulator. c) Check the voltage reading is 5 V +/- 1%.	1) Success (The input voltage is 8V) 2) Success

Appendix K Ultrasonic Sensor R&V Table

Requirements	Verifications	Verified?
1) The distance between the ground and the underside of the vehicle the sensors are mounted underneath must be between 2 cm-400 cm. 2) The sensors should achieve a sampling rate of at least 17 Hz at a distance of 1m from the surface.	1) Measure distance to the ground from sensors with a meter stick or ruler to ensure values are within the specified 2 cm-400 cm range. 2) Verification for Item 2: a) Load test program on to an Arduino Uno to probe and read the data from a single ultrasonic sensor. b) Connect a laptop to monitor the sampling rate of the sensor. c) Connect one ultrasonic sensor to the Arduino Uno. d) Connect power to the Arduino Uno e) Observe the sample rate is at least 10 Hz using the test program with the surface 1m away.	1) Sensor didn't work below 40cm 2) After filtering noise out, the sampling rate was cut in half to 9Hz

Appendix L Processing Module R&V Table

Requirements	Verifications	Verified?
1) The MCU should receive distance data from the ultrasonic sensors. 2) The interference detection algorithm on the MCU should detect the number of interferences on a given road within +/- 2 interferences 3) The Bluetooth chip should receive all signals from the MCU and send them to the smartphone app within 2 seconds +/- 5%. 4) The Bluetooth chip should receive all signals from the smartphone app and send them to the MCU within 2 seconds +/- 5%.	1) Verification for Item 1: a) Connect each ultrasonic sensor to the MCU circuit. b) Connect power to MCU and ultrasonic sensors. c) Create a test program to send and receive the required signals to get distance data from the ultrasonic sensors. d) Ensure accurate distance data is being received with a ruler. 2) Verification for Item 3: a) Load interference detection algorithm onto the MCU. b) Connect each ultrasonic sensor to the MCU circuit. c) Connect power to MCU circuit and ultrasonic sensors. d) Connect a laptop to MCU	1) Success 2) Success 3) Success 4) Success

	circuit to monitor results. e) Move ultrasonic sensor array over a known number of interferences and count how many are actually detected. f) Ensure the number of actually detected interferences are within +/- 2 interferences of the true value. 3) Load a test program onto the arduino uno. Attach the Bluetooth chip to the correct pins on the Arduino Uno. Using a test smartphone app, attempt to send data from the arduino to the smartphone and measure the time it takes with a stopwatch. Ensure the time is within 2 seconds +/- 5%. 4) Load a test program onto the arduino uno. Attach the Bluetooth chip to the correct pins on the Arduino Uno. Using a test smartphone app, attempt to send data from the smartphone to the Arduino and measure the time it takes with a stopwatch. Ensure the	

	time is within 2 seconds +/- 5%.	