Pothole Tracker

Mike Catalano, EE, Daniel Chin, CSE, Bill Quigg, EE, and Muhammad Mir, CSE

***Abstract*—The Pothole Tracker is a system of devices that gathers and stores information about potholes encountered on the road. The first Raspberry Pi board, containing sensor units, is mounted on the front of the vehicle while the second Raspberry Pi board, containing the storage, image processing program, and button to activate the system, is located within the vehicle. The system is connected to the internet through a Wi-Fi dongle which sends information gathered by the sensors to a database. A website is connected to the database to display the data on Google Maps to users, such as the Department of Public Works, in a user-friendly manner.**

I. INTRODUCTION

POTHOLES are holes formed in the pavement as by excessive use or by extremes of weather. They have become a nuisance and hazard to drivers everywhere since the dawn of paved roads. These craters have caused serious damages to cars resulting in costly repairs. According to AAA, potholes have cost drivers $6.4 billion a year. This is around $2,000 throughout the life span of a car just from road conditions. Road conditions are costing drivers, potholes being at the forefront of the cause forcing drivers to demand their efficient repair.

The first step to solving a problem like repairing potholes is being able to know where a pothole is located. Not all potholes are big enough in size and depth to cause damage and certainly not every pothole can be repaired. How do we keep track of potholes and their level of needing repair?

Most places like cities have a database of potholes setup, storing their location, size, and status of repair. The city of Seattle, Washington has an implementation similar to this but a key factor is how this information is acquired and stored. The citizens of Seattle have to submit forms reporting potholes, which then are reviewed and put into a database. Although this method is useful because who better to ask than those who actually use the roads but submitting a form is a hassle and so is manually storing the information into a database. With the advancement of technology there are better ways to keep track of potholes.

The majority of today's adults drive on the roads making this problem one that affects the vast majority. On top of it they have to deal with the consequences of something they have no control over fixing besides submitting a request form. Potholes are timeless in the sense that for as long as there are paved roads, they will become damaged and potholes will form.

An efficient solution to this problem would be a system that can track potholes and submit the required data into a database. Regardless of the technology, the system will need to go out into the field and analyze the potholes first hand.

Therefore the device needs to be compact enough to fit into a vehicle and not be in the way of anything. In addition it needs to be able to be powered by the car itself. It has to transfer data so wireless and network capabilities are a must. There will also need to be driver input to tell the device when to start analyzing the pothole.

Specification	Value
Power	<5V
Height	<6in
Width	<6in
Length	<6in

II. DESIGN

A. Overview

The solution we came up was to use sensors and image processing to extract important information like size, depth, and location to store into a database. We decided to use Raspberry Pi boards for our system because they are compact powerful microcomputers. They are capable of a lot at an affordable price. There will be two boards that will be used in the system. One will be used for all the sensors like the accelerometer, sonar, GPS, and camera and the second will serve as the NAS and image processing board. The driver has to travel around 30 mph, which is a good speed for residential areas. When the driver sees a pothole coming up from at least 50ft away they press a button that activates all the sensors. The camera takes 20 pictures in a sequence and the board chooses one to be sent off to the NAS to be stored along with a GPS location, accelerometer reading, and sonar depth. When the NAS receives the pothole picture, the image processing runs and extracts the diameter. When the driver presses a second button the NAS sends the data to a database to be stored. A webpage will access the database and display the locations of potholes on a map.

Incorporating a button into a system requires additional hardware that ensures reliable switching. Switch bouncing is a

M. Catalano from Abington, Ma (mcatalan@umass.edu).
D. Chin from Stoughton, Ma (dhchin@umass.edu).
B. Quigg from Pembroke, Ma (wquigg@umass.edu).
M. Mir from Stoughton, Ma (mmir@umass.edu).

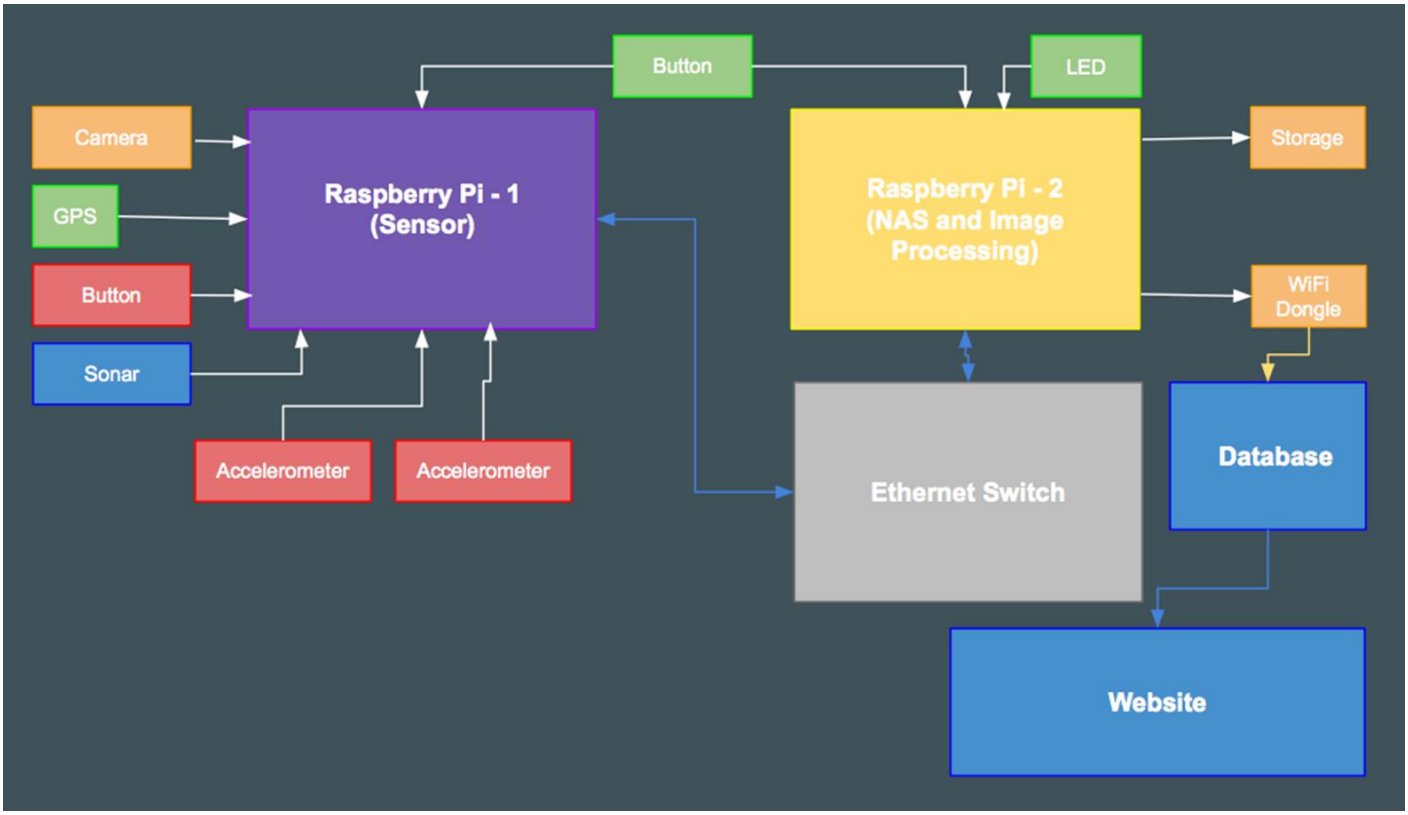

Figure 1: Block Diagram

problem that degrades the integrity of a mechanical switch. A switch bounce occurs when a mechanical switch is opened and closed. When this happens, unnoticeable vibrations on the contacts within the switch vibrate. This vibration causes the switch to open and close several times, typically within a period of a few hundred microseconds. An example of a switch bounce is shown in Figure 3, where the switch bounces twice before settling to a logical High. Switch debouncing in digital circuitry is very important when applications require precise timing. If a bounce were to occur, the Raspberry Pi would interpret the faulty input as multiple button pushes. There exists both software and hardware solutions to the switch bouncing issue. For the purposes of this project, hardware will be used. This decision was made based on the need for conserving processor resources for the image processing. Image processing will consume considerably more resources than switch debouncing, so eliminating the need for software is desirable whenever possible.

The circuit chosen to settle the switch bouncing is an RC debouncer that utilizes a Schmitt Trigger. The schematic for this circuit is shown in Figure 4. The circuit operates as following. When the button is not pressed, the capacitor charges through the series combination of R_1 and R_2 to a voltage close to V_{cc}. The voltage across the capacitor is the input voltage to the Schmitt Trigger inverter. When the capacitor is charged to V_{cc}, the output of the Schmitt Trigger

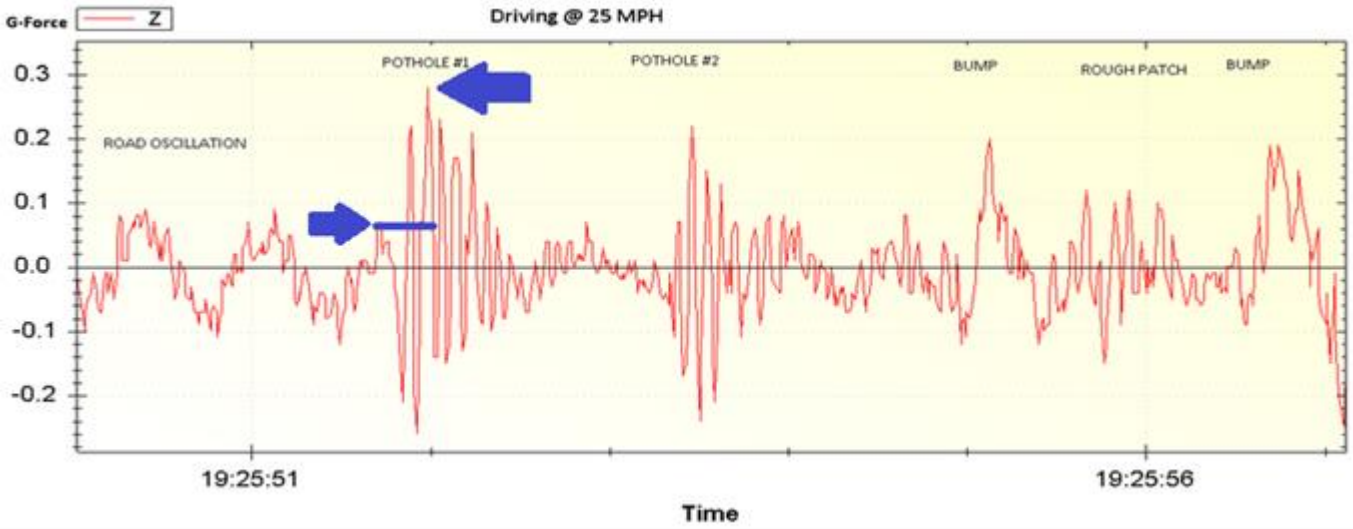

Figure 2: Accelerometer Reading

will output 0V, a logical Low. Alternatively, when the button is pressed after charging for some time, the capacitor will discharge through R_2. This causes the voltage across the capacitor to go to 0V. If the voltage across the capacitor is 0V, the Schmitt Trigger will output V_{cc}, a logical High.

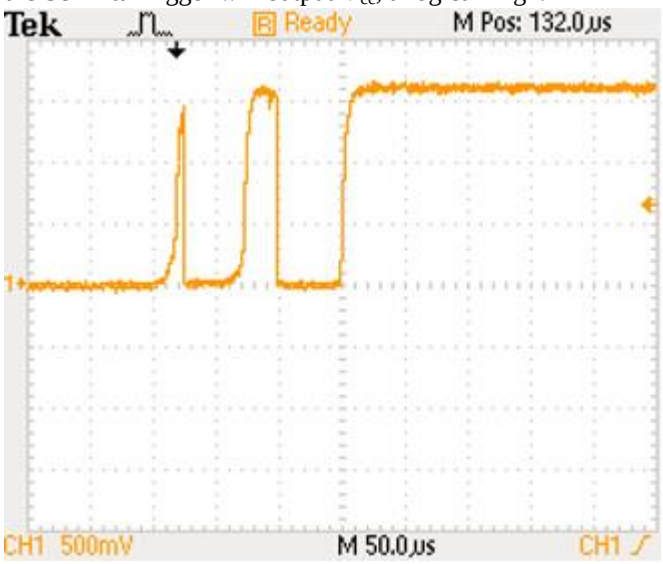

Figure 3: Before Debouncer

The design of this circuit focuses on maintaining a certain voltage long enough to settle any bouncing. The equation that dictates the voltage across a charging capacitor as a function of time is,

$$V_{cap} = V_{final}(1 - e^{-t/RC})$$

The previous equation can be rearranged in order to solve for R,

$$R = -t/C*\ln(1 - V_{cap}/V_{final})$$

Choosing a value of 0.1µF for the capacitor, C, is the first step in determining values for R_1 and R_2. The equation accounts for resistance in the exponent as R, which is equivalent to $R_1 + R_2$. V_{cap} is the voltage across the capacitor. The Schmitt Trigger will switch from a logical Low to High when the voltage across the capacitor goes above 0.8V. Therefore, $V_{cap} = 0.8V$. V_{final} is the voltage the capacitor is charging to, which is V_{cc}. For this project, $V_{cc} = 5V$. Finally, the time required to settle the bouncing comes from Figure 3, where the switch settles in approximately 175µs. This is the worst-case time recorded from numerous tests. Substituting these values into the previous equation yields $R = R_1 + R_2 = 10037\Omega$, which will be rounded to the closest standard resistance value of 10kΩ.

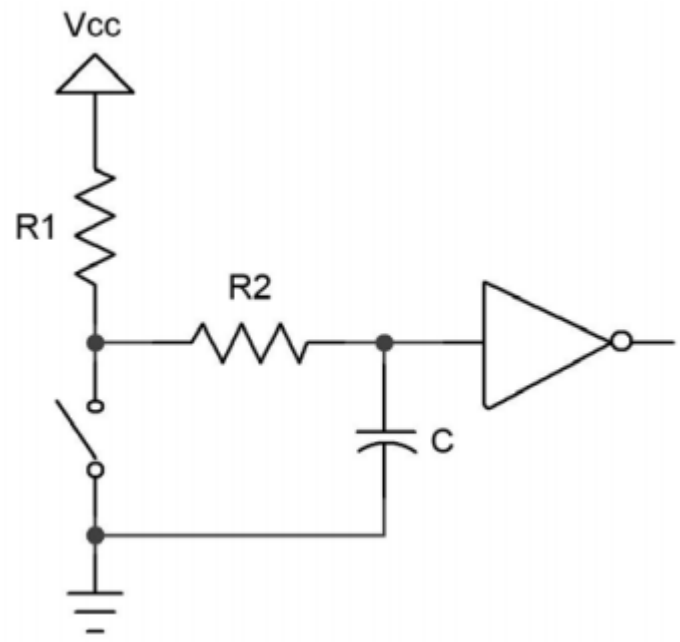

Figure 4: Debouncer Circuit

Another expression for R is needed in order to find values for R_1 and R_2. Consider the following equation for a capacitor discharging,

$$V_{cap} = V_{initial}(e^{-t/RC})$$

This equation can be rearranged as an expression for R,

$$R = -t/C*\ln(V_{cap}/V_{initial})$$

V_{cap} assumes a value of 1.6V, the voltage at which the Schmitt Trigger will switch from a logical High to Low. $V_{initial}$ is equivalent to V_{cc} (5V). The values for C and t used in the previous calculation remain the same in this calculation as well. Substituting these values into the previous equation for R yields $R = R_2 = 1536\Omega$. The value for R_2 will be rounded to 1.5kΩ.

There is now a system of two equations in order to solve for R_1 and R_2,

$$R1 + R2 = 10k\Omega$$
$$R2 = 1.5k\Omega$$

Solving for R1 yields a value of 8.5kΩ, which is rounded to 8.3kΩ. The final design involves the following components, R1 = 8.3kΩ, R2 = 1.5kΩ, C = 0.1µF, and an SN74LS14N Schmitt Trigger will be used. The following graphic shows the output of the switch debouncing circuit. Note how the voltage rises sharply without any bouncing. This is the desired effect.

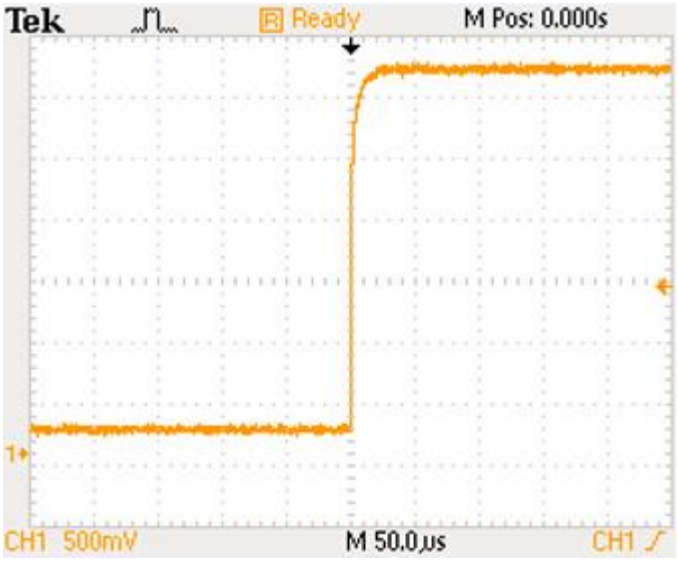

Figure 5: After Debouncer

There were many other technologies we considered aside from image processing. Lasers were a promising idea but obtaining one was expensive and out of an efficient budget. Lasers would have been able to perform a 3D reconstruction of the pothole, which would allow us to manipulate it and perform the necessary calculations. There was also an issue of generating multiple lasers or rotating one laser across the road.

The complications lead us to find a more simple method. We also considered using sonar over the use of a button input to allow the system to automatically detect when a pothole was approaching. Unfortunately the available sonars for Raspberry Pis had a limited range of fifteen feet restricting the amount of time to process and record from the cameras.

B. Block 1: Raspberry Pi 1

The first Raspberry Pi board will be equipped with a camera, GPS, accelerometer, and sonar. The system will be activated with the press of a button. In order to attain accurate results for the depth of potholes, accelerometers are placed on the front, left and right control arms of the test vehicle. This mounting location was chosen because the control arm is unaffected by the vehicle's suspension system. If the accelerometers were mounted elsewhere, the suspension system of the car would dampen the force experienced by the accelerometers, yielding an inaccurate result. The accelerometers utilized in our final design are ADXL335 analog accelerometers. Analog accelerometers were chosen over digital because the output of each accelerometer is filtered before being converted into a digital signal via an analog to digital converter(ADC). In order to determine how the accelerometer output should be filtered, we mounted accelerometers to the vehicle and recorded several tests where potholes were struck. These recordings were then analyzed by a MATLAB script that computed the Fast Fourier Transform(FFT) of the time-domain signal. The FFT of a pothole test is shown in Figure X. These results yielded spikes at approximately 15-18Hz. This indicates that the test vehicle oscillates at around 15Hz when a pothole is struck.

The next step in the design of the accelerometer block in the system was to construct filters to suppress frequencies above the 15-18HZ range. Two filters are needed, one filter for each accelerometer. We chose a third order Sallen-Key low pass filter topology to achieve a desired bandwidth at very low frequencies. The cutoff frequency of the filters was chosen to be approximately 20Hz. The third order configuration provides a 60dB/dec roll off, which very effectively rejects frequencies beyond the 15-18Hz range while maintaining the desired signals. In order for the Raspberry Pi to analyze the data yielded from the filters, a 10-bit , 8-channel ADC was chosen to convert the analog, filtered data. Only two channels were utilized, one channel for each accelerometer. The number of channels refers to the number of inputs the ADC can handle. With a resolution of 10 bits, the ADC outputs values from 0-1023, which corresponds to a voltage detected on the input of the ADC. For example, an ADC output of 0 means that 0V is applied to the input. An ADC output of 1023 occurs when 3.3V or higher is applied to the input. In order for the signal to rest in the middle (512) of the ADC's range, the output of the filters were DC biased at 1.65V. Biasing the signals at 1.65V allows for a maximized swing in accelerometer readings. If the signal were coupled to ground, only the positive components of the signal would be present. The final design was soldered to a prototype board in order to make the filter and ADC circuitry more robust. A printed circuit board (PCB) was designed in EAGLE to accommodate the circuitry. However, the PCB was too expensive for our budget, which is why the prototype board was utilized.

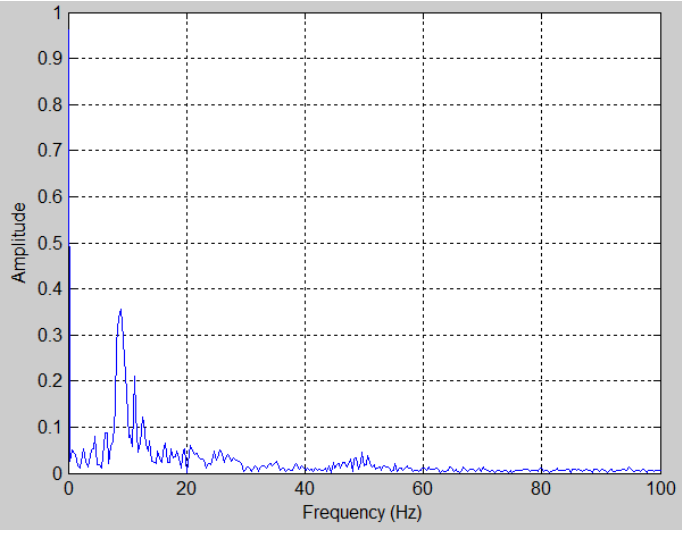

Figure 6: Frequency Domain Representation of Force from Pothole

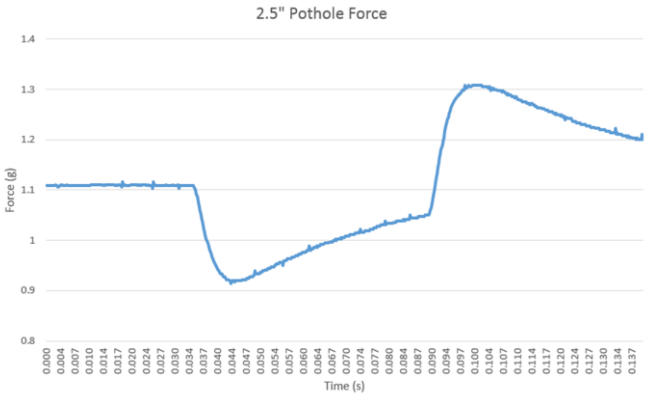

Figure 7: Time Domain Representation of Force from Pothole

For our GPS we chose to buy an Adafruit Ultimate GPS due to its low cost at $40 and low maximum error of about 10 meters. The GPS is connected to a USB port on the raspberry pi sensor board through a USB to TTL serial cable. As soon as the GPS receives power it outputs NMEA sentences. The National Marine Electronics Association created a protocol to define the format certain devices transmit data and our GPS follows this protocol. Before the GPS achieves a fix on the satellites the NMEA sentences it outputs will be incomplete, which means achieving a fix as fast as possible was crucial. To achieve a fix faster we attached an external antenna to our GPS, which was placed on top of the car. The antenna improved our time from powering on the GPS to getting a fix from about 5 minutes to around 30 seconds. Once the GPS has a fix we had to extract the appropriate data out of our NMEA sentences. We chose to find a $GPGGA sentence because they

contain the time, latitude and longitude. When the user presses the button to signify a pothole a python program will read the data coming into the USB port from the GPS, once it reads a $GPGGA sentence it records the sentence and writes it to a text file, then stops reading data from the USB port. From there the python program parses the $GPGGA text file for the time, latitude and longitude. The program then converts the latitude and longitude into a format that Google Maps can read. This is because with NMEA sentences the first 2 digits of the latitude (and 3 digits of the longitude) are in degrees formats while all the following digits are in minutes, so we divide the values reported in minutes by 60 and then add those to the respective values reported in degrees to give our latitude and longitude in degree format. Then we create a new GPS text file that contains the folder number, the time, the latitude and the longitude in a format that the NAS can grab and recognize.

Getting the depth of a pothole using an accelerometer required the car to drive into the pothole, which is not ideal unless the pothole is unavoidable. In most cases a driver will drive over a pothole. Originally, the depth was to be extracted through image processing but it proved to be difficult. Therefore a sonar was added in order to extract the depth from sound waves. The idea behind a sonar is that it sends sounds waves and when those waves hit a solid wall or object, the waves return. Using the time it takes for the waves to come back, you can calculate the distance to the object or wall the waves bounced off of. The distance to an object is calculated by multiplying the speed by the time. The speed of sound is 343 m/sec. We use half of the recorded time value because we want the to use the time it took the waves to reach the object, not the time it takes to bounce off the object and come back. The sonar runs for about 3 seconds, calculating distance values. When the car drives over the pothole, the distance will be greater than any other value because all other distances should be the distance to the ground. The largest distance is taken and the distance to the ground is subtracted from it to calculate the depth of the pothole. The depth is written to a text file and sent to the NAS.

All four sensors are synced in that they all run in parallel when the user presses the button. Once each has gathered its data and created its text file, it sends the text file off to the NAS.

C. Block 2: Network Attached Storage/Image Processing

The second Raspberry Pi board will be dedicated to become a Network Attached Storage for the first board and also serve as the board dedicated to processing the pothole images. The NAS will be connected to the other board through an Ethernet switch. The Ethernet connections will have the issue of dealing with a limited bandwidth while a high amount of data will be streamed from the first Raspberry Pi board, though not at the same time.

The NAS receives incoming files from the sensor board and stores those files into a folder. The files that are received contain data gathered by the accelerometer, sonar, and GPS. Each file contains the folder number they belong to which the NAS uses to store them in the appropriate folders. This process is used to keep data from different potholes separate. When the button to transmit is pressed, the each folder is opened and the contents inside each file are read. The contents are then sent to the database where they are stored. Following the data transmission, the data will be relocated and the state gets reset to a clean state.

A USB flash drive will be connected to the board so that when the user chooses to send all the data to the database, the raw files will be transferred to the flash drive into a folder with the date and time of the transfer as the folder name.

The data flow is relatively straightforward. The front Raspberry Pi board sends a picture and three text files to the second board. The data also is sent from the second board to the database.

Since the database is a separate system from the two Raspberry Pi boards, an internet connection is required to send data to it. To allow the board to access the internet through Wi-Fi, a USB Wi-Fi dongle will be connected in one of the USB ports. This allows the board to access a network connection without needing an Ethernet cable.

The second Raspberry Pi board also serves as the dedicated image-processing node. Image processing is a crucial step in the project because that is how we will determine the diameter of the pothole.

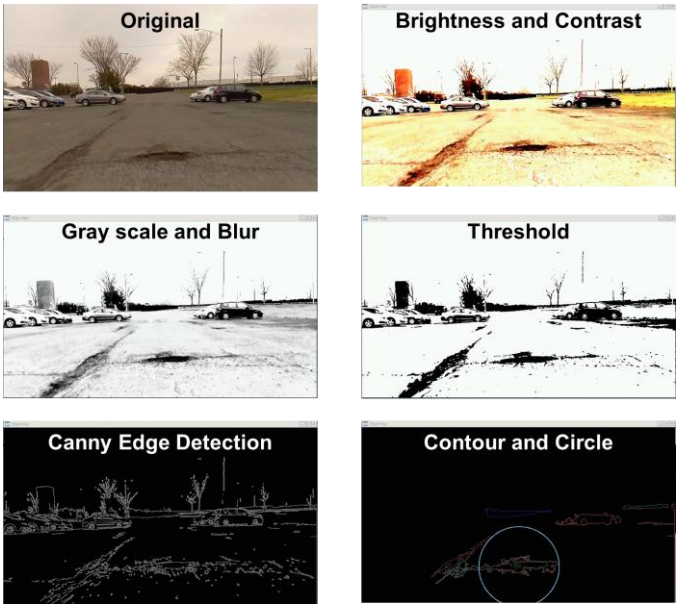

Figure 8: Image Processing Steps

The first step in the processing requires the image's brightness and contrast to be altered in such a way that the edges are more prominent. It then converts the image to gray scale and blurs it so that smaller less important edges are not as visible. Then a threshold value is calculated and the image is converted to black and white based off of the threshold. Now the algorithm runs Canny Edge detection and maps out the edges on the image. Once this step is finished the program draws out contours in the image and using these it draws out

circles matching a certain criteria. Depending on the location of the circle on the image, the algorithm calculates a diameter of the circle in inches. It takes into account whether the actual pothole is near the bottom, closer to the car, or near the top, further away. Depending on the proximity of the pothole, it will look bigger or smaller. That is why the algorithm uses the location of the pothole in the picture from its coordinates. The diameter is written to a text file and stored into its proper folder.

D.Block 3: Database and Webpage

The database stores the location, size, depth, and accelerometer reading of all potholes. The webpage grabs this data and plots the locations onto a map. This allows any user to now exactly where a pothole is and be able to prioritize which potholes to repair first.

Figure 9: Website Map Displaying Potholes

III. Project Management

We currently have one board that has the camera interface, accelerometer, and GPS working. The board is able to take pictures and store them as bitmap files. The other board is the image processing board. We have been able to process the image to outline the pothole but we are still working on improving the processed image to make it more ideal to extract the size and depth. Our progress for MDR was greatly affected due to having to go back to designing stage of our project. The parts were ordered later due to this fact and thus our ability to have each subsystem working was not possible with the given amount of time.

Each of our group members offers skills or expertise that will contribute to the project. Michael has experience doing group work involving both circuit design and computer programming. His experience with Java, C, and some Python will be useful when programming the Raspberry Pi. His circuitry skills will be invaluable when interfacing with the Raspberry Pi's hardware components. Various elements that will require hardware design in this project include the button that the driver activates when encountering a pothole and the accelerometers that will collect data regarding potholes. Michael is very interested in many disciplines of electrical engineering. His coursework focuses primarily on microwave engineering and signal processing. His knowledge of signal processing will be utilized when analyzing the data collected by the accelerometers. While his primary interests lie in electrical engineering, he also enjoys exploring interdisciplinary design problems. This curiosity will help when designing an enclosure to contain the finished project. Michael will also need to find a portion of a car frame to mount the accelerometers onto in order to collect pothole information.

William has experience building and verifying physical circuits in a group environment. He helped construct a switching voltage regulator using an efficient combination of several subsystems. This project improved his ability to verify that subsystems work together as planned. With regards to software William has learned multiple programming languages through coursework and his independent project of building an Android app. William will lead the synchronization of the GPS, camera and accelerometer subsystems. He will design an algorithm to assign the most appropriate GPS location to data collected from the cameras and accelerometers. His experience synchronizing the physical subsystems of a switching voltage regulator and familiarity with multiple programming languages will be valuable in the synchronization phase. William is also knowledgeable of networks and communications from engineering course work, he will provide assistance with the NAS and WIFI components if Daniel or Muhammad request it.

Daniel has experience in working in groups on both coding projects and database projects. He has experience in coding in C, C++, Java, and Visual Basic. Daniel was able to put his programming knowledge to the test during his internship at Verizon. Here he combined his programming knowledge with setting up and creating databases using both Visual Basic and MySQL. These skills will come in handy when programming the boards and working on image processing. Daniel also has experience in networking through courses he has taken in college. This skill will be help when handling data from all three Raspberry Pi's along with sending data to the database. Though he might need to learn Python, it shouldn't be hard has experience learning new languages.

Muhammad has extensive experience in web development and working with databases in MySQL. He is able to run efficient queries through the database to get desired content and also able to present the data onto a webpage. He has created many websites and added content to existing ones through his internship at the DOT. Muhammad also has experience working with MySQL databases with C/C++. These skills will come in handy when the data from the NAS will be sent to be stored into the database and then displayed onto a map on a webpage. Muhammad has had experience coding in C/C++ throughout his years in college, which will help with the overall software implementation of the Raspberry Pis.

There are two CSE's and two EE's in our group therefore each person has one other to ask for help in terms of their area. Daniel and Muhammad work together on the software end of the boards, the majority of which is the image processing. Mike and Bill help each other in implementing all the devices like accelerometer and GPS and wiring them up. We are able to help each other out too because most devices have a

software end to them and each of the boards have a hardware part.

Our team has been efficient with communication. With the convenience of text messaging we are able to communicate at all times. Whenever files or documents are involved, email and Google Docs are used.

IV. Conclusion

It has been a long journey and the project has evolved significantly. There were many last minute changes to the project especially with how the system determines the depth of the pothole. The addition of the sonar was added only a few weeks prior to FDR and thus was why it proved difficult to complete that portion of the system. The sonar accurately measured distance form it to a solid wall when in the lab but the physics of it changed when added to the car. There are many improvements that can be made but given the progression of our project as a whole we are all proud of work.

Specification	Goal	Achieved
Power	• All components should be powered by the car battery.	• The two Raspberry Pi boards, USB hub, and Ethernet switch are powered by separate car chargers.
Ease of Use	• One button for activating sensor units and another for sending data to the database. Indication of max capacity reached.	• The two buttons and the red LED are on the green breadboard which is located near the driver.
Efficiency	• System can gather data and process image in less than 30 seconds.	• Takes about 10 seconds from push of button to diameter text file.
Accuracy	• Diameter: + 5 inches • Accelerometer: + 1 inches • GPS: + 40 Yards • Sonar: + 3 inches	• Diameter is within range the majority of the time. • Accelerometer does not meet the range 40% of time. • GPS is always within range. • Sonar is almost never within accurate range of depth.
Storage	• Store a days worth of potholes on the Raspberry Pi.	• Set to store 100 potholes worth of information before sending to database is necessary.

Figure 10: Specification Table

Development

Part	Price
Raspberry Pi's	$77.40
Raspberry Pi Camera	$26.61
GPS + Antenna	$58.99
Accelerometers	$29.90
Sonar	$3.95
Resistors	$0.00
Capacitors	$0.00
WiFi Dongle	$9.99
Micro SD Cards	$19.98
Ethernet Switch	$9.95
Ethernet Cables	$16.98
USB to TTL Serial Cable	$9.95
Micro USB Cables	$14.98
5-Port USB Car Charger	$14.99
Power Adapter Car Charger	$16.95
Button Board	$0.00
Accelerometer Filter	$0.00
Total	$310.62

Production (1000)

Part	Price
Raspberry Pi's	$60.00
Raspberry Pi Camera	$20.00
GPS + Antenna	$48.68
Accelerometers	$23.92
Sonar	$1.41
Resistors	$0.17
Capacitors	$0.88
WiFi Dongle	$9.99
Micro SD Cards	$19.98
Ethernet Switch	$9.74
Ethernet Cables	$11.89
USB to TTL Serial Cable	$7.96
Micro USB Cables	$14.98
5-Port USB Car Charger	$14.99
Power Adapter Car Charger	$16.95
Button Board	$3.46
Accelerometer Filter	$10.35
Total	$275.35

Figure 11: Development and Production Cost Table

Acknowledgment

We would first off like to thank our faculty advisor Professor Maciej Ciesielski for making time in his busy schedule to help us through the difficult process of designing and implementing our project. We would also like to thank our faculty evaluators Professor Bill Leonard and Professor Sandip Kundu. Last but not least we would like to thank Professor C.V. Hollot for his patience and helpful advice and Fran Caron for his help in the lab. A special thank you, again, to Professor Leonard for lending us his monitors.

References

[1] A. detection, 'Accelerometer data smoothing filtering pothole detection', *Electronics.stackexchange.com*, 2015. [Online]. Available: http://electronics.stackexchange.com/questions/56238/accelerometer-data-smoothing-filtering-pothole-detection. [Accessed: 09- Dec- 2015].

[2] E. Buza, S. Omanovic and A. Huseinovic, 'Pothole Detection with Image Processing and Spectral Clustering', University of Sarajevo, 2005

[3] Seattle.gov, 'SDOT - Seattle Pothole Information', 2015. [Online]. Available: http://www.seattle.gov/transportation/potholes/. [Accessed: 09- Dec- 2015].

[4] WUSA9, 'AAA: pothole damage costs drivers $6.4 billion a year', 2015. [Online]. Available: http://www.wusa9.com/story/news/nation/2014/02/24/potholes-damage-cost-us/5773501/. [Accessed: 09- Dec- 2015].