VISION-BASED ROAD DEFECT DETECTION AND CLASSIFICATION FOR IMPACT EFFECT MITIGATION USING ADAPTIVE SUSPENSIONS

Shane M Haught

Follow this and additional works at: https://researchrepository.wvu.edu/etd

Recommended Citation
Haught, Shane M, "VISION-BASED ROAD DEFECT DETECTION AND CLASSIFICATION FOR IMPACT EFFECT MITIGATION USING ADAPTIVE SUSPENSIONS" (2018). Graduate Theses, Dissertations, and Problem Reports. 7185.
https://researchrepository.wvu.edu/etd/7185

This Dissertation is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Dissertation in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself. This Dissertation has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.
VISION-BASED ROAD DEFECT DETECTION AND CLASSIFICATION FOR IMPACT EFFECT MITIGATION USING ADAPTIVE SUSPENSIONS

Shane M. Haught

Thesis submitted
to the Benjamin M. Statler College of Engineering
at West Virginia University

in partial fulfillment of the requirements for the degree of

Master’s of Science in
Mechanical Engineering

Andrew Nix, Ph.D., Chair
Jason Gross, Ph.D.
Yu Gu, Ph.D.

Department of Mechanical and Aerospace Engineering

Morgantown, West Virginia
2018

Keywords: Computer Vision, Road Defect, pothole detection

Copyright 2018 Shane Haught
ABSTRACT

VISION-BASED ROAD DEFECT DETECTION AND CLASSIFICATION FOR IMPACT EFFECT MITIGATION USING ADAPTIVE SUSPENSIONS

Shane M. Haught

The contemporary automotive consumer demands vehicles with high performance, a strong safety record, and a comfortable ride. These desires are often compromised by defects in the road on which the vehicles are driving (e.g., potholes). Therefore, to meet high consumer standards, a novel method for the autonomous, proactive detection of road surface hazards, along with a strategy for adjusting vehicle suspension to improve performance, safety, and ride quality is presented. The approach uses 3D, infrared depth sensors and a state-of-the-art algorithm for point cloud processing to detect these hazards before they are encountered, giving the vehicle ample time to adjust the suspension before impact. This offers distinct advantages over the methods used in most vehicles on the market today. The underlying theory of the proposed road defect mitigation system, which is new and developed specifically for this application, is presented. The developed algorithm was implemented in MATLAB and tested against both simulated and field data. The algorithm was later converted from MATLAB to C++ in order to increase speed of the data capture and image processing. The field data was collected with a Microsoft Kinect on roadways in the vicinity of Morgantown, WV. Results from the field test demonstrate this approach is able to detect road hazards of arbitrary shape and size. Tests performed on the bench testing platform, which is designed and fabricated to simulate a tire impacting a series of road defects, suggest that it is feasible to alter the damping of an adaptive shock by varying a voltage being transmitted to the shock. The focus for this thesis is fourfold: (1) develop and present algorithms (2) migrate algorithms to a framework for processing in real-time, (3) classifying road defects into 6 discrete categories which will also include the mean and max depth of the road defect, and (4) bench testing and incorporation of suspension control logic and algorithms.
Dedication

This thesis is dedicated to my family. I want to thank my parents for supporting me on this journey. I know it has been a long time in the making, but we can finally see the light at the end of the tunnel and the conclusion to this chapter of my life. Finally, I want to thank my fiance, Rose Perroots, for all her love and support. I would not be here at this moment about to finish this academic pursuit without you pushing me to chase this goal and better myself. You have been there to lean on during all the hardships and have kept me grounded. I could not have done it without you, and I look forward to the next chapter in our story.
Acknowledgements

I would like to acknowledge a few individuals for the assistance and support provided to me during the pursuit of this degree. First I would like to thank my thesis committee members. Firstly, Dr. Andrew Nix, my committee chair and faculty advisor, for giving me the opportunity to join the West Virginia University EcoCAR 3 Team as the Project Manager and Innovation Team Lead which also presented the topic for this thesis and the platform to perform the necessary testing. You were always there to review my work and give me constructive feedback. Secondly, Dr. Jason Gross, for accepting me as one of your graduate students and taking over as my interim faculty advisor when I was left with none. You made it very comfortable for me to come and talk to you at any time about issues I was having. Thirdly, Dr Yu Gu, for agreeing to being one of my committee members. I know this thesis might not strongly relate to your area of expertise but you were still willing to take it on anyway.

I would also like to thank the Innovation Team which is part of the West Virginia University EcoCAR 3 team and, more specifically, three fellow team members. Andrew Rhodes was a member of the Innovation Team in Year 3 of the EcoCAR 3 competition and he contributed a great deal to the detection portion of the detection and classification algorithm. The development of the bench testing platform that was fabricated for testing purposes would not have been possible without the help from the EcoCAR 3 Controls Team Co-Lead, Nick Connelly, and the EcoCAR 3 Electrical Team Lead, Andrew Weers. Nick was instrumental in helping me develop the controls for the actuator which simulates a wheel impacting road defects and always willing to contribute his skills even at the risk of putting off his own work. Andrew graciously endured my endless questions regarding electrical systems and sensor setup.
Contents

Abstract .............................................. ii
Dedication ........................................... iii
Acknowledgements ................................ iv
Contents ............................................... v
List of Figures ...................................... vii
List of Tables ....................................... x
Nomenclature ....................................... xi

1 Introduction ...................................... 1
  1.1 Innovation Topic .................................................. 1

2 Literature Review .................................. 3

3 Technical Goals and Impact Metrics ......... 6

4 Road Defect Detection and Classification .... 8
  4.1 3D Data Capture Sensor - Microsoft Kinect Sensor .... 10
  4.2 Detection and Classification Algorithm Development .... 11
    4.2.1 Geometric Constraints .................................. 11
    4.2.2 Point Cloud Processing ................................ 14
      4.2.2.1 Determine Point Cloud Inliers and Outliers .... 14
      4.2.2.2 Road Defect Classification using Saliency Maps .. 16
  4.3 Real-Time Data Collection and Code Conversion .... 19

5 Adaptive Shock Experimentation .............. 20
  5.1 Bench Testing Platform ......................... 20
    5.1.1 Controls Development .................................. 25
    5.1.2 Sensor Selection ......................................... 28
      5.1.2.1 Laser Distance Measurement Sensor .............. 28
      5.1.2.2 Compression Button Load Cell ..................... 29
      5.1.2.3 Single-Axis Accelerometer ......................... 30
    5.1.3 Testing Adaptive Suspension on the Bench Testing Platform .... 31

6 Results ............................................. 36
  6.1 Real-Time Data Collection and Code Conversion .... 36
  6.2 Point Cloud Image Processing .................... 37
  6.3 Adaptive Suspension ....................................... 43
    6.3.1 Road Defect Simulation ................................. 43
List of Figures

1. Architecture of proposed road defect detection system .................................................. 2
2. Microsoft Kinect mounting bracket ..................................................................................... 9
3. Mounting system for the Microsoft Kinect ....................................................................... 9
4. Microsoft Kinect sensor ..................................................................................................... 10
5. Field of view of Kinect sensor ......................................................................................... 10
6. Geometry of road surface parameterization ..................................................................... 12
7. A noise free case of a road plane containing defects. Road defects shown in red demonstrate the importance of analyzing both normal vector alignment and perpendicular distance to the plane. This figure is best viewed in color. ................................................................. 15
8. Alignment-Distance Graph ................................................................................................. 15
9. 2D illustration of 3D points representing inliers or the planar road surface ....................... 16
10. 3D point cloud illustrating inlier points (shown in blue), outlier points above the road surface (shown in red), and outlier points below the road surface (shown in green) and representing the road defect ................................................................. 17
11. Shown above is: a) outlier points below the road surface before clustering, b) outlier points below the road surface after clustering, c) points from (b) overlaid onto inlier points (points within road surface) ....................................................................................................................... 17
12. Shown above is: a) actual color image of the road defect, b) blurred image from saliency map, c) smoothed image of road defect including the length and width indicated by the red lines ................................................................. 18
13. Kinect sensor field of view ............................................................................................... 20
14. Monroe OESpectrum C1501 adaptive shock .................................................................. 21
15. CAD drawing of proposed bench testing platform ............................................................ 22
16. Weights to represent 1/4 of a vehicle ................................................................................. 22
17. Double-acting pneumatic actuator .................................................................................. 23
18. Compressed air metering and control components ............................................................ 23
19. Image of fabricated bench testing platform with components labeled ............................. 25
20. The pneumatic system of the bench testing platform, including the control valve diagram .................................................................................................................................. 26
21. The electrical system of the bench testing platform illustrating the interaction of controller, sensors and data logger .......................................................................................... 27
22. Illustration of the linear relationship between the amount of time the solenoid is opened to the distance the piston travels .................................................................................. 27
23. Piston travel upward/downward using 2 different solenoid open times ......................... 28
24. Laser Distance Measurement Sensor ............................................................................. 29
25. Relationship between load cell signal and actual force (in kg) ...................................... 29
26. Compression button load cell ........................................................................................... 30
27. Single-axis accelerometer .................................................................................................. 31
28. Power supplies and switches ............................................................................................ 31
29. Proposed simulation of a series of uniform road defects .................................................. 32
30. Actual displacement attempting to follow the simulation of a series of uniform road defects .................................................................................................................................. 33
31. Indications of the maximum peak forces ......................................................................... 33
32. Test procedure to determine response time (Version 1) .................................................. 35
Test procedure to determine response time (Version 2)

Grant Ave. - Actual Color Image

Grant Ave. - Point Cloud

Results of Clustering

Grant Ave. - Outliers Overlaid with Inliers

Defining the Outline of the Road Defect

Actual measurements of road defects including: a) length, and b) width. Also shown in c), the dimensions generated by the algorithm including the classification assigned to the road defect.

Additional examples of road surface defects and algorithm post-processing

Illustration of actual displacement (purple) and force (yellow) using a uniform simulation

Illustration of actual displacement (purple) and force (yellow) using a uniform simulation

A set of four subfigures showing the force acting on the shock given an applied voltage of: (a) zero volts; (b) one volt; (c) 2.5 volts; and, (d) 5 volts.

Comparison of shock voltage to force

Expected uncertainty at various applied voltages

Top level control triggering the function

The real-time function broken into sensor and series of road defects (illustrates the feedback from the sensors)

Sensor functionality

Switch between rocker switch operation and series of road defect simulation

Triggers for the rocker switch

Rocker switch operation (up)

Rocker switch operation (down)

Simulation of a series of road defects

Operation amplifier combining a summing amplifier with an inverting amplifier

Gibbons St. - Actual Color Image

Gibbons St. - Point Cloud

Results of Clustering

Gibbons St. - Outliers Overlaid with Inliers

Defining the Outline of the Road Defect

Afton St. - Actual Color Image

Afton St. - Point Cloud

Results of Clustering

Afton St. - Outliers Overlaid with Inliers

Defining the Outline of the Road Defect

McLane Ave. - Actual Color Image

McLane Ave. - Point Cloud
List of Tables

1  Qualify Function Deployment (QFD) relationship matrix that maps customer benefits to technical metrics ................................................. 7
2  Comparison of Matlab algorithm execution speed with converted C++ algorithm execution speed .................................................................................. 37
3  Comparison of maximum lengths and widths of defects obtained from manual measurements and the proposed method ............................................ 42
4  Mean peak forces for varying discrete shock values .......................................................................................................................... 46
5  Calculations of the standard deviation and expected uncertainty ........................................................................................................ 48
**Nomenclature**

Bolded variables or symbols indicate vectors or matrices

\[ADG \quad \text{alignment distance graph}\]

\[D \quad \text{matrix of } \hat{x}, \hat{y} \text{ and } \hat{z} \text{ components for all points in a 3D point cloud}\]

\[d \quad \text{signed perpendicular distance}\]

\[depth \quad \text{depth value at pixel location } i\]

\[FOV \quad \text{field of view}\]

\[h_{\text{pixel}} \quad \text{individual pixel height}\]

\[I \quad \text{inliers from clustering of 3D point cloud}\]

\[k \quad \text{number of runs in a test}\]

\[n \quad \text{normal vector}\]

\[n \quad \text{number of all points in a 3D point cloud}\]

\[O \quad \text{outliers from clustering of 3D point cloud}\]

\[P \quad \text{source air (compressed)}\]

\[p_c \quad \text{3D point in the camera frame}\]

\[p_w \quad \text{3D point in the wheel frame}\]

\[R^w_c \quad \text{rotation matrix from the camera frame to the wheel frame}\]

\[r_w \quad \text{translation vector in the wheel frame}\]

\[t_a \quad \text{angular threshold}\]

\[t_d \quad \text{distance threshold}\]

\[u_{\text{expected}} \quad \text{expected uncertainties}\]

\[u_i \quad \text{elemental uncertainties}\]

\[u_{\text{manufacturer}} \quad \text{manufacturer uncertainties}\]

\[u_{\text{measurement}} \quad \text{measurement uncertainties}\]

\[U \quad \text{matrix whose columns contain orthonormal vectors}\]

\[V \quad \text{matrix with eigenvectors in the columns}\]

\[u_{\text{pixel}} \quad \text{individual pixel width}\]
\( x \)  component of a 3D point in the x direction
\( \bar{x} \)  mean of x components of all points in a 3D point cloud
\( \hat{x} \)  de-normalized x component of a 3D point
\( y \)  component of a 3D point in the y direction
\( \bar{y} \)  mean of y components of all points in a 3D point cloud
\( \hat{y} \)  de-normalized y component of a 3D point
\( z \)  component of a 3D point in the z direction
\( \bar{z} \)  mean of z components of all points in a 3D point cloud
\( \hat{z} \)  de-normalized z component of a 3D point

**Greek Symbols**

\( \alpha \)  angle of road pitch
\( \beta \)  angle of road bank
\( \delta \)  trigonometric components of the normal vector
\( \epsilon_i \)  signed perpendicular distance with respect to an inclined hill
\( \Sigma \)  diagonal matrix with eigenvalues from SVD
\( \sigma_s \)  standard deviation

**Subscripts**

\( a \)  angular
\( c \)  camera frame
\( d \)  distance
\( i \)  pixel location i
\( w \)  wheel frame
\( x \)  in the direction of the x component
\( y \)  in the direction of the y component
\( z \)  in the direction of the z component

**Units of Measure**

\( Hz \)  hertz
\( in \)  inches
\( kg \)  kilograms
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m/s</td>
<td>meters per second</td>
</tr>
<tr>
<td>mA</td>
<td>milliamperes</td>
</tr>
<tr>
<td>mm</td>
<td>millimeters</td>
</tr>
<tr>
<td>mph</td>
<td>miles per hour</td>
</tr>
<tr>
<td>ms</td>
<td>milliseconds</td>
</tr>
<tr>
<td>mV</td>
<td>millivolts</td>
</tr>
<tr>
<td>psi</td>
<td>pounds per square inch</td>
</tr>
<tr>
<td>sec</td>
<td>seconds</td>
</tr>
<tr>
<td>V</td>
<td>volts</td>
</tr>
</tbody>
</table>
1 Introduction

The modern road surface is susceptible to developing defects due to material and weather erosion, construction, heavy vehicular traffic, or infrastructure access covers. Encountering these road defects adversely affects the vehicle components, ride quality, and driver safety [1]. According to The Road Information Program (TRIP), a national transportation research group, in West Virginia alone deficient roads caused by these road surface defects costs motorists a total of $1.4 billion annually in vehicle operating costs (VOC), congestion-related delays and traffic collisions [2]. Operating costs can come in the form of repairs to damaged vehicle components such as suspensions, wheels, tires, etc. or even premature depreciation of the vehicle. Increased traffic collisions can result from these road defects when motorists opt to swerve around the road defects to avoid striking them which causes the vehicle to depart from their driving lane and into opposing traffic. The road defects can also cause the motorist to lose control of the vehicle if they strike the road defect and sustain a tire blowout [2]. Therefore, when approaching a road defect, the driver has at three primary options:

1. **Avoid the defect**: The driver may steer around the defect if space permits. This action, however, may cause the driver to unsafely cross driving lanes into oncoming traffic or designated pedestrian areas.

2. **Abruptly change the vehicle speed**: The driver may apply the brakes to suddenly slow down before encountering the defect. While this limits damage to vehicle components, it creates an uncomfortable ride and causes unsafe driving conditions for the surrounding vehicles.

3. **Drive over the defect without speed modification**: When the driver does not see the defect, have space and time to avoid the defect, or desires to maintain safe driving conditions, the smooth ride is compromised and vehicle components may incur damage.

While the first two options may cause unsafe driving conditions, the third option presents an opportunity for innovation. Mitigation of the adverse affects of driving over a road defect may be accomplished by using computer vision algorithms to preemptively identify road defects, followed by adjustment of an adaptive suspension to more efficiently absorb the impact. This innovative topic meets the desires of the local target market around Morgantown, WV where road conditions are poor and dangerous. This technology could also translate to other markets where high traffic areas and adverse weather conditions cause road defects in highways.

1.1 Innovation Topic

The proposed method for mitigating road defect impacts uses a 3D infrared depth sensor and state-of-the-art algorithms for point cloud processing to detect these hazards before they are encountered. After detection of the road defect, the defect is then classified by length, width and depth and is assigned a discrete classification number from 0 - 5. Based on the classification assigned, a signal is sent to an adaptive shock increasing or decreasing the damping of the shock in order to reduce the severity of the impact with the road defect. With this preemptive approach to road defect impact mitigation, the vehicle has ample time to adjust the suspension before impact. This novel concept adds three components to the vehicle: (1) a sensor mounted inside the grill of the vehicle, (2) efficient computer vision algorithms added to an advanced driver assistance systems (ADAS) computer package, and (3) adaptive shock systems on each of the wheels. Figure 1a shows
The proposed placement of these components on a test vehicle.

Real-time, autonomous detection of road surface defects and adjustment of the vehicle suspension system leads to an improved customer experience and extended lifespan of vehicle components. The proposed detection and classification of road surface defects (e.g., potholes, manhole covers, etc.) is accomplished using efficient computer vision techniques on 3D point clouds (development of a road hazard detection system is consistent with the functionality of ADAS). The addition of road hazards to the current list of recognizable objects (e.g., vehicles, pedestrians, road signs, etc.) by ADAS equipment is an improvement to the system as a whole. This contribution requires a sensor mounted on the front of the vehicle that returns 3D point clouds and an efficient computer algorithms to be included in the ADAS computer package.

Benefits to the industry and consumer are realized by proactive adjustment of the suspension system for the period in which the vehicle travels over the defect. Conventional suspension systems on vehicles are not adjustable; however, an expanded interest in ride quality has lead to industry adoption of active and adaptive suspension systems. Varying the shock absorber's viscous damping coefficient in an adaptive suspension leads to numerous advantages for the consumer and industry such as a smoother ride, increased vehicle component lifespan, and improved driver safety. In theory, the adaptive suspension would replace the conventional system on all four wheels as shown in Figure 1b.

The goals of this thesis are to increase the driver acceptability and ride quality, improve vehicle driving safety, and to extend the lifespan of vehicle components by reducing the impact sustained by the tires and suspension components. Successfully achieving these goals requires the development of computer vision algorithms that operate quickly and reliably, a 3D sensor that operates in various lighting and weather conditions, and the installation of an adaptive suspension system. The primary focus of this thesis is in development with respect to proof of concept and implementation of the computer vision algorithms used in conjunction with adaptive suspension systems. This thesis will also focus on testing and control of the adaptive suspension, and increasing the speed of the computer vision algorithms.
2 Literature Review

The objective of this proposed innovation is to design a 3D vision-based system capable of identifying and predicting road surface defects in real-time driving scenarios. Vision-based systems identify road surface defects ahead of the vehicle and before they are encountered by the wheels, in contrast to current systems that use accelerometers to detect road defects as they are encountered. The ability to predict road surface defect impacts provides the information necessary to intelligently adjust the vehicle suspension. Such suspension adjustment plays a critical role in extending the operating life of many vehicle components, including tires, wheels, and axles. Suspension adjustment also improves ride quality and comfort by reducing the impulse felt on the driver when encountering a road defect. The proposed vision-based method will operate in real-time, whereas existing vision-based methods of road surface defect detection, typically for road surveying, do not operate in real-time. This system represents a significant advancement in the state-of-the-art, and would result in a smoother ride and reduced wear on vehicle components.

Modifying the suspension of a vehicle may be accomplished using an active or adaptive technique. Active suspensions use a type of actuator (i.e., hydraulic, electronic, ...etc.) to physically raise and lower the chassis with respect to the suspension to reduce the impact on the driver when a tire encounters a road defect. While active suspensions keep the tires perpendicular to the road in corners, allowing for better traction and control, they are also more costly compared to adaptive suspensions and add additional mass to the vehicle along with complications due to the addition of multiple actuators. Because of these reasons, only the adaptive suspension is considered in this thesis. Adaptive suspension systems adjust the stiffness of the damper to promote a smoother ride when faced with adverse road conditions. The most common type of dampers use hydraulic oil, but other types include pneumatic or magneto-rheological (MR) fluid developed by the BWI Group and commonly referred to as MagneRide [3]. MR fluid is a state-of-the-art fluid that contains micrometer-sized magnetic particles that respond nearly instantaneously (less than 5 milliseconds) within an applied magnetic field. The magnetic particles align in such a way as to effectively change the properties of the fluid such as viscosity, essentially altering the damping of the adaptive shock. Bose uses a linear electromagnetic motor to create motion between the wheel and car body which could counteract the vehicle pitch and roll when encountering a defect [4]. The hydraulic and pneumatic systems have the longest response times while the electromagnetic systems respond much quicker. Any one of these adaptive suspension techniques could be used in this study, however, due to accessibility, this thesis will focus on a hydraulic damper.

The available commercial method for road defect impact mitigation is to react to the instantaneous acceleration of a wheel falling into a road defect sensed by accelerometers attached to the wheels. A patent by Bridgers and Chiang [6] uses an accelerometer and a 5-megapixel camera with a frame rate of at least 720p/60 high definition HD video capture in this manner to identify when a tire has encountered a road defect. They use measurements from the accelerometer that quantify the size of the defect and 2D images from the camera to create a database of road defects that can track defect changes through time by time stamping the images and associating geo-coordinate data with the images. The patent application is designed for road surveying by maintenance crews, not real-time modification of suspension for ride quality. Yu et al. [7] utilize an inertia system comprised of an Integrated Circuit Piezoelectric (ICP) accelerometer to evaluate highway pavement conditions for the purpose of surveying road conditions to determine where
road maintenance should be focused. The ICP accelerometer was chosen because it is not as susceptible to outside noise such as vibrations or tapping of wires. Engineers at Ford Motor Company have introduced a system equipped with a road defect mitigation system that utilizes an adaptive suspension and a collection of 12 high-resolution sensors capable of adjusting the damping of the shocks every 2 milliseconds [8]. An article announcing the technology from the Ford Motor Company does not definitively state that the sensors being used are accelerometers, but an engineer is quoted saying, “Our new pothole mitigation technology works by actually detecting potholes and ‘catching’ the car’s wheel before it has a chance to drop all the way into the pothole”. This indicates a reactive system similar to vibration-based systems using accelerometers. Their suite of sensors detect when the tire contacts the leading edge of a road defect and, very rapidly, stiffens the suspension before the tire has had time to fall all the way to the bottom of the road defect. When the tire contacts the trailing edge of the road defect, the tire is not as deep into the road defect and therefore does not impact the defect as roughly. The information from the front sensors located at the front tires also are transmitted to the shocks located at the rear tires which have more time to respond to the impact [8]. Jaguar Land Rover announced that they too have developed a system capable of detecting and mitigating the impacts of road defects. They have implemented the MagneRide suspension system on the Range Rover Evoque along with sophisticated sensors that profile the road surface underneath the tires and monitor motion of the vehicle and changes in the height of the suspension. Information from these sensors is sent to the MagneRide suspension system to modify the damping characteristics. The technology developed by Jaguar Land Rover also include the ability to transmit the road surface data and location to connected vehicles using the Cloud. While the article claims the system could also scan the road ahead there is no information on how this might be accomplished using the current configuration of sensors [9]. A vibration-based system boasts advantages such as relatively low cost, ease of packaging into a vehicle due to their small size, and the requirement of significantly smaller data storage devices compared to that needed of vision-based systems. However, inertia type methods using accelerometers may have only a marginal effect on ride comfort and safety as they wait until the moment of road defect impact to function limiting the amount of time in which a suspension system has to adjust.

One detection method is to use a modulated Doppler microwave sensor to infer the road surface conditions [5]. The Doppler method has the potential to miss road surface hazards as it emits only a singular microwave beam, and disregards large portions of the road. This proposed approach even suggests using additional sensors, aligned with the front wheels, in conjunction with the radar beam to determine the condition of the road surface under each wheel incorporating more information into the systems for analysis. Tribe et al. [5] suggest that the results of this Doppler-based systems could be used to alter vehicle components controlling the dynamics of the vehicle such as steering, suspension and braking management. Although, there is little information detailing how the incorporation of the system’s results could be accomplished.

A vision-based detection and prediction system yields time for the vehicle to modify the suspension before encountering the road defect, which will contribute to a more comfortable and safer ride, while extending the durability of vehicle components and systems. Most existing vision-based road defect detection systems operate in 2D and focus on road quality surveying for highway repair and maintenance crews. Images of roads with potential road defects are converted to gray scale, and then thresholding techniques are applied to cluster pixels as existing either in the foreground or background. The two popular types of intensity thresholding are the Otsu method [10][11][12], which minimizes the variance of intensities of the two clusters, or
the triangle method [13][14] which finds the maximum offset between the histogram and a line connecting the maximum and minimum histogram values. These methods apply arbitrary threshold values that may be affected by lighting conditions or road surface reflectance. Other arbitrarily chosen criteria used in these studies, such as requiring the entire defect be visible in the image, limits the effectiveness of these methods. A more robust method of road defect detection would be to analyze a 3D point cloud returned from a 3D sensor, such as the method under current investigation. All of the current vision based techniques are for pavement assessment processes used by survey and maintenance crews of the Long-Term Pavement Performance (LTPP) program established by the U.S. Department of Transportation, but none operate in real-time [13]. These technologies exist merely to log information into databases for future usage and not to modify vehicle systems.

Using a Microsoft Kinect sensor as a means for detecting road defects is a method that has been researched by a number of research groups, although none have suggested utilizing the sensor in a real-time environment to alter a vehicle’s suspension. The Kinect sensor is suited for image acquisition as it contains an infrared (IR) camera which provides depth images as well as a RGB camera which provides color images. One of the most thorough discussions on detection and characterization of road defects in 3D images is presented by Jahanshahi et. al. [10], where the focus is primarily on road surveying. This study analyzes the RGB and depth images returned by a Microsoft Kinect to quantify road defects. First, a plane is fitted to the 3D data using random sampling consensus (RANSAC) [15]. The depth values are then subtracted from the fitted plane and negative values are set to zero (where down into the defect is in the positive direction). This eliminates the ability of the system to identify road defects that arise up from road surface such as speed bumps or manhole covers. A grayscale image is then created by normalizing all relative depth values by the maximum depth. A histogram of the grayscale image undergoes Otsu thresholding to separate the image into two distinct regions — defect and non-defect. Finally, road defects are quantified by their depth and width and attached with a GPS coordinate for inclusion in the road defect tracking database. This method does not perform in real-time and there are limits on the maximum speed of the vehicle when collecting data. The Kinect operates in various lighting conditions (day or night), but it does not function in direct sunlight. A study by Kamal et. al. [16] determined that the depth image returned by the Kinect is negatively affected when the defect is partially or completely full of water. Therefore, use of the Microsoft Kinect to detect road defects needs further analysis before it can reliably be implemented. An alternative application of the Kinect sensor is provided by Moazzam et al. [17] in which he calculates volumes of road defects from the depth images acquired by the IR camera. Again, this experimentation is not for use in real-time environments as Moazzam statically affixes the Kinect sensor to a tripod and situates it directly above the road defect. Joubert et al. [18] are using the Kinect sensor along with a high-speed camera to acquire 3D point clouds of road defects and determine characteristics of these road defects such as length and width. Unlike most researchers, Joubert has attached the sensor system to a test vehicle and attempted to acquire images of road defects from a moving vehicle. Their goal was to accomplish the test on a vehicle traveling at a maximum speed of 60 km/h. While they achieved this goal, it is unclear if the processing of the data to detect the road defect was completed in real-time or the image was collected and then post-processed at a later time. Also, this proposed system does not include modification of vehicle components as a way to improve ride quality and overall comfort for the driver.
One of the most computationally expensive steps of road defect detection in 3D images is that of estimating the road surface plane. Many methods found in literature are designed for grouping points from a range image into planar clusters. These methods include using the Hough transform for multi-resolution analysis [19], a region growing algorithm refined by a polygonization step [20], measuring the congruency of surface normals and geometric distance from one another [21], or by applying normalized cuts to the range data and fitting a least squares plane [22]. The objective of all of these studies is to separate the range image into planar clusters, but not to fully characterize the equation of a plane. The objective of this thesis is to estimate the equation of the plane so as to find which range measurements constitute the road surface or the road defects(s). The plane estimation method presented by Zhou et. al. [23] estimates the equation of a plane using spherical coordinates instead of Cartesian coordinates. To find the azimuth and elevation angles, they derive an analytic expression using the 3D gradients of the range image. At every point in the point cloud, the gradients are found by applying separable horizontal and vertical filters to the 3D data. Then the surface normal at the point is defined by analytic equations containing the gradients. Points with similar surface normal vectors are clustered together using the iso-cluster unsupervised method [24], which works equally well with horizontal and vertical planes. Unfortunately, taking gradients of 3D range data is complicated and no one method is agreed upon within the scientific community. Instead, the error between the estimated plane and the 3D data is minimized by beginning from a maximum likelihood standpoint. In this way, the plane estimation is more numerically stable and robust because no gradient derivatives of the range image are taken.

The proposed method detects road defects in a sensed 3D point cloud with the intent to increase or decrease the damping of the suspension before the defect is struck. A 3D point cloud returned from a depth image sensor is analyzed to describe the nominal road level using known geometric constraints between the camera and vehicle. By characterizing the road surface, points that do not lie on the plane and thus may constitute a defect or other obstacle can be determined. Prediction of road defects in the vehicle’s path in real-time permits pre-impact manipulation of the suspension system to create a more comfortable ride and reduce damage to car components. This 3D vision-based defect detection system represents a significant advancement in the state-of-the-art method of road defect identification.

3 Technical Goals and Impact Metrics

The customer and industry benefits produced by the proposed innovation are a more comfortable ride, less wear and tear on vehicle components, and improved vehicle driving safety [13]. The successful adoption of this innovation requires that the solution be reliable, non-invasive to regular driving, and cost effective. Since these customer-centered metrics are hard to quantify, a Quality Function Deployment (QFD) was used as a tool to help translate these customer benefits into actionable and measurable technical metrics [14]. The results of the QFD assessment are summarized in Table 1, where the rows indicate customer benefits and the columns indicate the technical metrics that were tracked. Each technical metric may be mapped to one or more customer benefits. Furthermore, the QFD analysis provides an understanding to which metrics are the most important given a particular set of customer preferences.
Table 1: Qualify Function Deployment (QFD) relationship matrix that maps customer benefits to technical metrics.

<table>
<thead>
<tr>
<th>Customer Benefits</th>
<th>Technical Metrics</th>
<th>Maximum detection</th>
<th>Algorithm speed</th>
<th>Outlier and noise removal</th>
<th>Operation in direct sunlight</th>
<th>Operation in indirect sunlight</th>
<th>Algorithm complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smoother ride</td>
<td></td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Less wear and tear on tires, wheels</td>
<td></td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>No adverse affect on normal driving</td>
<td></td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Reliability</td>
<td></td>
<td>9</td>
<td>3</td>
<td>9</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Safety</td>
<td></td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Cost</td>
<td></td>
<td>3</td>
<td>3</td>
<td>9</td>
<td>1</td>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>Totals</td>
<td></td>
<td>9</td>
<td>33</td>
<td>21</td>
<td>12</td>
<td>10</td>
<td>4</td>
</tr>
</tbody>
</table>

Scores within the QFD relationship matrix are on an exponential scale (with valid entries of 1, 3, or 9). Such a weighting scheme was chosen to disproportionally weigh the technical metrics that most affected the specified customer benefits [25]. Furthermore, an effort was made to keep the relationship matrix sparsely populated to focus on only the most dominant effects. Since it is assumed that each of the customer benefits are of equal importance, the scores for each technical metric are simply summed, and the totals are reported at the bottom of each column. A higher score means that metric is more important to satisfying the customer’s desires. The scores populating the QFD are based on how the author interprets the technical metric related to the customer benefit. As a result of the QFD analysis, performance against all eight technical metrics from Table 1 were tracked. Because higher scores indicate a higher priority, the primary focus will be on maximizing the “Percentage of potholes detected” and “Algorithm speed”. The remaining six metrics will also be tracked (albeit with a lower relative importance) and an effort will be made to maximize performance against these criteria as well. A stretch goal or future work would be to complete impact analysis of each of these metrics based on results of project tasks, as well as evaluation of several of the lower ranked metrics, such as “Outlier and noise removal”, “Operation in direct sunlight” and “Maximum detection range”, once vehicle on-board testing commences.

To verify the road defect detection/classification algorithm detects a high percentage of road defects in the camera frame, a video with a known number of road defects in each frame (ground truth data) would be fed through the detection algorithm. The ground truth data would then be compared to the number of road defects identified by the algorithm. To achieve the goal, the algorithm would need to acquire and process at least 12 images per second to ensure that images overlap and that road defects were not overlooked. For a successful test, the algorithm should be able to positively identify 80% of all road defects within a video segment. To quantify the accuracy of the algorithm’s ability to detect and classify road defects, results from the road defect detection/classification algorithm are used and compared to the maximum width and length of the road defect as determined from physically measuring the road defect. The resulting percent error will be used as a performance metric for the system to help determine accuracy of the algorithm and quantify one of the two primary impact metrics. The goal is to have the estimated length and width within ±30% of the actual measurement. If the framerate at full resolution (using each and every point in the point cloud)
is not achievable at full driving speed, resolution could be sacrificed to ensure real-time processing. At lower vehicle speeds, the algorithm will be able to use full resolution since a lower framerate will be required. The algorithm will be capable of recording the time elapsed for acquiring the required number of images per second and the post processing of each of the images. Finally, the reaction time of the system will need to be assessed to ensure that, once the sensor and algorithm detects the defect, the adaptive shock can alter its damping before it impacts the road defect. This reaction time will measure the time it takes to adjust the damping ratio of the adaptive shock once the sensor identifies a road defect. Assuming a vehicle is traveling at 70 mph and a road defect is detected 4 meters from contacting the tire, the system will need to react within at least 128 milliseconds. Testing on the bench testing platform will confirm the reaction times of the system.

If a situation arises where the test hardware (i.e., Kinect Sensor) fails or the algorithm fails to adjust the adaptive shock completely or does not adjust it in the timeframe prior to encountering a road defect, the adaptive shock would simply act and respond similarly to a standard shock with a constant damping ratio. This failure would not likely lead to an adverse driving condition for the consumer, and, therefore, does not result in a vehicle systems safety issue.

4 Road Defect Detection and Classification

To ensure the safety of the drivers operating the test vehicle, adjustments to the vehicle steering, braking and/or suspension were not permitted. Real-time, on-road alterations on a vehicle equipped with adaptive suspension would not be utilized. Therefore, a two-pronged approach was adopted to effectively test the detection/classification algorithm separately from the adaptive suspension. This strategy required design and fabrication of a bench testing platform equipped with an adaptive shock that is compressed and extended using a pneumatic actuator. Details on the bench testing platform are discussed in Section 5. The on-road data collection and image post-processing was performed using a test vehicle equipped with a mount for the Kinect sensor. The portion of the mount in which the Kinect sensor was affixed was 3D printed in the Industrial Engineering Workshop at West Virginia University (WVU). This 3D CAD drawing for the Kinect sensor bracket is shown in Figure 2a and an image of the Kinect sensor actually mounted in the bracket is shown in Figure 2b.
The bracket was then attached to a wooded frame which was then securely fastened to the tailgate of a pickup truck using ratchet straps. In order to test the road defect detection/classification algorithm, images of actual road defects needed to be gathered. The test vehicle was outfitted with the Kinect sensor and traveled around the Morgantown, WV vicinity stopping at identified road defects in order to capture static images that returned 3D point clouds of the road surface. The point cloud of the road surface was analyzed to determine if it contained any objects that may constitute a road surface defect (e.g., pothole). The mounting system for the Kinect sensor is shown in Figure 3.

Once images were collected using the sensor and the developed data acquisition code required by the sensor, the proposed detection/classification algorithm processes the raw data from the sensor and identifies a road defect if one is visible in the frame of the sensor. The algorithm then classifies each detected road defect into one of 6 discrete groups (0, 1, 2, 3, 4 or 5) depending on length, width and depth of the defect. This classification will later be used to determine the damping setting for the adaptive shock.
4.1 3D Data Capture Sensor - Microsoft Kinect Sensor

In order to collect 3D point cloud data, a 3D data capture sensor is required to deliver the 3D point cloud necessary for the image processing algorithm that is being developed. The Microsoft Kinect sensor, shown in Figure 4, was chosen as the data acquisition hardware for this research. The Kinect sensor was originally developed and produced to be used in conjunction with the Microsoft Xbox gaming system as a motion-sensing device. The sensor identifies human limbs and joints via skeletal tracking and can even distinguish between an open and closed hand. The sensor achieves this by utilizing both an RGB camera that delivers color images and an IR sensor that provides a 3D depth image. The coordinate system orientation is displayed in Figure 4 which defines the Z axis growing out in the direction the sensor is facing.

![Figure 4: Microsoft Kinect sensor](image)

The RGB camera boasts a color image resolution of 1920 x 1080 pixels, a horizontal field of view (FOV) of 84.1° and vertical FOV of 53.8°. The IR sensor boasts a depth image resolution of 512 x 424 pixels, a horizontal FOV of 70.6° and vertical FOV of 60.0°. These specifications are shown in Figure 5.

![Figure 5: Field of view of Kinect sensor](image)

The depth image provides a 2D, 512 x 424 matrix populated with a depth measurement at each pixel location. While developing the detection and classification algorithm it was discovered that it was beneficial to be able to calculate the size of these single pixels using a real-world measurement such as inches. Using geometry, known depths at each pixel location and FOV specifications from Figure 5, the widths and heights of individual pixels can be calculated.
where \( w_{\text{pixel}} \) and \( h_{\text{pixel}} \) are the pixel width and height, respectively, in terms of meters. These values are later converted to inches. The variable, \( depth_i \), refers to depth values corresponding to the \( i^{th} \) pixel location. \( FOV_h \) and \( FOV_v \) are the horizontal and vertical fields of view in degrees, respectively.

### 4.2 Detection and Classification Algorithm Development

The Kinect sensor supplies a 3D point cloud, but a 3D point cloud alone is not capable of detecting and classifying a road defect. Extensive algorithms and computer-vision techniques are necessary for processing the raw data retrieved from the Kinect sensor. Because the raw data that is retrieved from the Kinect sensor includes both RGB color images and IR depth images, the raw data needs to first be sorted. Once sorted, a Gaussian blur is applied to the data to clean up the data and remove noise.

Section 4.2.1, Section 4.2.2 & Section 4.2.2.1 were developed largely by Andrew Rhodes, a fellow graduate student at West Virginia University (WVU) and former fellow member of WVU’s Applied Space Exploration Laboratory. The topic of this thesis is derived from an Innovation Project associated with the EcoCAR 3 program, an advanced vehicle technology competition sponsored by General Motors and the U.S. Department of Energy. The objective of the program was to convert a 2016 Chevy Camaro into a plug-in hybrid electric vehicle for the purpose of: (1) reducing emission, (2) increasing fuel economy, and (3) maintaining the performance of an iconic sports car such as the Chevy Camaro. The Innovation Project was an ancillary project that required the EcoCAR team (more specifically the Innovation sub-team) to develop an innovative topic to further the state-of-the-art in an automotive related technology.

#### 4.2.1 Geometric Constraints

To simplify the algorithm and increase the speed of the image processing, some geometric constraints can be exploited. Assume a vehicle is equipped with a 3D camera pointed towards the road. The camera returns a 3D point cloud of the road surface in the camera frame. It is then mathematically beneficial to transform this point cloud into the wheel frame as

\[
p_w = R_w^c p_c + r_w \tag{3}
\]

where \( w \) and \( c \) refer to the wheel and camera frames respectively, \( R_w^c \) is a \( 3 \times 3 \) rotation matrix from the camera frame to the wheel frame, \( r \) is a \( 3 \times 1 \) translation vector in the wheel frame, and \( p \) is an \( n \times 3 \) matrix of point locations in either the wheel or camera frames. The transformation consisting of \( R_w^c \) and \( r_w \) are known since the camera is rigidly mounted to the vehicle. Without loss of generality, the origin of wheel frame is defined as the center of the line connecting the contact points between the two front wheels and the ground.
as depicted earlier in Figure 1b. The point cloud in the wheel frame now has a few interesting geometric constraints that support detection of road surface defects. First, assume that the measured road surface is planar. The camera is pointed down towards the ground at an angle of 30° below the local horizontal as shown in Figure 6. Because the FOV for the Microsoft Kinect is 70.6° x 60°, the result is a usable measured point cloud of about 4 x 3 meters. Images of road defects were captured using a variety of angles (i.e., 25°, 30°, 35°, 40°...etc) and the angle of 30° was chosen because it provided the best range and a large amount of points within the road defect which enabled an accurate classification.

![Figure 6: Geometry of road surface parameterization](image)

It is reasonable to assume that a measured road section of this size could be classified as planar. Then the equation of a planar road surface is

\[ n_x x + n_y y + n_z z + d = 0 \]  \hspace{1cm} (4)

where \( n = [n_x, \ n_y, \ n_z]^T \) is the surface normal vector of the plane and \( d \) is the signed perpendicular distance from the origin to the plane. The normal vector may also be defined in terms of road pitch \( \alpha \), and the road bank \( \beta \). By examining Figure 6, it is clear that the surface normal may also be expressed as

\[ n = [\cos(\beta)\cos(\alpha), \ \cos(\beta)\sin(\alpha), \ \sin(\beta)]^T \]  \hspace{1cm} (5)

Second, notice that the wheels of the vehicle are always in contact with the road. Thus the signed distance, \( d = 0 \), is because the road plane is in contact with the origin of the wheel frame. Also notice that if the z-axis of the wheel frame is pointing vertically up towards the sky, and since the wheels are in contact with the road, the nominal surface normal vector for the road plane in the wheel frame is \( n_w = [0, \ 0, \ 1]^T \).

Suppose that the vehicle is approaching an inclined hill, but has not yet begun ascent. In this scenario, the vehicle is on the flat road surface while the camera returns a point cloud of the hill. The normal vector of the plane that describes the hill is not the same as the nominal normal vector because the wheels are not yet in contact with the hill. The goal is to find the normal vector of the hill. Let the signed perpendicular distance, \( \epsilon_i \), of any measured point to the plane be given as
\[ \epsilon_i = (x_i - \bar{x}) \cos(\beta) \cos(\alpha) + (y_i - \bar{y}) \cos(\beta) \sin(\alpha) + (z_i - \bar{z}) \sin(\beta) \]  

(6)

where

\[ \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i, \quad \bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i, \quad \bar{z} = \frac{1}{n} \sum_{i=1}^{n} z_i \]  

(7)

In order to meet the requirement of real-time implementation, an efficient solution to the variables \( \alpha, \beta \) is sought. Begin by de-normalizing the measurements

\[ \hat{x}_i = x_i - \bar{x}, \quad \hat{y}_i = y_i - \bar{y}, \quad \hat{z}_i = z_i - \bar{z} \]  

(8)

and then stacking all the measurements

\[ \epsilon = \begin{bmatrix} \hat{x}_1 & \hat{y}_1 & \hat{z}_1 \\ \hat{x}_2 & \hat{y}_2 & \hat{z}_2 \\ \vdots & \vdots & \vdots \\ \hat{x}_i & \hat{y}_i & \hat{z}_i \end{bmatrix} \begin{bmatrix} \cos(\beta) \cos(\alpha) \\ \cos(\beta) \sin(\alpha) \\ \sin(\beta) \end{bmatrix} \]  

(9)

or equivalently as

\[ \epsilon = D\delta \]  

(10)

The numeric solution is found by taking the singular value decomposition (SVD) of \( D \) as

\[ D = U \Sigma V^T \]  

(11)

The last column of \( V \), which corresponds to the smallest singular value of \( \Sigma \), is the closest numeric solution to the vector \( \delta \). Since \( \delta = n \), it must be normalized to \( ||\delta|| = 1 \) or

\[ \sqrt{\sin(\beta)^2 + \cos(\beta)^2 \cos(\alpha)^2 + \cos(\beta)^2 \sin(\alpha)^2} = 1 \]  

(12)

We find the optimal values of \( \alpha, \beta \) by:

\[ \alpha = \arctan \left( \frac{\delta_2}{\delta_1} \right), \quad \beta = \arcsin(\delta_3) \]  

(13)

where \( \delta_1, \delta_2, \) and \( \delta_3 \) are the components of the vector \( \delta \). By applying these geometric constraints inherent to the situation, the values of \( \alpha \) and \( \beta \) completely define the plane of the hill. Fast and efficient point cloud processing algorithms may now be applied to separate the defective areas from the road surface.
4.2.2 Point Cloud Processing

Processing of the measured point cloud must be fast and efficient for real-time implementation on a vehicle. The algorithm also must be deterministic in order to have repeatable and dependable results. Section 4.2.2.1 presents the the portion of the algorithm necessary to separate outlier points from the road surface and cluster the outlier points into groups. Presented here is a novel and innovative method of detecting road defects from a measured road plane using a 3D point cloud.

4.2.2.1 Determine Point Cloud Inliers and Outliers

Suppose there is a noise free measured point cloud of a road surface in the wheel frame. According to Equation 4 and the geometric constraints stated in Section 4.2.1, \( d = 0 \) and every point has the same normal vector of \( \mathbf{n}_z = [0, \ 0, \ 1]^T \). This, however, is not ordinarily the situation because of sensor measurement noise and the presence of road surface defects. These two causes present similar point cloud distortions and are thus difficult to analyze separately. A method of detecting point cloud outliers from a planar road surface is presented in this section.

For this procedure, the normal vectors of each point in the point cloud must be found. Normal vectors are not a measurement returned by the sensor, but they may be calculated using any number of methods \[26]\[27]. All of these methods operate similarly to finding the normal vector to the road plane as presented in Section 4.2.1, but apply more restrictions on the local curvature and inter-point distances. Algorithms for calculating normal vectors for every point in a point cloud are found in nearly all point cloud processing applications and do not present any risk to the overall real-time operation. Since this process is not the focus of this report, these values are assumed to be available for use in further analysis.

In the current constrained situation, there are two criteria that may be used to define an outlier point:

1. Has a normal vector that is significantly different than the nominal. The inner product (dot product) of \( \mathbf{n}_i \) and \( \mathbf{n}_z \) is less than an angular threshold \( \theta_a \).

\[
\mathbf{n}_i^T \mathbf{n}_z < \theta_a \quad (14)
\]

2. Lies far from the true road surface plane. The inner product of \( \mathbf{p}_i \) and \( \mathbf{n}_z \) is greater than a Euclidean distance threshold \( t_d \).

\[
\mathbf{p}_i^T \mathbf{n}_z > t_d \quad (15)
\]

Consider Figure 7a which shows a point cloud with outlier points (in red) lying some distance away, but parallel to the road surface plane (in black). In this situation, the angular differences between the normal vectors of the outlier points and the road surface normal vector are minimal. These points are considered outliers because their signed perpendicular distances from the road surface plane are large compared to the sensor noise. Alternatively, examine Figure 7b which shows a point cloud with outlier points lying near to, but tilted at some angle with respect to the road surface plane. While the signed perpendicular distances between the points and the plane are small, these points are considered outliers because the angular differences...
between the normal vectors and the road surface normal vector are significant. By simultaneously examining both 1) the signed perpendicular difference to the plane, and 2) angular differences in the normal vectors compared to the road surface normal vector, outliers may be robustly detected in many more situations than if only one characteristic was examined.

Further analysis of the interplay between the two characteristics reveals details about other possible road surface situations. Examine Figure 8 which is a scatter plot of normal vector alignment with respect to road surface normal vectors and the signed perpendicular distance to the plane for all points in a given point cloud. This plot has been designated as the Alignment-Distance Graph (ADG).

Different types of road defects (or obstacles) appear in different locations on the ADG. The blue area consisting of a square and triangular region is considered the road plane. The midline of the road plane is determined by capturing images of known flat surfaces such as a tile floor and then fitting a plane surface to the points. Any point within 0.003 meters (0.118 in.) from this surface is also included in the road surface or inlier points. This threshold was experimentally chosen to ensure a substantial number of inlier points and outlier points. Any points outside of the area defined as the road plane are labeled as outliers. Then there are two sets of points: inliers placed in the set $I$, and outliers placed in the set $O$. 

Figure 7: A noise free case of a road plane containing defects. Road defects shown in red demonstrate the importance of analyzing both normal vector alignment and perpendicular distance to the plane. This figure is best viewed in color.
Since points constituting a road surface defect may have either one or both of these aforementioned characteristics, it is beneficial to examine both together. The set of inlier points, $I$, in blue on Figure 8 is defined by

$$I = p_i : \begin{cases} \mathcal{I}_1 : (n_i^T n_z < t_a) \land (|p_i^T n_z| < t_d) \\ \mathcal{I}_2 : (n_i^T n_z > t_a) \land (p_i^T n_z < m(n_i^T n_z - t_a) + t_d) \\ \mathcal{I}_3 : (n_i^T n_z > t_a) \land (p_i^T n_z > -m(n_i^T n_z - t_a) - t_d) \end{cases}$$

(16)

where $I = \mathcal{I}_1 \cup \mathcal{I}_2 \cup \mathcal{I}_3$, $m = nt_d / (1 - t_a)$, $n_i$ is the normal vector of $p_i$, and $t_a$, $t_d$ are the angular and distance thresholds defined in the criteria. The current implementation uses $t_a = \cos(30^\circ)$ and $t_d = \sigma_s$, where $\sigma_s$ is the standard deviation of the sensor’s measurement noise. An example of the inliers identified in a single image represented in 2D form is shown in Figure 9.

Figure 9: 2D illustration of 3D points representing inliers or the planar road surface

The set of outlier points, $O$, is defined as

$$O = p_i : p_i \notin I$$

(17)

or all points not belonging to the set of inliers.

4.2.2.2 Road Defect Classification using Saliency Maps

The preliminary version of the developed Matlab code was capable of identifying the potential outline of a road defect. The algorithm correctly processed the point cloud, and categorized each point as either being within the plane of the road surface (inliers), an outlier above the surface, or an outlier below the surface (i.e., a pothole), but it did not estimate the length, width, or the approximate depth. An example of the point cloud categorization is shown in Figure 10. It is desirable to categorize the characteristics of the road defects for use in later situations such as determining the time to impact, creating a catalog of road defects for maintenance crews [13], or for avoiding heavily defected roads with GPS navigation [30]. Particularly important characteristics for the purpose of this thesis includes defect length, width and depth. Charac-
teristics that may be important for the other situations may include (but are not limited to) defect change through time, material (e.g. asphalt, concrete), and geographic location.

Figure 10: 3D point cloud illustrating inlier points (shown in blue), outlier points above the road surface (shown in red), and outlier points below the road surface (shown in green) and representing the road defect.

Using the inlier points in the road surface, $I$, and outlier points below the surface, $O$, a clustering technique is applied that divides each image into a checkerboard of blocks (30 rows and 30 columns) and then compares the number of points in each category. If the number of inlier points in the block is greater than the number of outlier points below the road surface in the block, the outlier points are dismissed as being considered part of the road defect under the probability that the 3D points in said block are more likely to belong to the plane of the road surface. The number of divisions in the rows and columns to produce the checkerboard were determined by processing various examples of road defects and visually examining the resulting image (such as seen in Figure 11b). The desired number of divisions chosen was based on a road defect that was accurate in length and width compared to the actual length and width while keeping the computational cost of comparing each block of the checkerboard to a minimum. Figure 11 shows images of the outliers below the road surface before clustering and after clustering. It also shows the outliers overlaid on the inliers (road surface points). The elimination of points can be seen from Figure 11a to Figure 11b. This refinement is the direct result of the clustering technique and represents one step toward defining the outline of the road defect.

Figure 11: Shown above is: a) outlier points below the road surface before clustering, b) outlier points below the road surface after clustering, c) points from (b) overlaid onto inlier points (points within road surface)
After the clustering is completed, a saliency map is applied to the image containing outlier points overlaid on inlier points (see Figure 11c). The saliency map isolates foreground pixels from background pixels in an image to yield a blurred, grayscale image of the road defect as shown in Figure 12b. The pixels in this image contain a range of gray shades from black to white, where black pixels contain a value of 0, white pixels contain a value of 1 and gray pixels contain a value between 0 and 1.

Figure 12: Shown above is: a) actual color image of the road defect, b) blurred image from saliency map, c) smoothed image of road defect including the length and width indicated by the red lines

The blurred, grayscale image of the road defect is then smoothed using a threshold determined by experimenting with different threshold values and then comparing algorithm-generated lengths and widths with the road defect true lengths and widths. The algorithm then displays the 2D binary, black-and-white image of the road defect containing only zeros and ones. Using this binary, black-and-white image the algorithm scans the rows of pixels from top to bottom searching for pixels containing ones. It identifies the uppermost row containing a one and the lowermost row containing a value of one. If there happens to be multiple ones contained in these rows, the algorithm identifies the location of the middle pixel in these rows. Using this uppermost pixel and lowermost pixel, the algorithm produces a red line connecting the two, defining one major axis of the road defect. This process is repeated from left to right to determine the second axis of the road defect. This yields the length and width of the road defect as shown in Figure 12c.

Length, width and depth measurements are first checked to determine if they are under a specified threshold. For instance, if the width and depth of the road defect are significant but the length of the road defect is less than 1 inch, then there is no need to adjust the shock. These thresholds are strategically chosen to ensure that there would be nominal effect on the ride quality of the consumer if a tire of average size and width were to encounter a defect parameterized by lengths, widths and depths of equal or lesser value to these thresholds. After initial checks are done on each of the road defect dimensions and it is found they are beyond the thresholds, then the algorithm calculates the volume of the road defect and classifies it into one of 6 discrete values that categorizes the road defect between 0 and 5 based on the volume. Zero is the mildest of the road defects and will require small, if any, adjustment of the adaptive shock. Five is the most severe and covers all road defects with calculated volumes over 350 cubic inches. The number of discrete values were arbitrarily chosen and these discrete values were used because it simplified the controls algorithm for the bench testing platform. A compilation of images of road defects were collected from the Morgantown,
WV area which included actual length, width and depth measurements. These images were used to evaluate the accuracy of the dimension estimation from the algorithm.

4.3 Real-Time Data Collection and Code Conversion

To effectively capture and process data quickly enough to react to road defects with adequate speed, the overall data collection and processing operations from the Kinect Sensor have to be executed at a rate of at least 12 Hz. A value of 12 Hz, or 0.0833 seconds, is determined by assuming a vehicle traveling at 70 mph, or 31.2 m/s, and a Kinect Sensor maximum range of 4 meters while affixed to a vehicle at a 30 degree downward angle from horizontal. The specification regarding the angle of the sensor and the subsequent range is determined from experimentation during the on-road data collection sessions. If only image processing time were taken into account, the algorithm could potentially operate at a minimum of 8 Hz, or 0.125 seconds. Because the systems requires additional time for the adaptive shock to respond once it receives an electric signal, a buffer is built into the algorithm speed requirement. Therefore, if the algorithm takes 0.0833 (12 Hz) seconds to process the image, the adaptive shock has 0.0416 seconds to alter the damping, keeping the combined response of the system within the 0.125 second requirement needed for the system to operate effectively at speeds of up to 70 mph. While 12 Hz is determined as the minimum operational speed for the data capture according to vehicle velocity and sensor range, the Kinect sensor is reported as capable of capturing 30 frames per second (fps). Therefore, the ideal operational speed of the data collection script would be 30 Hz, or 0.0333 seconds. The original MATLAB script responsible for early data gathering was not adequate for this operation, usually running at 1-2 Hertz with little room for optimization. To remedy this limitation with the MATLAB script, a C++ program is developed and compiled to vastly improve the Kinect Sensor data capture rate. If the algorithm processing speed goal can not be met, resolution might have to be sacrificed or the top speed that the vehicle can travel with defect detection active will have to be decreased. Reducing the vehicle speed will give the vehicle more time for the algorithm to process before the road defect hits the tires, but would limit the prototype system to low-speed driving which is problematic. A reduction in resolution would decrease the time per frame by at least the percentage of resolution reduction.

A compiler such as C++ is a low-level programming language that generates a relatively optimized machine code whereas MATLAB is a high-level programming language. MATLAB has the benefit of containing libraries with commands already included to perform commonly used functions. However, when executed, the functions must first be converted to machine code. The lower-level C++ code is closer to the machine code and executes scripts more efficient and faster than MATLAB especially when there are large amounts of code to be executed.

The program being developed for capturing images was written in C++ using Microsoft Visual Studio 2017 and the publicly available Kinect SDK 2.0 code. The converted program initiates the sensor and waits for user input. Once the user triggers a frame capture by laptop keyboard input, a frame is be captured and stored in a plain text file for observation. A timer reports how long it takes to capture a frame and displays it at the end of the test for observation. This frame is then be stored in a one-dimensional array of vectors that stores each distance value for each “pixel” of the depth sensor (i.e., a two-dimensional image with pixel values representing the distance from the XY plane of the Kinect Sensor as shown in Figure 13). The recorded time is saved to later be used in an analysis illustrating the improvement in speed of the C++ code.
over the MATLAB code.

5 Adaptive Shock Experimentation

As discussed in Section 4, because altering of the vehicle steering, braking and/or suspension is prohibited, it is not feasible for the adaptive suspension portion of the road defect mitigation system to be incorporated into a test vehicle. Therefore, a separate bench testing platform had to be designed, developed and fabricated in order to simulate a single adaptive shock contacting a road defect such as a pothole. The testing platform will perform tests necessary to confirm the ability of the system to properly alter the damping of the adaptive shock and to verify that the system was capable of altering the damping of the shock prior to the tire striking the road defect under normal vehicle operating speeds. The data necessary to validate the results of these tests is provided by a suite of sensors incorporated into the best testing platform.

5.1 Bench Testing Platform

A bench testing platform was developed to apply a dynamic load to an adaptive shock to test both time response of the system to signals sent from the detection algorithm, as well as dynamic response of the adaptive shock and ability to dampen applied impact forces. A Monroe OESpectrum 1502 shock with adjustable damping was procured to test the defect detection/classification algorithm and to determine response time. The adaptive shock used in this experimentation is shown in Figure 14.
The test rig consists of a rectangular, steel frame fabricated from 1” square tubing that is welded to a $\frac{1}{2}$” thick base plate made of steel. Attached to the base plate is a pneumatic piston-cylinder system which has an adaptive shock pinned to the top of the piston. The other end of the shock is attached to a $22'' \times 22'' \times \frac{1}{2}''$ thick “floating” plate with 4 - 6” sections of $1\frac{1}{8}''$ square tubing welded into each corner allowing the plate to travel along the lengths of the upright tubing on the frame (see Figure 15: Section A). The design consists of weights that are cut out of $\frac{1}{2}''$ steel plate, using a CNC plasma cutter, with additional weights stacked onto the “floating” plate to simulate the weight of the vehicle over one wheel. Figure 15 shows the “floating” plate (in yellow) which supports the weights. Between the “floating” plate and shock, a compression load cell is positioned. Figure 16 shows 412.5 lbs of weights stacked onto the actual bench testing platform. Standard weights similar to the weights used for weightlifting were acquired from a fellow student. The amount of weights used in the bench testing platform was determined by researching the average weight applied to a single shock in a suspension system. A piece of 1” steel, round stock is welded on top of and centered on the “floating” plate”. This makes it easy for the weights to be added and removed. A 1” clamping collar was obtained and tightened down on the round stock after the weights are added. This collar is put in place as a precautionary measure to ensure the safety of the bench testing platform operators by prohibiting the weights from sliding off of the shaft during operation.
A pneumatic actuator extends/compresses the shock so that it simulates a car wheel encountering a road defect. The piston-cylinder actuator system is double-acting and operated using compressed air whose flow...
is metered and controlled by a regulator and an electromechanical, 3-position air directional control valve or solenoid. This control valve was chosen because the 3 positions make it suited to actuate a double-acting piston-cylinder system going both up and down. The actuator is shown in Figure 17 and the regulator and solenoid are shown in Figure 18a and Figure 18b, respectively.

![Actuator](image1.png)

Figure 17: Double-acting pneumatic actuator

![Regulator and Solenoid](image2.png)

(a) Compressed air regulator  
(b) Directional air control valve

Figure 18: Compressed air metering and control components
When the pneumatic cylinder pushes against the adaptive shock, the load cell records the force acting on the shock. This apparatus could also be used as a visual aid to demonstrate the displacement of the weights with the algorithm not active and then comparing to the displacement of the weights when the algorithm is active. The assumption is that the weights will experience a decreased displacement when the algorithm is adjusting the adaptive shock prior to striking road defects. The apparatus is also used to determine if the design and the adaptive shock itself can withstand the required forces that are applied to the rig once the weights are added, ensuring the pneumatic piston is able to lift the weights required for the tests. To consider the adaptive shock successful, it is determined that the shock will need to limit the displacement of the “floating” plate a maximum of 3″ of its original position under standard compression with the algorithm active. The maximum value of 3″ is determined under the assumption that, on average, the majority of road defects will not have a maximum depth of more than 6″ and the weight representing the vehicle will be required to displace less than this maximum depth (assumed 50% or 3″) indicating a smaller disturbance to the passenger.

Locating and acquiring an adaptive shock that could be used for the bench testing platform proved to be a difficult task as these types of components are normally reserved to be sold to automotive distributors and maintenance garages. Additionally, there was limited documentation on the adaptive shock components and operating specifications, largely due to the proprietary information regarding the electrical signal range required to adjust the damping of the adaptive shock. Also longer than expected lead times were experienced on delivery of the solenoid. The fabrication of the bench testing platform was made possible, in part, by the EcoCAR lab and Industrial Engineering Workshop at WVU which provided the required tools necessary for fabrication of the test platform such as a welder, CNC Plasma cutter and a manual Bridgeport Mill. The workshops also contain several 3D printers that were utilized to produce parts such as a mounting bracket for the Kinect sensor to be mounted to the test vehicle and a housing that is mounted to the test rig which secures a rocker switch in place.
5.1.1 Controls Development

In order for the completed algorithm to be integrated with the adaptive shock and pneumatic cylinder in the bench testing system, a set of controls needed to be developed. A discrete, closed-loop feedback control system was designed for integration of the pneumatic cylinder and adaptive shock. A finite number of 6 outputs, ranging from 0 to 5, were designated according to the calculated volume of the road defect. These outputs are sent to the adaptive shock in the form of an electrical signal.

A control system was fabricated and assembled that utilizes a Woodward model HCM-0563 (Hydraulic Control Module) controller to operate the electromechanical, spring-returned solenoid and thereby the pneumatic cylinder to which it is connected. The Woodward controller is a sealed control module complete with a Freescale MPC563, 40 MHz microprocessor. The module uses the Simulink programming language and is capable of delivering complex control strategies using up to 16 analog inputs and up to 10 outputs. Because the module is designed for use in the automotive industry, it also includes output pins for a CAN bus which is also commonly used in the automotive industry to transmit data and is utilized in this experimentation to send sensor data to a CAN logger. The Woodward controller requires two connectors for operation which are labelled Connector A and Connector B. The pin-out configuration of these two connectors are shown in Appendix A and Appendix B, respectively. The solenoid is a control valve with 3 positions and 5 ports that directs compressed air from a source (P) in two possible directions depending on the position of the spool within the solenoid. A schematic of the pneumatic plumbing in the bench testing platform is shown in Figure 20, including a diagram showing the 3 positions of the solenoid.
Currently the Woodward controller is flashed with a Simulink model that consists of two unique modes that can be selected using a toggle switch. When the toggle switch is in the off position, a separate rocker switch can be switched in one direction energizing one of the relays in the air directional control valve via a 12 V signal, sliding the electromechanical spool within the solenoid to one side for a set period of time (currently 50 milliseconds (ms)) and then returns to the center position moving the piston upward a specified distance. In the right position the solenoid supplies compressed air to port B, while port A is exposed to an exhaust port. The compressed air flowing through port B is attached to the bottom of the cylinder forcing the piston up. When the electrical current is removed, the spool returns to the center position halting any flow of compressed air and any corresponding movement of the piston. Alternatively, when the rocker is switched in the other direction, the opposite side of the air directional control valve is energized and the solenoid spool moves in the opposite direction for same amount of time. This movement allows compressed air to flow through port A. The change in flow also reverses the movement of the piston forcing it downward. Again when the electrical current is removed, the spool returns to the center position halting any flow of compressed air and any corresponding movement of the piston. A Simulink model of the controls algorithm used to operated the bench testing platform is shown in Appendix C.

When the toggle switch is turned on, the actuator acts on the shock simulating a series of predetermined road defects with varying lengths and depths over a period of 30 seconds. Once the series of predetermined road defects have been completed, the program ends. The program can be rerun by switching the toggle switch off and back on again. A schematic illustrating the electrical wiring is shown in Figure 21.
Preliminary tests were performed on the controls which actuate the solenoid thereby moving the piston up and down. With zero weights stacked on the “floating” plate and the compressed air regulated to 100 psi, the piston-cylinder system is actuated one step at a time opening the solenoid for 50 (ms) during each step. The distance that the piston travels is recorded after each step and this process is completed for the piston moving up and then moving back down. These measured values of piston travel are plotted in Figure 22, and it can be seen that the movement of the piston compared to the amount of time the solenoid remained open maintains a linear relationship in either direction. The equations of both lines are displayed in slope-intercept form and, relative to the pressure, weight and time parameters, the recorded data demonstrates the piston's rate of change in position (speed and therefore total distance traveled) remains the same whether going up or down.

Keeping the compressed air regulated to 100 psi, weights are stacked onto the “floating” plate totaling 412.5 lbs. When the same test is performed as described above actuating the piston-cylinder system one step at a time by opening the solenoid for 50 ms for each step, it is observed that the “floating” plate does not travel upward as far as it does downward. Therefore, to ensure the actuator travels upward at the same rate as it
travels downward, the amount of time the solenoid stays open for upward travel needs to be greater than
the amount of time the solenoid stays open for downward travel. The Woodward Controller only permits
this amount of time to be set intervals such as 25 ms or a multiple of this time. Therefore it was determined
to use a time of 100 ms when the piston travels upwards and 25 ms when the piston travels downward.
These times guarantee that the piston travels relatively the same distance whether going up or down with
the specified air pressure and weights applied. This result can be seen by the similar slopes (one increasing
and one decreasing) in the two plots shown in Figure 23.

![Figure 23: Piston travel upward/downward using 2 different solenoid open times](image)

5.1.2 Sensor Selection

There were 3 dynamic sensors chosen to be integrated into the bench testing platform in order to collect
the data necessary to validate the testing results and confirm that the algorithm and controls performs as
intended. All 3 of the sensors are configured to be read by the Woodward controller. The 3 sensors are as
follows, each will be detailed in the following sections:

1. **Laser Distance Measurement Sensor (LDMS)**: Measures the distance and therefore the dis-
   placement of the “floating” plate by subtracting the absolute distance from an initial datum line.

2. **Compression Button Load Cell**: Measures the force acting on the adaptive shock.

3. **Single-Axis Accelerometer**: Measures the acceleration of the “floating” plate which relates to
   consumer ride experience.

5.1.2.1 Laser Distance Measurement Sensor

The laser distance measurement sensor (LDMS) selected is the Wenglor OPT2011. The LDMS uses a Class
1 laser light emission to measure the distance from the base of the bench testing platform to the “floating”
plate. This sensor was chosen because it is a short-range, high-precision sensor with a response time of 2
milliseconds and is capable of having the output signals configured to a range of 0 - 10 volts (V) or 4 - 20
milliamperes (mA). The LDMS also has a digital screen which displays the output signal (in V or mA) and
the distance in millimeters (mm). An image of the LDMS used in the bench testing platform can be seen in
Figure 24a and the actual LDMS is shown in Figure 24b.
5.1.2.2 Compression Button Load Cell

The compression button load cell chosen was a RobotShop RB-Phi-205 with a maximum load capacity of 1000 kg. The load cell uses strain gauges to determine the forces acting on the adaptive shock. It was selected because it is relatively inexpensive and the 1000 kg load capacity is capable of handling the amount of weight that would be applied to the bench testing platform in order to simulate the weight of 1/4 of a vehicle. To determine the conversion of the output signal (in millivolts) from the sensor to a force in kilograms, weights were added to the bench testing platform in relatively equal increments and the electrical signal was read using a multimeter and recorded at each increment. The data is plotted and a linear curve is fit to the data to determine the relationship between the signal and the force or weight in kilograms. This relationship is shown in Figure 25.

As shown in Figure 25, the electric signal output from the compression load cell is in millivolts which is typically too small a value to use in data acquisition. Therefore the signal had to be preconditioned using...
an amplifier to boost the signal into a range that can be read by the Woodward controller. Because the Woodward controller requires input signals to be within a range of 0-5 V, a Smowo RW-ST10A amplifier was chosen to achieve this required specification. The amplifier boasts a durable, waterproof design and is capable of supplying an output voltage in a range of 0-10 V or 0-5 V, so it is a suitable solution to amplify the compression load cell sensor. An image of the load cell used in the bench testing platform can be seen in Figure 26a and the actual load cell mounted on the bench testing platform is shown in Figure 26b.

![Load Cell](image)

(a) RobotShop Rb-Phi-205 compression button load cell (1000 kg)  
(b) Load cell mounted on bench testing platform

Figure 26: Compression button load cell

5.1.2.3 Single-Axis Accelerometer

A Dytran 3035B single-axis accelerometer was acquired through West Virginia University’s Advanced Mechatronics Laboratory along with an amplifier. The accelerometer proved to be most difficult to integrate into the bench testing platform. As mentioned earlier in this section, the Woodward controller requires an input signal in the range of 0-5 V from the sensors. The accelerometer provides an output signal in a range from ± 5 V. In order to convert this signal into a range of 0-5 V an operational amplifier (op-amp) is utilized and constructed on a protoboard. The designed op-amp shifts the range up 5 V converting the ± 5 range into a 0-10 V range and then scales down the signal to a range of 0-5 V. These two processes are executed using a combination of a summing op-amp and an inverting op-amp. An image of the accelerometer used can be seen in Figure 27a and a diagram of the operational amplifier used is shown in Appendix D. The actual accelerometer mounted on the bench testing platform can be seen in Figure 27b.

![Accelerometer](image)
As mentioned earlier in this section, a 12 V signal is used to energize the relays of the directional control valve. This voltage is provided by a 12 V power source shown in Figure 28. This 12 V power source also powers the Woodward controller as well as the rocker switch and toggle switch. Figure 28 also shows the accelerometer amplifier which boosts the signal of the accelerometer from millivolts to volts. The power for the operational amplifier is provided by a 3-channel power source capable of providing positive 0 - 5 V and negative 0 - 5 V required by the operational amplifier which scales and then shifts the voltage of the accelerometer into a signal that can be used by the Woodward controller. The voltage on this same power source can be metered and adjusted making it an optimal choice to power the adaptive shock as well. These power sources, amplifiers and switches are shown and labeled in Figure 28.

5.1.3 Testing Adaptive Suspension on the Bench Testing Platform

The sensors detailed in the previous section were incorporated into the bench testing platform. The Woodward Controller was then wired to a Vector CANcase XL Log which logs all the data transmitted by the
sensors. The data can be logged on an SD card or transmitted to a computer via a standard USB printer cable using the development and testing software tool called CANoe.

The bench testing platform was developed using the previously mentioned sensors with the ambition to validate two important specifications of the adaptive shock and the defect detection/classification system.

1. **Functionality of the Adaptive Shock**

2. **Response Time of Adaptive Shock**

As mentioned in Section 5.1, the adaptive shock that was procured did not include documentation. The lack of documentation means that the voltage range required by the adaptive shock is unknown which makes validating the functionality of the adaptive shock a trial and error procedure. It must first be determined if the electric signal being sent to the adaptive shock actually alters the damping ratio of the shock. To test this, an automated series of road defect simulation built into the bench testing controls is used as well as the compression load cell. Instead of using the simulation with a series of road defects with varying sizes and depths, a separate simulation is created with a series of uniform road defects which is more suited to performing baseline testing. The uniform series of road defects that needed to be achieved is illustrated in Figure 29 and an example of the actual displacement of the “floating” plate from the simulation using the pneumatic actuator is shown in Figure 30.

![Figure 29: Proposed simulation of a series of uniform road defects](image-url)
It was theorized that, when an electric signal is sent to the adaptive shock to alter the damping (either 0, 1 or a value between the two), the mean of these peak forces will also be altered due to a change in the stiffness of the shock. To test this theory, the shock is sent an initial discrete value of 0 indicating the “loosest” damping setting of the adaptive shock. With the damping set to 0, the shock is subjected to full compression and the forces are measured using the compression load cell. An example of how the resulting forces appear due to the testing and measured by the compression load cell can be seen in Figure 31. With this baseline of forces from a series of test runs, the mean of the maximum peaks of the forces can be calculated and recorded. An example of the resultant maximum peaks are illustrated in Figure 31. The $X$ values indicate the time in seconds and the $Y$ values show the forces in kilograms.
This process is then repeated, each time sending a different discrete voltage value to the adaptive shock. The increase in value being sent to the adaptive shock indicates a stiffening increase in the damping of the shock. After each series of testing runs, the maximum force and mean of the peak forces are observed and recorded.

Once it is determined that the adaptive shock can reliably be controlled by sending it electric signals, the next step in the validation process can begin. It has to be demonstrated that, once the electrical signal is sent to the shock, the adaptive shock can alter the damping ratio prior to striking the road defect assuming a vehicle traveling at an average highway speed of 70 mph. As discussed in Section 4.3, the shock has to respond to the electrical signal and alter the damping within 0.0416 seconds in order to mitigate the effects of the road defect. If it is determined that the shock requires more time to alter the damping, the detection and classification algorithm speed would need to be increased to ensure the combination of the processes (defect detection/classification and damping alteration) are completed within 0.125 seconds. Testing the response time of the adaptive shock is accomplished using a similar simulation of uniform road defects shown in Figure 30. The actuator retracts allowing the adaptive shock to extend and then, after a period of time, the actuator extends compressing the shock. The force at the shock compression is measured. For the next step, a signal is immediately sent to the shock to alter the damping ratio after the shock extension. Again, after a period of time, the shock compresses and the force is measured. Depending on whether the signal being sent to the adaptive shock loosens or stiffens the shock, the force measured should respond accordingly. The initial period of time is chosen arbitrarily to guarantee that the shock has had a sufficient amount of time to alter the damping ratio before compressing the shock and recording the force. To determine the actual response time, the amount of time between the electric signal being sent and the compression of the adaptive shock is gradually reduced in equal increments until it can be concluded by the unchanged force that the adaptive shock does not have enough time to alter the damping. This iterative test is illustrated in Figure 32. To more clearly show how the timing of the electrical signal initiation alters from one iteration to the next, Figure 33 outlines the process followed.
To extend the lifespan of various vehicle components, and to make the ride smoother and safer for consumers, the vehicle suspension must be modified to more efficiently absorb the impact of encountering a defect. Since the novel road defect detection and classification algorithm preemptively identifies road defects, the suspension system is pro-actively adjusted. This is substantially different than current market applications which wait until impact to modify the suspension [8][9].
6 Results

According to the Quality Function Deployment in Figure 1, success was measured as the ability to achieve the technical metrics. Metrics that affect the algorithm were evaluated for completeness and quality. The metrics corresponding to the physical properties of the sensor – maximum detection range, operation in direct sunlight and inclement weather, and cost – were analyzed through a literature review and product search. The following sub-sections report the results from point cloud image processing using the detection/classification algorithm and the functionality results of the adaptive suspension using the bench testing platform.

6.1 Real-Time Data Collection and Code Conversion

The new C++ code for capturing data was tested alongside the original Matlab data gathering script, to effectively observe and compare the timing differences between the two. In simple observation, the C++ program is noticeably faster, reporting a frame capture time of 125-1000 Hz during execution. This is faster than the Matlab script by nearly 3 orders of magnitude, far surpassing the desired time for data capture, and offers a great deal of timing flexibility for the data processing code. The increased speed of data capture helps achieve the goal of attaining a high algorithm speed capable of operating in a real-time environment which is identified as the 2nd most important metric (Algorithm Speed) according to Section 3. The program for data collection needed to be connected to the post-processing algorithm, so the collection program needed to work on the same development platform as the processing program. This allows rapid resource transfer (an overall program will share resources, no transfer or temporary storage necessary) and also provides a common development foothold such that multiple scripts (data collection and image processing) can be executed with the same program. The source code for the C++ program is included in Appendix F.

Converting the road defect detection and classification algorithm to C++ was necessary to improve the speed of the algorithm. The Kinect sensor can capture data at 30 fps so ideally the algorithm needs to run in under 0.0333 seconds to process frames in real time with the maximum frame rate. The MATLAB implementation is not capable of delivering this speed. The development of the C++ post-processing algorithm is based on the popular image-processing library OpenCV 3.4. This library has been well tested and cut down on the amount of work required to convert the MATLAB code to C++. The conversion process is critical for delivering on the impact metrics, since real-time processing on road defects is required to allow the shock to react before tires encounter the road defect. Testing of the converted algorithm was required to evaluate if the algorithm speed is sufficient for real-time detection which is an impact metric goal defined in Section 3. The converted code was tested and it is determined that the code can execute in 0.228 seconds, or 4.34 Hz. While these speeds do not meet the frame capture rate of the Kinect sensor of 0.0333 seconds (30 Hz) nor the preliminary goal 0.0833 (12 Hz), they do show a significant improvement over the MATLAB code which executes with a time of 6.20 seconds (0.16 Hz) allowing for low speed data operation. A table outlining these improvements is shown in Table 2.
Table 2: Comparison of Matlab algorithm execution speed with converted C++ algorithm execution speed

<table>
<thead>
<tr>
<th>Road Defect #</th>
<th>Matlab Code Run Average (seconds) (3 Runs)</th>
<th>C++ Code Run Average (seconds) (100 Runs)</th>
<th>% Decrease</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.48</td>
<td>0.262</td>
<td>95.96</td>
</tr>
<tr>
<td>2</td>
<td>6.15</td>
<td>0.222</td>
<td>96.39</td>
</tr>
<tr>
<td>3</td>
<td>5.96</td>
<td>0.201</td>
<td>96.64</td>
</tr>
<tr>
<td>Mean</td>
<td>6.20</td>
<td>0.228</td>
<td>96.33</td>
</tr>
</tbody>
</table>

It was discussed in Section 4.3 that image resolution could be considered to also increase the speed of the algorithm. However, it is desired to disallow the reduction of image resolution in order to achieve the real-time vehicle travel speeds of 70 mph because this reduction would reduce the accuracy of the system. Several suggestions for increasing the overall image processing time are discussed in Section 7 to prevent the image resolution reduction and permit normal operation of the system at vehicle speeds of up to 70 mph.

6.2 Point Cloud Image Processing

The algorithm would be considered successful if the road defect(s) can be found in the image while disregarding noise and operating quickly. A total of 11 test scenarios were considered, and the road defects were successfully found in least 85.7% of all circumstances. A representative scenario is displayed in the following figures (34, 35, 36a, 36b, 37, 38a & 38b). Figure 34 is the actual color image of the road defect being processed and Figure 35 shows the 3D points returned by the Kinect sensor grouped by inliers, outliers above the road surface and outliers below the road surface. Ten more scenarios are illustrated in Appendix G.
Figure 34: Grant Ave. - Actual Color Image

Figure 35: Grant Ave. - Point Cloud
(a) Grant Ave. - Before Clustering          (b) Grant Ave. - After Clustering

Figure 36: Results of Clustering

Figure 37: Grant Ave. - Outliers Overlaid with Inliers
The elimination of points emanating from sensor and/or environmental noise is shown in Figures 36a & 36b. This refinement is the result of the clustering technique described in Section 4.2.2.2. The saliency map effectively distinguishes the road defect points (shown as the green points in Figure 37) from the road surface points or inliers (shown as the purple points in Figure 37) and displays the road defect as a grayscale image shown in Figure 38a. The threshold discussed in Section 4.2.2.2 is used to smooth the blurred, grayscale image produced by the saliency map yielding a binary, black-and-white image containing only zeros and ones. The maximum length and width axes are overlaid onto this black and white image using the process described in Section 4.2.2.2 and is illustrated by Figure 38b.

![Saliency Map](image1.png) ![Smoothed Image](image2.png)

(a) Grant Ave. - Saliency Map  (b) Grant Ave. - Smoothed Image

Figure 38: Defining the Outline of the Road Defect

Images from the actual road defect measurements are shown in Figure 39 along with the dimensions generated from the algorithm. The actual length and width of the road defect displayed are approximately 24" and 32", respectively. It was difficult to obtain actual measurements from the road defects as there are often soft edges that fall below the road surface prior to encountering the most obvious edge of the road defect. Because of the difficulty in manually determining the actual road defect edge, the actual length and width values are approximated, which is sufficient for this application. Even with the approximations in actual length and width, the algorithm is calculating the length and width from image processing and has proven it can achieve the accuracy goal of 30% that was determined as a reasonable accuracy for the purposes of categorizing the defects. The accuracy of 30% is considered reasonable because the length, width and depth measurements are used to determine volume and the volume, in turn, is used to categorize the road defect into classes or “bins” that are made up of a range of volumes. Therefore, unless the volume of a road defect falls near the boundary of one class with another, a relatively small inaccuracy will not change the class in
which the road defect falls. With more refinements and improvements to the dimension estimation section of the algorithm, the accuracy error could potentially be reduced down to 15%. These improvements in accuracy are aligned with the metric of “Percentage of Potholes Detected” identified as the most important quantifiable metric using the Quality Function Deployment (QFD) tool detailed in Section 3 of this document. This metric can also be further validated by on-road testing with known defects (ground truth data).

Figure 39: Actual measurements of road defects including: a) length, and b) width. Also shown in c), the dimensions generated by the algorithm including the classification assigned to the road defect.

A selection of additional road defects collected from the Morgantown, WV area roads are shown in Figure 40. The complete collection of image processing data from the road defects can be seen in Appendix E.
Figure 40: Additional examples of road surface defects and algorithm post-processing

The image processing results from all 11 scenarios are shown in Table 3 which compare the actual lengths and widths of the road defect with the lengths and widths obtained from the detection/classification algorithm.

Table 3: Comparison of maximum lengths and widths of defects obtained from manual measurements and the proposed method

<table>
<thead>
<tr>
<th>Defect Number</th>
<th>Manual Measurement</th>
<th>Proposed Method</th>
<th>Relative Error</th>
<th>Proposed Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Length (mm)</td>
<td>Width (mm)</td>
<td>Length (%)</td>
<td>Width (%)</td>
</tr>
<tr>
<td>1</td>
<td>21.00</td>
<td>32.00</td>
<td>19.37</td>
<td>36.69</td>
</tr>
<tr>
<td>2</td>
<td>21.00</td>
<td>15.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>28.00</td>
<td>32.00</td>
<td>19.60</td>
<td>24.70</td>
</tr>
<tr>
<td>4</td>
<td>16.00</td>
<td>15.00</td>
<td>14.45</td>
<td>22.22</td>
</tr>
<tr>
<td>5</td>
<td>15.00</td>
<td>11.00</td>
<td>25.80</td>
<td>27.38</td>
</tr>
<tr>
<td>6</td>
<td>27.00</td>
<td>29.00</td>
<td>16.95</td>
<td>31.45</td>
</tr>
<tr>
<td>7</td>
<td>14.00</td>
<td>11.00</td>
<td>13.76</td>
<td>19.25</td>
</tr>
<tr>
<td>8</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>9</td>
<td>0.00</td>
<td>0.00</td>
<td>22.20</td>
<td>21.22</td>
</tr>
<tr>
<td>10</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>11</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Sensor noise was present in all images, but is appropriately removed by four methods. First, border pixels that are known to be dominated by sensor noise are removed from consideration. Second, the clustering
routine presented in Section 4.2.2.2 for separating the image array into squares eliminates many noisy pixels that would have otherwise been included in a non-intelligent clustering scheme. Third, points remaining after the first two methods are completed are subjected to a saliency map which separates foreground pixels from background pixels further refining the shape of the road defect. Finally, the image is smoothed using a threshold for all remaining points in the image which finalizes the edges of the road defect and includes the red lines indicating the axes for the length and width of the road defect. The threshold was determined by means of a sensitivity analysis where all collected road defects were processed with the threshold being varied from a range of 0.2 to 0.5. The relative error of the length and width are compared to determine the optimal value of the threshold which proves to be a value 0.35.

Limitations of the Kinect sensor restrict use to specific weather conditions. Because of the IR projection that Kinect uses to return a point cloud, it is not suitable for use in direct sunlight or when bright reflections are present. Operation is also limited in inclement weather conditions as the infrared projection is distorted by the moisture in the air (for example, car exhaust on cold days can obscure the Kinect measurement). The Kinect’s field of view is limited, but works best when it is tilted 30° below the horizontal and toward the ground. This provides about a 4 x 3 meter usable field of view. Measurements beyond 4 meters are too noisy to include in processing. The Kinect sensor satisfies the Technical Metric of cost with an inexpensive price of $100. Despite the limitations of using the Kinect sensor, the results are sufficient to validate the proof of concept.

This novel approach to road defect detection and impact mitigation, which offers substantial improvements to existing systems, makes use of a 3D sensor and state-of-the-art point cloud processing algorithms. Of particular note, new algorithms were developed for road defect identification and classification that are uniquely suited for the problem at hand. The underlying theory and mathematics of this new algorithm was presented. Algorithm speed is crucial for this algorithm to operate in real-time. This algorithm was implemented in MATLAB and is capable of processing a single 3D point cloud in about 6.2 seconds. Improvements in speed were gained by the migration of the MATLAB code to compiled C++ code. The MATLAB code was successfully converted into C++ and decreased the post-processing time of the algorithm to 0.22 seconds. This allowed for testing of the algorithm to be completed in real-time at relatively low speeds of 15-25 mph. The desired processing speeds can be reached, allowing for real-time testing using average highway speeds of 70 mph, by several methods to be discussed in Section 7.

Implementation of the detection/classification algorithm will result in different technological obstacles. The algorithm will be faster since it will be programmed on a dedicated integrated circuit, but the collected data will be of dynamic driving situations which may cause different noise such as varying lighting conditions. In driving scenarios, the presence of other vehicles must be removed from the point cloud before processing.

6.3 Adaptive Suspension

6.3.1 Road Defect Simulation

The original controls strategy for controlling the actuator's piston travel was initially a open-loop system. This open-loop system requires a velocity for the actuator piston to travel both going upward and downward.
As mentioned in Section 5.1.1, when weights were stacked onto the bench testing platform, the piston did not travel the same distance up as it did going down when the air directional control valve was opened for the same amount of time. Therefore, the velocity used for upward travel is different than the velocity for downward travel. Unfortunately, when a signal is sent to the adaptive shock to alter the damping, these velocity had to be calibrated after each change which proved to be time consuming. Without the calibration, the actuator moves down the desired distance to simulate the road defect, but does not return to the original position. It would returns to a position either below or above the road surface and steadily maintains this error over the cycle lasting 30 seconds. This effect causes the appearance of the road surface to be inclining or declining. Calibrating the up/down velocity removes the error and produces a flat road surface with uniform road defects. There should not be an overshoot present when the tire actually falls into a road defect. Although, the current controls algorithm contains an overshoot, the calibration ensures that the overshoot is relatively small. An example of a uniform road defect simulation after calibration is shown in Figure 41.

Figure 41: Illustration of actual displacement (purple) and force (yellow) using a uniform simulation

The above figure shows the “floating” plate starting at 660 mm (or 25.98 in.) and then decreasing to 596 mm (or 23.46 in.). This is a drop of 2.52 in. which indicates an overshoot of 0.52 in. (or 26%) to the desired 2 in. drop defined by the uniform road defect simulation. This overshoot is slightly higher than desired for this experimentation but after the initial overshoot, the system levels out at 610 mm (or 24.01 in.) which is a 1.96 in. drop from the starting position and close to the desired 2 in. which is acceptable. The bottom portion of Figure 41 shows the forces acting on the adaptive shock during the simulation. The acting forces are consistent with what is expected. At equilibrium, the load cell shows a force of 187 kg, or 412.5 lbs, which is the amount of weight initially loaded on the bench testing platform to simulate the operational weight that the shock would normally be under. As the actuator compresses, the shock is pulled away from the load cell relieving some of the weight bearing down on the shock and the force sharply reduces. The tire reaches the bottom of the road defect and rebounds upward directing more force into the actuator as it is compressed. This increase in force is indicated by the sharp spike in force of up to 310 kg as shown in Figure 41.
In order to forego the time spent calibrating the velocity after each change in damping of the adaptive shock, a closed-loop, feedback system was developed which incorporates real-time distance measurements from the LDMS. This feedback system eliminates the need for the up/down velocity and prevents the displacement plots from steadily increasing or decreasing. However, this feedback system produces a larger overshoot than what is produced by the open-loop system. A plot illustrating the results from this feedback system is shown in Figure 42. While the closed-loop program retains the desired simulation of a tire striking a series of road defects, the figure shows a significant increase in the overshoot over the open-loop system from Figure 41. Because of the large overshoot, there is also an increase in maximum force spikes when the shock rebounds after striking the simulated road defect. These increased force spikes are illustrated in Figure 42 with values reaching as high as 475 kg (or 1047 lbs).

![Figure 42: Illustration of actual displacement (purple) and force (yellow) using a uniform simulation](image)

**6.3.2 Adaptive Shock Functionality**

One objective of building the bench testing platform was to determine if the damping of the adaptive shock could effectively be controlled by sending it varying voltages. To test this functionality, the adaptive shock is connected to a power supply and delivered a voltage. The shock is then fully compressed and the forces are measured using the compression load cell. This compression and measurement test is completed multiple times at the same applied voltage to allow the calculation of a mean force using the maximum peak force of each run. Next, the voltage being delivered to the adaptive shock is increased and the test was repeated. This procedure was completed 4 times using 4 different voltages: 0 volts, 1 volt, 2.5 volts and 5 volts. The results of this procedure are shown in Figure 43 and illustrated the multiple runs at different applied voltages to the shock.
Figure 43: A set of four subfigures showing the force acting on the shock given an applied voltage of: (a) zero volts; (b) one volt; (c) 2.5 volts; and, (d) 5 volts.

A table outlining the results from Figure 43 is shown in Table 4. The increase in the mean of peak forces corresponding to the increase in the applied voltage indicate the ability to effectively alter the damping of the adaptive shock.

Table 4: Mean peak forces for varying discrete shock values

<table>
<thead>
<tr>
<th>Shock Voltage (V)</th>
<th>Force (kg)</th>
<th>Run 1</th>
<th>Run 2</th>
<th>Run 3</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>17.75</td>
<td>17.75</td>
<td>17.75</td>
<td>17.75</td>
<td>17.75</td>
</tr>
<tr>
<td>1</td>
<td>18.86</td>
<td>17.75</td>
<td>17.75</td>
<td>17.75</td>
<td>18.12</td>
</tr>
<tr>
<td>2.5</td>
<td>18.86</td>
<td>19.96</td>
<td>18.86</td>
<td>19.23</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>19.96</td>
<td>23.3</td>
<td>23.3</td>
<td></td>
<td>22.19</td>
</tr>
</tbody>
</table>
The graph in Figure 44 shows the mean of the peak forces measured and a fitted trendline that suggests a linear relationship between the applied voltage to the adaptive shock and the resulting forces determined by the altered damping of the shock. The figure also displays the equation of the trendline in slope-intercept form.

![Figure 44: Comparison of shock voltage to force](image)

6.3.3 Uncertainty Analysis

One concern that arises from this plot is that there is a relatively small difference of force between an applied voltage of 0 V and an applied voltage of 5 V. There is only a 4.44 kg difference in force between the minimum setting and maximum setting. Because of this small difference, there is the possibility that the damping of the adaptive shock is not being altered and that only uncertainty in the data collection process suggests the increase in force with the changing applied voltages. Therefore an experimental uncertainty analysis is completed to find the expected uncertainty. This expected uncertainty is calculated by

\[ u_{\text{expected}} = \sqrt{\sum_{i=1}^{n} u_i^2} = \sqrt{(u_{\text{measurement}})^2 + (u_{\text{manufacturer}})^2} \]  

(18)

where \( u_i \) are elemental uncertainties such as precision uncertainties stemming from inaccurate measurement collection and bias uncertainties which are the result of sensor deficiencies. The measurement uncertainty is defined as

\[ u_{\text{measurement}} = \pm 2 \times \bar{x} \]  

(19)

where

\[ \bar{x} = \sqrt{\frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{k - 1}} \]  

(20)

The variable, \( \bar{x} \), is the mean of the peak forces measured from each the three runs performed for each voltage setting and \( k \) is the number of runs. The manufacturer’s uncertainty, \( u_{\text{manufacturer}} \), is the repeatability of the sensor being used, in this case, the compression load cell. The repeatability value is obtained from the sensor specifications sheet. For this load cell, the repeatability is listed as being 0.089% of the full scale, which is 1000 kg for this particular sensor. Therefore, the \( u_{\text{manufacturer}} \) is calculated to be 0.89 kg. Table
5 outlines the standard deviations, measurement and manufacturer uncertainties and the final expected uncertainty.

Table 5: Calculations of the standard deviation and expected uncertainty

<table>
<thead>
<tr>
<th>Shock Voltage (V)</th>
<th>Force (kg) Run Average</th>
<th>$\sum_{t=1}^{n} (x - \bar{x})^2$</th>
<th>Std. Deviation $\sigma_5$</th>
<th>Measurement Uncertainty $u_{Measurement}$</th>
<th>Manufacturer Uncertainty $u_{Manufacturer}$</th>
<th>Expected Uncertainty $u_{Expected}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>17.75</td>
<td>17.75</td>
<td>0.000</td>
<td>$\pm 0.60$</td>
<td>$\pm 0.89$</td>
<td>$\pm 0.89$</td>
</tr>
<tr>
<td>1</td>
<td>18.12</td>
<td>19.12</td>
<td>0.641</td>
<td>$\pm 1.28$</td>
<td>$\pm 0.89$</td>
<td>$\pm 1.56$</td>
</tr>
<tr>
<td>2.5</td>
<td>19.23</td>
<td>21.73</td>
<td>0.635</td>
<td>$\pm 1.27$</td>
<td>$\pm 0.89$</td>
<td>$\pm 1.55$</td>
</tr>
<tr>
<td>5</td>
<td>22.19</td>
<td>27.19</td>
<td>1.928</td>
<td>$\pm 3.86$</td>
<td>$\pm 0.89$</td>
<td>$\pm 3.96$</td>
</tr>
</tbody>
</table>

According to the results from Table 5, the values of the measurement uncertainties outweigh the manufacturer uncertainty and dominate the error produced from the performed test. If the expected uncertainty, $\pm u_{expected}$, is added to the force values from the runs performed during testing, the values could end up migrating to ranges which might indicate a different voltage setting and therefore a different damping setting than what is expected. This overlap can be seen using error bars as shown in Figure 45.

Figure 45: Expected uncertainty at various applied voltages

Because of the uncertainty of the values resulting from the current method described to alter the damping of the adaptive shock, it is not rational to attempt to use only the force measurements in the determination of next proposed test, response time of the adaptive shock. Therefore, a new method for testing the response time of the system will have to be developed.

7 Conclusion

The objective of this thesis was to explore a system for road defect detection/classification and adaptive suspension control as a means to improve vehicle performance, safety, and ride quality. Although progress was made on all aspects of this system, the technical focus was on the road defect detection and bench testing portions of this problem.
The proposed concept was tested in the field using 3D point clouds collected from a Microsoft Kinect mounted on the tailgate of a pick-up truck. Data was collected for a variety of road defects on roadways near Morgantown, WV. The algorithm effectively eliminates noise from the 3D point cloud obtained from the Kinect sensor, converts the data from the sensor frame to the tire frame, defines the nominal road surface and categorizes the 3D points (inliers, outliers above the road surface or outliers below the road surface). The algorithm is capable of finding the road defects in 85.7% of every image tested and results are provided for all representative cases. These results indicate that the approach is capable of detecting road hazards of arbitrary shape and size, however, multiple road defects in a single frame can tend to skew the length and width measurements calculated by the algorithm. Consequently, great strides were made towards meeting the two main technical metrics: (1) percentage of potholes detected and (2) algorithm speed. The conversion of the Matlab code to C++ increased the processing time from 6.2 seconds to 0.22 seconds. While this improvement in algorithm speed was significant, it still did not achieve the goal of the minimum algorithm processing speed of 0.0833 seconds (12 Hz). With the current development and testing completed, the system has a high probability to reach the desired algorithm speed of about 12 Hz for commercial applications after additional refinements and improvements. While additional tests are required, all results to date indicate that the algorithm is capable of detecting most road hazards.

In lieu of incorporating an adaptive suspension into a test vehicle, a bench testing platform was fabricated and developed so that it may be used to validate the functionality of the adaptive shock and determine the response time of the shock. It was placed permanently in a testing area and can potentially be used to test multiple types of suspension shocks. A compression load cell, laser distance measurement sensor and accelerometer were all incorporated into the bench testing platform and it was confirmed that their signals could be accurately read and transmitted to a laptop computer. A controls algorithm was written and developed in Simulink to actuate a pneumatic cylinder which is connected to the adaptive shock. This actuator compresses and extends in a pattern simulating a tire impacting a series of road defects. Using these simulations, tests were performed to determine that the damping of the adaptive shock could, in fact, be altered by sending electrical signals to the shock from the Woodward controller via the CAN bus. Data from these tests suggest that there is a linear relationship between the voltage sent to the adaptive shock and the force required to fully compress the shock. However, after performing an uncertainty analysis, it was determined that the relationship between the applied voltage and force could not be unequivocally confirmed using the devised method. Because the test to determine the response time of the shock depended on using the results of the shock functionality test, and because the results were not conclusive, an alternative method for testing the response time of the adaptive shock will need to be developed.

7.1 Future Work

While this thesis focused on developing an algorithm and bench testing platform to validate a proof of concept for a road surface defect detection and classification system, there are a number of ways the system can be refined and improved to move closer to a marketable product. First, the algorithm will need to increase the execution speed to function in a real-time environment. Second, alternative 3D sensors should be considered that are better suited for taking dynamic images, operation in different weather conditions, and increased detection range. Third, controls for the bench testing platform would need to be refined to help validate the accurate operation of the system. Finally, control algorithms coupled with vehicle suspension components would be integrated into a test vehicle to test actual on-road operation of the complete systems.
The primary task remaining for improving the data capture system is making minor refinements and performing further testing to improve the stability of the data collection program. The data capture rate must eventually be tested running in a continuous fashion, for a long period of time, to effectively simulate a realistic collection environment. Any issues or bugs must be fixed before long-term, on-board vehicle execution. In regards to the post-processing algorithm, the major task remaining is to further increase the speed of the detection and classification algorithm. There are a number of ways the algorithm speed can be increased including:

1. **Utilizing specialized image processors**: Especially expensive computation may also be moved to a Graphics Processing Unit (GPU) or a Field Programmable Gate Array (FPGA) — both commonly used for image processing applications where parallelization may offer speed advantages.

2. **Use information from previous frames**: Another major portion of the algorithm which is computationally expensive is the point cloud clustering described in Section 4.2.2.1. This section is time consuming because of the sheer number of points contained in a point cloud obtained by the Kinect sensor. The resolution of the point cloud can be reduced to reduce the number of points being processed, but this would also sacrifice the accuracy of the sensor. Instead, the number of points being processed can be strategically reduced by using information about the road defect from a previous frame or image to narrow the search area of the current frame. This approach would ensure that the road defect will accurately be identified in all frames in a real-time environment and also increase the speed of the algorithm.

3. **Incorporate information already being transmitted throughout the vehicle**: Vehicles already contain sensors measuring the angle of the steering wheel and, therefore, the tire angle. These measurements are already being transmitted throughout the vehicle via a CAN bus and are used in other subsystems within the vehicle to control operation of the vehicle. These signals could be intercepted and utilized by the algorithm reducing the amount of calculations the post-processing algorithm must execute thereby increase the speed of the algorithm.

The bench testing platform will need to be tested to determine if there is enough resolution in the discrete values, which the algorithm assigns to the road defect based on volume. By increasing the number of defect size categories represented by these discrete values and corresponding to suspension adjustment levels, ride quality might improve, but defect measurement accuracy requirements will be more stringent. The testing platform is used to determine if the adaptive shock is indeed altering the damping ratio based on the classifications (0-5) that the algorithm assigns to the road defect. Additional studies will need to be performed to determine if these discrete values are appropriate for altering the damping of the adaptive shock. The question remains regarding what discrete value is most effective to mitigate the impact of the tire with the road defect. For instance, it is possible that there should be 3 settings of damping on the shock. First, a medium setting could be used for normal operating conditions. Second, a fully stiffened setting for smaller road defects allowing the tire to essentially “skip” over the defect. Third, a fully loosened setting for scenarios with extremely large road defect in which it would be unlikely that the tire could “skip” over. This is just one consideration for refinements to assigning discrete damping settings to various classes of road defects as classified by the proposed algorithm. With additional on-road tests, it would be beneficial to also incorporate the vehicle velocity when considering the damping settings assignments. At much lower speeds, a tire
impacting a smaller road defect will less likely “skip” over the defect and be more susceptible to striking the defect harshly. Therefore, the vehicle velocity should be considered in these assignments. Because the current system utilizes a CAN bus for transmission of data, the vehicle velocity signal (already being transmitted for use by other vehicle components and systems) could easily be intercepted by the system and incorporated.

To operate the suspension system in real-time, the bench testing platform will continue to be operated electronically, using the Woodward controller. Additionally, the ideal solution is operating the bench testing platform over the CAN bus, with the CAN messages detailing the timing increments in which the solenoid will open and close, effectively controlling the operation of the pneumatic cylinder. The CAN bus is a widely used standard in the automotive industry which allows communication between Electronic Control Units (ECUs) in a system devoid of a host computer. This will start as simple testing, as additional research and tests need to be conducted relating to the Woodward controller’s CAN protocol, such that a program can be created for directly operating the suspension system over the CAN bus.

The strategy for determining damping settings could also be determined using a genetic algorithm or neural network. Using data from accelerometers placed near the driver of the vehicle, the movement of the driver could be monitored and the system could learn the optimal damping setting necessary when subjected to a variety of vehicle velocities and varying dimensions of road defects. The optimal damping setting would be defined by the combination of damping setting, vehicle velocity and road defect which causes the least amount of movement to the driver or the smallest amount of acceleration endured by the driver.

Improvements in detection range and operation in more strenuous weather conditions could be achieved with a higher quality sensor, albeit also more expensive. There are several viable options for a higher quality sensor available in today’s market. A 3D LiDAR sensor, such as the M8 made by Quanergy or the HDL-64E made by Velodyne, can be found for under $5,000. Such a sensor has much higher range capabilities than the Kinect sensor, reaching to 200 meters. Sensor such as LiDAR are also not as susceptible to noise resulting from various lighting conditions such as direct sunlight as is the case of the Kinect sensor. They are also more adept to handle situations in which shallow road defects have been filled with rain water. Autonomous vehicles commonly incorporate a LiDAR sensor, so defect detection may be efficiently included in the technology development of autonomous vehicles effectively sharing the expense of the LiDAR sensor. Also, the price of LiDAR sensors will decrease as the growing market for autonomous vehicles will create more demand.

Not only can this technology be used as an impact mitigation system responsible for improving ride quality of vehicles and safety of the consumer, it can also be employed by government agencies such as the Department of Transportation by having locations and parameters of road surface defects collected and saved to databases for the purpose of maintaining and repairing highway systems. With the development of autonomous vehicles and vehicle-to-vehicle communication technology on the forefront of the automotive industry, the information collected regarding the road surface defects can be shared with other vehicles in-network and combined with GPS systems to propose alternative, safer routes for other vehicles on the highways. In autonomous vehicles, the vehicle could also pre-emptively avoid the surface road defect, by steering around the defect, with the knowledge of the defects location and size.
References


[8] Ford Motor Company,”All-new ford fusion v6 sport helps protect against potholes, enhancing comfort and handling”, February 2016


Appendix A - Woodward controller pin-out table for connector A

<table>
<thead>
<tr>
<th>Pin #</th>
<th>ControlCore Resource Name</th>
<th>Function Name</th>
<th>Notes</th>
<th>Wire Number Color Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>LSO10</td>
<td>Low Side Out 10</td>
<td>3 A</td>
<td>1 Pink/Light Blue</td>
</tr>
<tr>
<td>A2</td>
<td>LSO1</td>
<td>Low Side Out 1</td>
<td>2.5 A, Current Sense, Diode</td>
<td>2 Pink/Orange</td>
</tr>
<tr>
<td>A3</td>
<td>AN15_DG7</td>
<td>Analog/Discrete In</td>
<td>10K to Vcc</td>
<td>3 Yellow/Black</td>
</tr>
<tr>
<td>A4</td>
<td>LSO8</td>
<td>Low Side Out 8</td>
<td>3 A</td>
<td>4 White</td>
</tr>
<tr>
<td>A5</td>
<td>LSO6</td>
<td>Low Side Out 6</td>
<td>2.5 A, Current Sense, Diode</td>
<td>5 White/Dark Blue</td>
</tr>
<tr>
<td>A6</td>
<td>AN14_DG6</td>
<td>Analog/Discrete In</td>
<td>10K to Vcc</td>
<td>6 Black/Red</td>
</tr>
<tr>
<td>A7</td>
<td>LSO3</td>
<td>Low Side Out 3</td>
<td>2.5 A, Current Sense, Diode</td>
<td>7 Yellow/Orange</td>
</tr>
<tr>
<td>A8</td>
<td>LSO4</td>
<td>Low Side Out 4</td>
<td>2.5 A, Current Sense, Diode</td>
<td>8 Light Blue</td>
</tr>
<tr>
<td>A9</td>
<td>AN7</td>
<td>Analog In 7</td>
<td>10K to Vcc</td>
<td>9 Tan/Light Blue</td>
</tr>
<tr>
<td>A10</td>
<td>AN5</td>
<td>Analog In 5</td>
<td>10K to Vcc</td>
<td>10 Gray</td>
</tr>
<tr>
<td>A11</td>
<td>AN6</td>
<td>Analog In 6</td>
<td>10K to Vcc</td>
<td>11 Dark Blue</td>
</tr>
<tr>
<td>A12</td>
<td>AN8</td>
<td>Analog In 8</td>
<td>10K to Vcc</td>
<td>12 Dark Blue/White</td>
</tr>
<tr>
<td>A13</td>
<td>LSO2</td>
<td>Low Side Out 2</td>
<td>2.5 A, Current Sense, Diode</td>
<td>13 White/Light Blue</td>
</tr>
<tr>
<td>A14</td>
<td>LSO5</td>
<td>Low Side Out 5</td>
<td>2.5 A, Current Sense, Diode</td>
<td>14 White/Black</td>
</tr>
<tr>
<td>A15</td>
<td>AN13_DG5</td>
<td>Analog/Discrete In</td>
<td>51.1K to GND</td>
<td>15 Black/Yellow</td>
</tr>
<tr>
<td>A16</td>
<td>GND</td>
<td>System Ground</td>
<td>Battery Ground</td>
<td>16 Black/White</td>
</tr>
<tr>
<td>A17</td>
<td>N/C</td>
<td>No Internal Connection</td>
<td>No Internal Connection</td>
<td></td>
</tr>
<tr>
<td>A18</td>
<td>LSO9</td>
<td>Low Side Out 9</td>
<td>3 A</td>
<td>18 Pink/Brown</td>
</tr>
<tr>
<td>A19</td>
<td>AN16_DG8</td>
<td>Analog/Discrete In</td>
<td>10K to Vcc</td>
<td>19 Orange</td>
</tr>
<tr>
<td>A20</td>
<td>DRV/P</td>
<td>Driver Power</td>
<td>Wire via MPR contacts</td>
<td>20 Orange/White</td>
</tr>
<tr>
<td>A21</td>
<td>DRV/P</td>
<td>Driver Power</td>
<td>Wire via MPR contacts</td>
<td>21 Black/Blue</td>
</tr>
<tr>
<td>A22</td>
<td>MPRD</td>
<td>Main Power Relay Driver</td>
<td>1.0 A max.</td>
<td>22 Yellow/Purple</td>
</tr>
<tr>
<td>A23</td>
<td>LSO7/EasyLink</td>
<td>Low Side Out 7</td>
<td>1 A</td>
<td>23 Red/Blue</td>
</tr>
<tr>
<td>A24</td>
<td>LSO11</td>
<td>Low Side Out 11</td>
<td>3 A</td>
<td>24 Black/White</td>
</tr>
</tbody>
</table>
Appendix B  - Woodward controller pin-out table for connector B

<table>
<thead>
<tr>
<th>Pin #</th>
<th>ControlCore Resource Name</th>
<th>Function Name</th>
<th>Notes</th>
<th>Wire Number Color Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ground</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B1</td>
<td>XDRG</td>
<td>Transducer Ground</td>
<td>Low Current Return</td>
<td>25 Green /White</td>
</tr>
<tr>
<td>B2</td>
<td>AN1</td>
<td>Analog In 1</td>
<td>100 W to Vcc via Diode</td>
<td>26 Tan</td>
</tr>
<tr>
<td>B3</td>
<td>AN2</td>
<td>Analog In 2</td>
<td>220K Pull Down, $\tau = 1$ ms</td>
<td>27 Yellow</td>
</tr>
<tr>
<td>B4</td>
<td>VR1-</td>
<td>Return VR Input</td>
<td>Leave open for Hall</td>
<td>28 Dark Blue/Pink</td>
</tr>
<tr>
<td>B5</td>
<td>VR1+</td>
<td>VR Sensor Positive</td>
<td>VR, Hall, Discrete</td>
<td>29 Red/Pink</td>
</tr>
<tr>
<td>B6</td>
<td>CAN2-</td>
<td>CAN2 Negative</td>
<td></td>
<td>30 White</td>
</tr>
<tr>
<td>B7</td>
<td>CAN2+</td>
<td>CAN2 Positive</td>
<td></td>
<td>31 White/Green</td>
</tr>
<tr>
<td>B8</td>
<td>KEY_SW</td>
<td>Key Switch</td>
<td>Module Wake</td>
<td>32 Pink/Blue</td>
</tr>
<tr>
<td>Wake</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B9</td>
<td>AN9</td>
<td>Analog In 9</td>
<td>51.1K Pull Up, $\tau = 1$ ms</td>
<td>33 Gray/Red</td>
</tr>
<tr>
<td>B10</td>
<td>AN3</td>
<td>Analog In 3</td>
<td>1K to Vcc, $\tau = 1$ ms</td>
<td>34 Orange/Black</td>
</tr>
<tr>
<td>B11</td>
<td>AN4</td>
<td>Analog In 4</td>
<td>220K Pull Down, $\tau = 1$ ms</td>
<td>35 Blue/Black</td>
</tr>
<tr>
<td>B12</td>
<td>AN10</td>
<td>Analog In 10</td>
<td>51.1K Pull Up, $\tau = 1$ ms</td>
<td>36 White/Orange</td>
</tr>
<tr>
<td>B13</td>
<td>NC</td>
<td>No Connection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B14</td>
<td>NC</td>
<td>No Connection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B15</td>
<td>VR2-</td>
<td>Return VR Input</td>
<td>Leave open for Hall</td>
<td>37 White/Yellow</td>
</tr>
<tr>
<td>B16</td>
<td>VR2+</td>
<td>VR Sensor Positive</td>
<td>VR, Hall, Discrete</td>
<td>38 Tan/Green</td>
</tr>
<tr>
<td>B17</td>
<td>GND</td>
<td>System Ground</td>
<td>Battery Ground</td>
<td>39 Green /Black</td>
</tr>
<tr>
<td>Ground</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B18</td>
<td>AN11_DG3</td>
<td>Analog/Discrete in</td>
<td>51.1K to GND</td>
<td>40 Green/Red</td>
</tr>
<tr>
<td>B19</td>
<td>AN12_DG4</td>
<td>Analog/Discrete in</td>
<td>51.1K to GND</td>
<td>41 Black/Green</td>
</tr>
<tr>
<td>CAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B20</td>
<td>CAN1+</td>
<td>CAN1 Positive</td>
<td></td>
<td>42 Purple</td>
</tr>
<tr>
<td>B21</td>
<td>CAN1-</td>
<td>CAN1 Negative</td>
<td></td>
<td>43 Tan/Purple</td>
</tr>
<tr>
<td>Power</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B22</td>
<td>BATT_IN</td>
<td>Battery Connection</td>
<td>Module Power</td>
<td>44 Red /White</td>
</tr>
<tr>
<td>B23</td>
<td>STOP_DIG_AN</td>
<td>E-STOP Input</td>
<td>100 W to Vcc via Diode, 16.5K to GND</td>
<td>45 Purple/Yellow</td>
</tr>
<tr>
<td>B24</td>
<td>XDRP</td>
<td>Transducer Power</td>
<td>300 mA</td>
<td>46 Black/ Orange</td>
</tr>
</tbody>
</table>
Appendix C - Simulink model to control the Bench Testing Platform

Figure 46: Top level control triggering the function

Figure 47: The real-time function broken into sensor and series of road defects (illustrates the feedback from the sensors)
Figure 48: Sensor functionality

Figure 49: Switch between rocker switch operation and series of road defect simulation
Figure 50: Triggers for the rocker switch

Figure 51: Rocker switch operation (up)

Figure 52: Rocker switch operation (down)
Figure 53: Simulation of a series of road defects
Appendix D  - Operational Amplifier for Compression Load Cell

Figure 54: Operation amplifier combining a summing amplifier with an inverting amplifier
Appendix E - MATLAB code of Detection/Classification Algorithm

% Andrew Rhodes, Shane Haught
% December 2016, Revised December 2017, Revised February 2018
%
% Estimate the surface normal vector and orientation of a point cloud
% Assuming: point cloud is a plane
% road and pothole contribute to the estimation
% road plane passes through the tire contact point

%%%%% USING KINECT: Code must include ~isnan/GoodData on most operations.
%%%%% Also need PlaneFitSVD.m and Transform_Init.m functions.
close all
clear
clc

% Load the point cloud data from the excel file
% Data is in the camera frame
% PotholeStatic_10-5-16/SRImage_14.38.227.csv
% PointCloudData = importdata('.../PointCloudImages/SwissRanger/PotholeStaticConcrete_10-5-
% 16/SRImage_16.11.471.csv'); %SRImage
PointCloudData = importdata('frame_5.mat');
PointCloudData(:,1) = PointCloudData.Location(:,1);
PointCloudData(:,2) = PointCloudData.Location(:,2);
PointCloudData(:,3) = PointCloudData.Location(:,3);
PointCloudData = rot90(PointCloudData);
ColorImageData = importdata('framecolor_5.mat');
ColorImage = ColorImageData.Color;
figure
imshow(ColorImage)

ImageHeight = 512;
ImageWidth = 424;
ImageSize = ImageHeight * ImageWidth;

% Clean up the data!

% Gaussian blur the range image
pointCloudData = reshape(PointCloudData, [1, 3];
for ii = 1: length(pointCloudData)
Range(ii, 1) = norm(pointCloudData(ii,:));
end

RangeImage = reshape(Range, ImageHeight, ImageWidth);

% Find where images successfully captured data
GoodData = ~isnan(RangeImage);

GaussFilter = fspecial('gaussian', 5, 1);
RangeImageFilt = imfilter(RangeImage, GaussFilter, 'replicate');
RangeImageFiltVec = reshape(RangeImageFilt, ImageSize, 1);
Scales = RangeImageFiltVec .* Range(:,1);

PointCloudDataScaled(1:ImageSize,1:3) = bsxfun(@times, Scales, PointCloudData(:,1:3));

% [Rotation, Translation, alpha, beta] = Transform_Init(PointCloudDataScaled, 
% [ImageHeight, ImageWidth]);
Alpha = alpha;
Beta = beta;

% Presets

62
% Min Pothole size (square in meters)
MinPotholeSize = 0.05;

% Average Width of 'Pixel'
AveDiff = sqrt(sum(diff(PointCloudDataScaled(GoodData)).^2,2));
q = quantile(AveDiff, [0.25,0.5,0.75]);
AveDiff[AveDiff> q(3) + 3*(q(3)-q(1))] = [];% Significant
PixelSize =
round(mean([median(AveDiff(~isnan(AveDiff))), mean(AveDiff(~isnan(AveDiff))))]),1,
'significant');

% Bounding Box (in pixels)
BoxSize = round(MinPotholeSize / PixelSize);

% Percentage that box is full
BoxFillPerc = 0.25;

% Number of outliers necessary in a box
MinNumBoxOutliers = floor(BoxFillPerc*BoxSize*BoxSize);

% Separate Image into blocks
k = 1;
c = 1;

mat = zeros(ImageHeight, ImageWidth);
for i = 1 : ImageWidth
  for j = 1 : ImageHeight
    mat(j,i) = k;
    if mod(j,BoxSize) == 0 && j ~= ImageHeight
      k = k+1;
    end
    if mod(i,BoxSize) == 0
      k = k +1;
      c = k;
    else
      k = c;
    end
  end
end

Blocks = reshape(mat,[],1);
Blocks(end) = [];

%% Transform Point Cloud to Tire Frame
%% Assume knowledge of transformation from camera frame to tire frame
%% Then Location_tire = R * Location_cam + t

%% Rough estimate of transformation from camera frame to tire frame for our
%% first set of measurement data
ang_x = 75;
ang_y = 180;
ang_z = 0;

Rz = [cosd(ang_z), -sind(ang_z), 0; sind(ang_z), cosd(ang_z), 0; 0,0,1];
Ry = [cosd(ang_y), 0, sind(ang_y); 0, 1, 0; -sind(ang_y), 0, cosd(ang_y)];
Rx = [1,0,0; 0, cosd(ang_x), -sind(ang_x); 0, sind(ang_x), cosd(ang_x)];
t = [0, 0, 0];

PointCloudTire = bsxfun(@plus, (Rotation * PointCloudDataScaled(~isnan(PointCloudDataScaled(:,1)),1:3)'), Translation);
%% Plot Rotation from Camera Frame to Tire Frame
figure
plot3(PointCloudDataScaled(:,1), PointCloudDataScaled(:,2), PointCloudDataScaled(:,3), 'r.')
hold on
plot3(PointCloudTire(:,1), PointCloudTire(:,2), PointCloudTire(:,3), 'b.')
plot3(0,0,0,'*')
hold off
xlabel('x')
ylabel('y')
ylabel('z')
view([75,20])
legend('Sensor Frame', 'Tire Frame', 'Origin', 'Location', 'NorthWest')
title('Rotation from Camera Frame to Tire Frame')
grid on

%% Organize Point Cloud Data

% Find surface normal of every points
PointCloudTire_ptc = pointCloud(PointCloudTire);
NormalsTire = pcnormals(PointCloudTire_ptc, 50);

% Combine data in structure
PointCloud.Location = PointCloudTire;
PointCloud.Normal = NormalsTire;
PointCloud.Count = length(PointCloud.Location);
PointCloud.Centroid = mean(PointCloud.Location);
PointCloud.Block = Blocks;

% Orient all normals in the same direction
NormalVectorOriginal = PlaneFitSVD(PointCloud.Location);
NormalVectorOriginal = [0 0 1]';
NormalsUp = NormalsTire * NormalVectorOriginal;
PointCloud Normal(NormalsUp < 0, :) = PointCloud.Normal(NormalsUp < 0, :);

%% Entire Point cloud with the original average surface normal.
figure
quiver3(PointCloud.Location(:,1), PointCloud.Location(:,2), PointCloud.Location(:,3), PointCloud.Normal(:,1), PointCloud.Normal(:,2), PointCloud.Normal(:,3))
hold on
quiver3(PointCloud.Centroid(1), PointCloud.Centroid(2), PointCloud.Centroid(3), NormalVectorOriginal(1), NormalVectorOriginal(2), NormalVectorOriginal(3))
hold off
title('Entire Point Cloud w/Original Average Surface Normal')
legend('All Normals', 'Average Surface Normal', 'Location', 'NorthEast')
xlabel('x')
ylabel('y')
zlabel('z')

%% Determine Outliers
%% Iterative Procedure

ContinueIter = 1;
Iter = 0;
MaxIter = 20;

AngleTol = 0.85;
DistTol = 0.0030;
nDist = 6;

%% Define Inlier / Outlier Bounds
mUpperBound = (nDist*DistTol - DistTol) / (1-AngleTol);
bUpperBound = DistTol - mUpperBound * AngleTol;

mLowerBound = (DistTol - nDist*DistTol) / (1-AngleTol);
bLowerBound = -DistTol - mLowerBound * AngleTol;

Inliers = true(PointCloud.Count, 1);
NumInliersOld = sum(Inliers);

while ContinueIter

    RoadSurface.Location = PointCloud.Location(Inliers, :);
    RoadSurface.Normal = PointCloud.Normal(Inliers, :);
    RoadSurface.Count = length(RoadSurface.Location);
    RoadSurface.Centroid = sum(RoadSurface.Location, 1) / RoadSurface.Count;

    [NormalVectorNew, alpha, beta] = PlaneFitSVD(RoadSurface.Location);

    NormalAlignment = PointCloud.Normal * NormalVectorNew;
    DistToPlaneCentroid = bsxfun(@minus, PointCloud.Location, RoadSurface.Centroid) * NormalVectorNew;

    % Dual Region - Rectangular and Triangular
    IdxOutliersAbove = (NormalAlignment < AngleTol & DistToPlaneCentroid >= DistTol)
        | (NormalAlignment >= AngleTol & (DistToPlaneCentroid >= mUpperBound * NormalAlignment + bUpperBound));
    IdxOutliersBelow = (NormalAlignment < AngleTol & DistToPlaneCentroid <= -DistTol)
        | (NormalAlignment >= AngleTol & (DistToPlaneCentroid <= mLowerBound * NormalAlignment + bLowerBound));
    IdxOutliers = IdxOutliersAbove | IdxOutliersBelow;

    % Opposite of Outliers
    Inliers = ~IdxOutliers;
    NumInliersNew = sum(Inliers);

    Iter = Iter + 1;

    if (Iter >= MaxIter) || abs(NumInliersNew - NumInliersOld) == 0 %||
        acosd(NormalVectorOld'*NormalVectorNew) < 1e-10
        ContinueIter = 0;
    end

    % Scatter plot of inliers and outliers
    figure
    plot(NormalAlignment(Inliers), DistToPlaneCentroid(Inliers), 'b.'
    hold on
    plot(NormalAlignment(IdxOutliersAbove),
         DistToPlaneCentroid(IdxOutliersAbove), 'r.'
    plot(NormalAlignment(IdxOutliersBelow),
         DistToPlaneCentroid(IdxOutliersBelow), 'g.'
    plot([0,1], [0,0], 'k-'
    plot([0, AngleTol, 1], [DistTol, DistTol, nDist*DistTol], 'k--'
    plot([0, AngleTol, 1], [-DistTol, -DistTol, -nDist*DistTol], 'k--'
    legend('Inliers', 'Outliers Above', 'Outliers Below', 'Zero Line', 'Upper Bound', 'Lower Bound', 'Location', 'South')
    else
        NormalVectorOld = NormalVectorNew;
        NumInliersOld = NumInliersNew;
    end
end

% title('Alignment Distance Graph')
% xlabel('Normal Vector Alignment')
% ylabel('Distance To Plane')
% Plot 3D Inliers, Outliers Above and Outliers Below
figure
plot3(PointCloud.Location(~IdxOutliers,1), PointCloud.Location(~IdxOutliers,2),
      PointCloud.Location(~IdxOutliers,3), 'b.'), hold on
plot3(PointCloud.Location(IdxOutliersBelow,1),
      PointCloud.Location(IdxOutliersBelow,2),
      PointCloud.Location(IdxOutliersBelow,3), 'g.'),
plot3(PointCloud.Location(IdxOutliersAbove,1),
      PointCloud.Location(IdxOutliersAbove,2),
      PointCloud.Location(IdxOutliersAbove,3), 'r.'), hold off
title('Outlier Points')
legend('Inliers','Outliers Below','Outliers Above','Location','NorthEast')

% Creating 2D images from 3D point clouds

% Show 2-D image of inliers (converted from 3D point cloud)
x = PointCloud.Location(~IdxOutliers,1);
y = PointCloud.Location(~IdxOutliers,2);
z = PointCloud.Location(~IdxOutliers,3);
[I,xr,yr,cls,ind] = pointcloud2image(x,y,z,ImageHeight,ImageWidth);
I = rot90(I,3);
I = imresize(I,[ImageHeight ImageWidth]);
x_max = xr;
x_min = xl;
y_max = yr;
y_min = yl;

% Show 2-D image of Outliers Below (converted from 3D point cloud)
x = PointCloud.Location(IdxOutliersBelow,1);
y = PointCloud.Location(IdxOutliersBelow,2);
z = PointCloud.Location(IdxOutliersBelow,3);
[I_below,xl,xr,yl,yr,cls,ind] = pointcloud2image(x,y,z,ImageHeight,ImageWidth);
I_below = rot90(I_below,3);
I_below = imresize(I_below, [ImageHeight ImageWidth]);

% % Determine the max and min of the x and y for the Inlier points and
% % Outliers Below (these values are needed when finding the depth later)
% if xr>x_max
% x_max = xr;
% else
% end
% if xl<x_min
% x_min = xl;
% else
% end
% if yr>y_max
% y_max = yr;
% else
% end
% if yl<y_min
% y_min = yl;
% else
% end

figure
% imshow(I)
% title('2D Image of Inliers')
% figure
% imshow(I_below)
% title('2D Image of Outliers Below')

thresh_high=0.6;
thresh_low = 0.1;
index=find(I>thresh_high);
I(index)=1;
index=find(I<thresh_low);
I(index)=1;
index=find(I_below>thresh_high);
I_below(index)=1;
index=find(I_below<thresh_low);
I_below(index)=1;

% Changing pixels that are the same in the two images to white
index1 = find(I==1);
index2 = find(I_below==1);
same = intersect(index1, index2);
I(same) = 1;
I_below(same) = 1;

% Finding pixels of color in below image that corresponds to white pixels
% of inliers
index1 = find(I_below==1);
index2 = find(I==1);
same = intersect(index1, index2);
I_below(same) = 0;

% Finding pixels of color in inlier image that corresponds to white pixels
% of outliers below
index1 = find(I==1);
index2 = find(I_below==1);
same = intersect(index1, index2);
I(same) = 0;

figure
imshow(I_below)

% Finding pixels of color in below image that corresponds to white pixels
% of outliers below

figure
% imshowpair(I,I_below)
% title('2D Image of Inlier overlaid on Outliers Below before clustering')

%% Clustering to compare inliers to outliers below
div = 30;
rows = round(ImageHeight/div);
cols = round(ImageWidth/div);

index1=0;
index2=0;
threshold = 0.85;
for j=1:div
    if j==1
        for i=1:div
            if i==1
                index1=find(I(1:(j*rows),1:(i*cols))==1);
                index2=find(I_below(1:(j*rows),1:(i*cols))==1);
                A = I_below(1:(j*rows),1:(i*cols));
                if isempty(index1) && isempty(index2)
                    else

if length(index1)>length(index2) || (length(index1)/length(index2))>threshold  % If there are more point in the surface than below
A(index2) = 1;
I_below(1:(j*rows),1:(i*cols))=A;
else
end
endif i==div
index1=find(I(1:(j*rows),((i-1)*cols)+1:ImageWidth)==1);
index2=find(I_below(1:(j*rows),((i-1)*cols)+1:ImageWidth)==1);
A = I_below(1:(j*rows),((i-1)*cols)+1:ImageWidth);
if isempty(index1) && isempty(index2)
else
if length(index1)>length(index2) || (length(index1)/length(index2))>threshold
A(index2) = 1;
I_below(1:(j*rows),((i-1)*cols)+1:Width)=A;
else
end
endif else
index1=find(I(1:(j*rows),((i-1)*cols)+1:(i*cols))==1);
index2=find(I_below(1:(j*rows),((i-1)*cols)+1:(i*cols))==1);
A = I_below(1:(j*rows),((i-1)*cols)+1:(i*cols));
if isempty(index1) && isempty(index2)
else
if length(index1)>length(index2) || (length(index1)/length(index2))>threshold
A(index2) = 1;
I_below(1:(j*rows),((i-1)*cols)+1:(i*cols))=A;
else
end
endif else
j==div
for i=1:div
if i==1
index1=find(I(((j-1)*rows)+1:ImageHeight,1:(i*cols))==1);
index2=find(I_below(((j-1)*rows)+1:ImageHeight,1:(i*cols))==1);
A = I_below(((j-1)*rows)+1:ImageHeight,1:(i*cols));
if isempty(index1) && isempty(index2)
else
if length(index1)>length(index2) || (length(index1)/length(index2))>threshold
A(index2) = 1;
I_below(((j-1)*rows)+1:ImageHeight,1:(i*cols))=A;
else
end
endif else
i==div
index1=find(I(((j-1)*rows)+1:ImageHeight,((i-1)*cols)+1:ImageWidth)==1);
index2=find(I_below(((j-1)*rows)+1:ImageHeight,((i-1)*cols)+1:ImageWidth)==1);
A = I_below(((j-1)*rows)+1:ImageHeight,((i-1)*cols)+1:ImageWidth);
if isempty(index1) && isempty(index2)
else

if length(index1) > length(index2) || 
(length(index1)/length(index2)) > threshold
    A(index2) = 1;
    I_below(((j-1)*rows)+1:ImageHeight,((i-1)*cols)+1:ImageWidth) = A;
else
    end
end
else
    index1 = find(I(((j-1)*rows)+1:ImageHeight,((i-1)*cols)+1:(i*cols)) == 1);
    index2 = find(I_below(((j-1)*rows)+1:ImageHeight,((i-1)*cols)+1:(i*cols)) == 1);
    A = I_below(((j-1)*rows)+1:ImageHeight,((i-1)*cols)+1:(i*cols));
    if isempty(index1) && isempty(index2)
    else
    if length(index1) > length(index2) || 
    (length(index1)/length(index2)) > threshold
    A(index2) = 1;
    I_below(((j-1)*rows)+1:(j*rows),((i-1)*cols)+1:(i*cols)) = A;
    else
    end
    else
    if i == div
    index1 = find(I(((j-1)*rows)+1:(j*rows),((i-1)*cols)+1:ImageWidth) == 1);
    index2 = find(I_below(((j-1)*rows)+1:(j*rows),((i-1)*cols)+1:ImageWidth) == 1);
    A = I_below(((j-1)*rows)+1:(j*rows),((i-1)*cols)+1:ImageWidth);
    if isempty(index1) && isempty(index2)
    else
    if length(index1) > length(index2) || 
    (length(index1)/length(index2)) > threshold
    A(index2) = 1;
    I_below(((j-1)*rows)+1:(j*rows),((i-1)*cols)+1:ImageWidth) = A;
    else
    end
    end
    else
    index1 = find(I(((j-1)*rows)+1:(j*rows),((i-1)*cols)+1:(i*cols)) == 1);
    index2 = find(I_below(((j-1)*rows)+1:(j*rows),((i-1)*cols)+1:(i*cols)) == 1);
    A = I_below(((j-1)*rows)+1:(j*rows),((i-1)*cols)+1:(i*cols));
    if isempty(index1) && isempty(index2)
```matlab
else
    if length(index1)>length(index2) ||
        (length(index1)/length(index2))>threshold
        A(index2) = 1;
        I_below(((j-l)*rows)+1:(j*rows),((i-l)*cols)+1:(i*cols)) = A;
    else
end
end
end

figure
imshow(I_below)
title('2D Image of Outliers Below after clustering')
figure
imshowpair(I, I_below)
title('2D Image of Inlier overlaid on Outliers Below after clustering')

%% Saliency map of Outliers Below
thresh = 0.35;

map_below = simpsal(I_below);
map_below = mat2gray(imresize( map_below , [ size(I_below,1) size(I_below,2) ] ));

% Find the max of the pothole for the saliency map
map_below_corr = map_below;

depth_max_saliency = max(max(map_below_corr));
[row, col] = find(map_below == depth_max_saliency);
location_depth_max_saliency = [row, col];

figure
imshow(map_below)
title('Saliency Map of Outliers Below after clustering')

% Smoothing of the Saliency Map into a BW image
indexMap=find(map_below<thresh);
map_below(indexMap) = 0;

indexMap = find(map_below==0);
map_below(indexMap) = 1;

% Check for multiple distinct potholes and disregard the smaller of the two
[L, num] = bwlabel(map_below);
if num > 1
    index_label_1 = find(L == 1);
    index_label_2 = find(L == 2);
    if length(index_label_1) > length(index_label_2)
        map_below(index_label_2) = 0;
    else
        map_below(index_label_1) = 0;
    end
else
end

% Determine the PIXEL length and width
[row, col] = find(map_below==1);
row_min = min(row);
```
row_max = max(row);
col_min = min(col);
col_max = max(col);

index = find(row==row_min);
lghtMin = col(index);
LengthMin = [row_min, round(mean(lghtMin))];

index = find(row==row_max);
lghtMax = col(index);
LengthMax = [row_max, round(mean(lghtMax))];

Length_in_pixels = sqrt((LengthMax(2)-LengthMin(2))^2+(LengthMax(1)-LengthMin(1))^2);

index = find(col==col_min);
widthMin = row(index);
WidthMin = [round(mean(widthMin)), col_min];

index = find(col==col_max);
widthMax = row(index);
WidthMax = [round(mean(widthMax)), col_max];

Width_in_pixels = sqrt((WidthMax(2)-WidthMin(2))^2+(WidthMax(1)-WidthMin(1))^2);

WidthMin = fliplr(WidthMin);
WidthMax = fliplr(WidthMax);
x_width = [WidthMin(1), WidthMax(1)];
y_width = [WidthMin(2), WidthMax(2)];

LengthMin = fliplr(LengthMin);
LengthMax = fliplr(LengthMax);
x_length = [LengthMin(1), LengthMax(1)];
y_length = [LengthMin(2), LengthMax(2)];

figure
imshow(map_below)
hold on
line(x_width, y_width, 'Color', 'red')
line(x_length, y_length, 'Color', 'red')
hold off
title('Smoothed Saliency Map that includes length and width lines')

%% Determine the ACTUAL length, width and depth

%% Find centroid of pothole
stats = regionprops(map_below);
centroid = stats.Centroid;
centroid = round(centroid);

%% Calculate real-world pixel height and width using depth at centroid
% location pixel
pixel_width = RangeImageFilt(centroid(2),centroid(1))*2*tand(35.3)/ImageWidth;
pixel_height = RangeImageFilt(centroid(2),centroid(1))*2*tand(30)/ImageHeight;

conv = 35.3701;
Length = (Length_in_pixels * pixel_height) * conv;
Width = (Width_in_pixels * pixel_width) * conv;

%% Find max depth and mean depth

%% Define the height and width of 3D pointcloud plot
row = y_max - y_min;

71
col = x_max - x_min;

% Divide the 3D image into rows and columns corresponding to 2D image
row_res = row/ImageHeight;
col_res = col/ImageWidth;

pothole_pointCloud = PointCloud.Location;

index = find(PointCloud.Location(:,2) > y_max-(LengthMin(2)*row_res));
pothole_pointCloud(index,:) = NaN;
index = find(PointCloud.Location(:,2) < y_max-(LengthMax(2)*row_res));
pothole_pointCloud(index,:) = NaN;
index = find(PointCloud.Location(:,1) < x_min+(WidthMin(1)*col_res));
pothole_pointCloud(index,:) = NaN;
index = find(PointCloud.Location(:,1) > x_min+(WidthMax(1)*col_res));
pothole_pointCloud(index,:) = NaN;

% Plot 3D Inliers, Outliers Above and Outliers Below
figure
plot3(PointCloud.Location(~IdxOutliers,1), PointCloud.Location(~IdxOutliers,2), PointCloud.Location(~IdxOutliers,3), 'b.' )
hold on
plot3(pothole_pointCloud(:,1), pothole_pointCloud(:,2), pothole_pointCloud(:,3), 'g.' )
plot3(PointCloud.Location(IdxOutliersAbove,1), PointCloud.Location(IdxOutliersAbove,2), PointCloud.Location(IdxOutliersAbove,3), 'r.' )
hold off
title('Outlier Points')
legend('Inliers', 'Outliers Below', 'Outliers Above', 'Location', 'NorthEast')

select = ~isnan(pothole_pointCloud(:,1));
index = find(select == 1);
depth = pothole_pointCloud(index,3);  % Depths of all points in the pothole
Depth = (abs(mean(depth)) * conv;  % Mean depth of all point in the pothole
depth_max = abs(min(depth)) * conv;  % Max depth of the pothole

% Calculate area and volume then assign classification value to road defect

Area = Length * Width;
Volume = Area * Depth;

check = 0;

if Length <= 3  % Initial spot check of length, width and depth
    class = 0;
    check = 1;
elseif Width <= 3
    class = 0;
    check = 2;
elseif depth_max <= 1
    class = 0;
    check = 3;
else
    % If spot check passes, a classification is assigned from 1-5 depending on the volume
    if isnan(Volume)
        class = 0;
    elseif Volume > 0 && Volume <= 50
        class = 1;
    elseif Volume > 50 && Volume <= 150
        class = 2;
    elseif Volume > 150 && Volume <= 250
        class = 3;
    elseif Volume > 250 && Volume <= 350
        class = 4;
    endif
end

72
if is_nan(Length)
    fprintf('The length of the defect is 0 inches \n\n');
else
    fprintf('The length of the defect is %.2f inches \n\n', Length);
end
if is_nan(Width)
    fprintf('The width of the defect is 0 inches \n\n');
else
    fprintf('The width of the defect is %.2f inches \n\n', Width);
end
fprintf('The max depth of the defect is %.2f inches \n\n', depth_max);
fprintf('The mean depth of the defect is %.2f inches \n\n', Depth);
if is_nan(Volume)
    fprintf('The volume of the defect is 0 cubic inches \n\n');
else
    fprintf('The volume of the defect is %.2f cubic inches \n\n', Volume);
end
fprintf('The class assigned to the defect is %d', class);
    if check == 0
        fprintf('.
\n\n');
    elseif check == 1
        fprintf(' because the length was less than 3 inches long \n\n');
    elseif check == 2
        fprintf(' because the width was less than 3 inches long \n\n');
    elseif check == 3
        fprintf(' because the max depth was less than 1 inch deep \n\n');
    else
        fprintf('\n\n')
end

clear x y z x_length x_width y_length y_width stats PixelSize NormalAlignment NormalsTire NormalsUp NormalVectorNew NormalVectorOld NormalVectorOriginal i j k index index1 index2 mat ii A c same div indexMap conv MaxIter q Scales
Appendix F  - C++ code of Detection/Classification Algorithm

```cpp
#define HEIGHT 424
#define WIDTH 512

#include <opencv2/highgui/highgui.hpp>
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv2/core/core.hpp>
#include <opencv2/photo.hpp>
#include <iostream>
#include <string>
#include <sstream>
#include <vector>
#include <iterator>
#include <fstream>
#include <math.h>
#include <time.h>
#include <chrono>
#include "opencv2/surface_matching.hpp"
#include "opencv2/surface_matching/ppf_helpers.hpp"
#include "opencv2/core/utility.hpp"
#include "opencv2/ml/ml.hpp"
#include "opencv2/flann/flann.hpp"
#include <pcl/io/io.h>
#include <pcl/io/pcd_io.h>
#include <pcl/features/integral_image_normal.h>
#include <pcl/point_types.h>
#include <pcl/features/normal_3d.h>
#include <pcl/features/normal_3d_omp.h>
#include <opencv2/saliency.hpp>
#include <opencv2/saliency/saliencySpecializedClasses.hpp>
#include <libfreenect2/libfreenect2.hpp>
#include <libfreenect2/frame_listener_impl.h>
#include <libfreenect2/registration.h>
#include <libfreenect2/packet_pipeline.h>
#include <libfreenect2/logger.h>
#include <atomic>

using namespace std;
using namespace cv;
using namespace ppf_match_3d;
using namespace saliency;

//Split function based on character. Used to read in the csv file for testing
template<typename Out>
void split(const std::string &s, char delim, Out result) {
  std::stringstream ss(s);
  std::string item;
  while (std::getline(ss, item, delim)) {
    *(result++) = item;
  }
}
// ROUNDS TO A CERTAIN NUMBER OF DIGITS. NEEDED FOR A SIGNIFICANT Figure OPeration
double round_to_digits(double value, int digits)
{
    if (value == 0.0) // otherwise it will return 'nan' due to the log10() of zero
        return 0.0;

    double factor = pow(10.0, digits - ceil(log10(fabs(value))));
    return round(value * factor) / factor;
}

// USES THE SPLIT OTHER SPLIT FUNCTION TO RETURN A
// VECTOR OF ELEMENTS AGAIN USED FOR READING CSV FILE.
std::vector<std::string> split(const std::string & s, char delim) {
    std::vector<std::string> elems;
    split(s, delim, std::back_inserter(elems));
    return elems;
}

// READS AND ENTIRE FILE INTO A STRING. USED FOR CSV
std::string slurp(ifstream & in) { // same deal as split might be inefficient
    stringstream sstr;
    sstr << in.rdbuf();
    return sstr.str();
}

// TWO FUNCTION FOR QUANTILE SIMILAR TO MATLAB IMPLEMENTATION.
// MIGHT NOT NEED IT I DON'T THINK THE QUARTILE IS USED FOR ANYTHING.
template<typename T>
static inline double Lerp(T v0, T v1, T t)
{
    return (1 - t)*v0 + t*v1;
}

template<typename T>
static inline std::vector<T> Quantile(const std::vector<T> & inData, const std::vector<T>& probs)
{
    if (inData.empty())
    {
        return std::vector<T>();
    }

    if (1 == inData.size())
    {
        return std::vector<T>({1, inData[0]});
    }

    std::vector<T> data = inData;
    std::sort(data.begin(), data.end());
    std::vector<T> quantiles;
    for (size_t i = 0; i < probs.size(); ++i)
T poi = Lerp<T>(-0.5, data.size() - 0.5, probs[i]);
size_t left = std::max(int64_t(std::floor(poi)), int64_t(0));
size_t right = std::min(int64_t(std::ceil(poi)), int64_t(data.size() - 1));
T datLeft = data.at(left);
T datRight = data.at(right);
T quantile = Lerp<T>(datLeft, datRight, poi - left);
quantiles.push_back(quantile);
}

return quantiles;
}

//plane fit SVD does not return alpha and beta.
//Takes in average array and the Matlab array
Mat planeFitSVD(double average[3], Mat m){
    average[0] = average[0]/m.size().height;
    average[1] = average[1]/m.size().height;
    average[2] = average[2]/m.size().height;
    Mat locationCheck(m.size().height,3, CV_64F);
    for (int i = 0; i < m.size().height; ++i)
        for (int k = 0; k<3; ++k){
            locationCheck.at<double>(i,k) = m.at<double>(i,k) - average[k];
        }

    Mat w,u,v,vt;
    Mat locationTransposed = locationCheck.t();

    Mat result = locationTransposed*locationCheck;
    SVD::compute(result,w,u,vt);
    v = vt.t();
    v = v.mul(-1);

    double normalizingFactor = sqrt(v.at<double>(0,2)*v.at<double>(0,2) +
v.at<double>(1,2)*v.at<double>(1,2) + v.at<double>(2,2)*v.at<double>(2,2));
    double normvect[3] = {v.at<double>(0,2)/normalizingFactor,v.at<double>(1,2)/
normalizingFactor,v.at<double>(2,2)/normalizingFactor};

    //don't know if I actually need the alpha beta stuff
    double alpha = atan(normvect[2]/normvect[1]);
    double beta = asin(normvect[0]);
    Mat vect = Mat(3,1,CV_64F);
    vect.at<double>(0,0) = normvect[0];
    vect.at<double>(1,0) = normvect[1];
    vect.at<double>(2,0) = normvect[2];
return vect;
}

Mat pointCloud2Image(Mat x, Mat y, Mat z, int numr, int numc)
{
    Mat d = Mat(x.size().height,1,CV_64F);
    for (int i = 0; i < x.size().height; ++i)
    {
        d.at<double>(i,0) = sqrt(x.at<double>(i,0)*x.at<double>(i,0) +
                                  y.at<double>(i,0)*y.at<double>(i,0) + z.at<double>(i,0)*z.at<double>(i,0));
    }
    double ymin,ymax, xmin,xmax;
    minMaxLoc(y,&ymin,&ymax);
    minMaxLoc(x,&xmin,&xmax);

    // my implementation of linspace not going to make it
    // a function because it is only used here
    double ydiff,xdiff,yinc,xinc;
    ydiff = ymax-ymin;
    xdiff = xmax-xmin;
    yinc = ydiff/(numc-1);
    xinc = xdiff/(numr-1);

    Mat tmp;
    hconcat(y,x,tmp);

    int K=1;

    Mat class_stat = Mat::zeros(numr*numc,1,CV_64F);
    for (int i = 0; i < tmp.size().height; ++i)
    {
        int place = round((tmp.at<double>(i,0)-ymin)/yinc)*HEIGHT +
                        round((tmp.at<double>(i,1)-xmin)/xinc);
        class_stat.at<double>(place,0) = d.at<double>(0,i);
    }
    Mat class_stat_M = class_stat.reshape(0,numc);
    Mat I;
    flip(class_stat_M,I,0);
    double imin,imax;
    minMaxLoc(I,&imin,&imax);

    for (int i = 0; i < I.size().height; ++i)
    {
        for (int j = 0; j < I.size().width; ++j)
        {
            if(I.at<double>(i,j) == 0 ){
                I.at<double>(i,j) = imax;
            }
        }
    }
}
for (int i = 0; i < I.size().height; ++i)
{
    for (int j = 0; j < I.size().width; ++j)
    {
        I.at<double>(i, j) = (I.at<double>(i, j) - imin) / (imax - imin);
    }
}

return I;
}

pcl::PointCloud<pcl::PointXYZ>::Ptr MatToPointXYZ(cv::Mat OpenCVPointCloud){
/*
 + Function: Get from a Mat to pcl pointcloud datatype
 + In: cv::Mat
 + Out: pcl::PointCloud
*/

pcl::PointCloud<pcl::PointXYZ>::
Ptr point_cloud_ptr(new pcl::PointCloud<pcl::PointXYZ>);
// (new pcl::PointCloud<pcl::PointXYZ>);

for(int i=0;i<OpenCVPointCloud.cols;i++)
{
    pcl::PointXYZ point;
    point.x = OpenCVPointCloud.at<double>(0, i);
    point.y = OpenCVPointCloud.at<double>(1, i);
    point.z = OpenCVPointCloud.at<double>(2, i);
    point_cloud_ptr->points.push_back(point);
}

point_cloud_ptr->width = (int)point_cloud_ptr->points.size();
point_cloud_ptr->height = 1;

return point_cloud_ptr;
}

int main( int argc, char** argv ) {

double pointCloud [HEIGHT][WIDTH][3];
static double originalData [HEIGHT][WIDTH][3];

int runtime = 0;

for(int z = 0; z < 1; z++)
{
    vector<string> columns;
    ifstream myfile ("frame_91.csv");
    string buffer = slurp(myfile);
    vector<std::string> toks = split(buffer, \\
    return;
}
for (int i = 0; i < HEIGHT; ++i) {
    columns = split(toks[i], ',');
    for (int k = 0; k < WIDTH; k++) {
        if (columns[k][0] != 'N') {
            pointCloud[i][k][0] = atof(columns[k].c_str());
            originalData[i][k][0] = atof(columns[k].c_str());
        } else {
            pointCloud[i][k][0] = nan("NaN");
            originalData[i][k][0] = nan("NaN");
        }
    }
}

std::ifstream myfile2("frame_92.csv");
buffer = slurp(myfile2);
toks = split(buffer, '\n');
for (int i = 0; i < HEIGHT; ++i) {
    columns = split(toks[i], ',');
    for (int k = 0; k < WIDTH; k++) {
        if (columns[k][0] != 'N') {
            pointCloud[i][k][1] = atof(columns[k].c_str());
            originalData[i][k][1] = atof(columns[k].c_str());
        } else {
            pointCloud[i][k][1] = nan("NaN");
            originalData[i][k][1] = nan("NaN");
        }
    }
}

std::ifstream myfile3("frame_93.csv");
buffer = slurp(myfile3);
toks = split(buffer, '\n');
for (int i = 0; i < HEIGHT; ++i) {
    columns = split(toks[i], ',');
    for (int k = 0; k < WIDTH; k++) {
        if (columns[k][0] != 'N') {
            pointCloud[i][k][2] = atof(columns[k].c_str());
            originalData[i][k][2] = atof(columns[k].c_str());
        } else {
            pointCloud[i][k][2] = nan("NaN");
            originalData[i][k][2] = nan("NaN");
        }
    }
}

//
    double numNans = 0;
auto ttc = std::chrono::high_resolution_clock::now(); // timing for benchmarking

Mat rangeImage(HEIGHT, WIDTH, CV_64F);
Mat gaussImage(HEIGHT, WIDTH, CV_64F);

// getting the distance from origin
for (int i = 0; i < HEIGHT; ++i)
    for (int j = 0; j < WIDTH; ++j)
    {
        double sum = 0;
        for (int k = 0; k < 3; ++k)
            sum = sum + pointCloud[i][j][k]*pointCloud[i][j][k];
        rangeImage.at<double>(i, j) = sqrt(sum);
    }

// applying Gaussian filter

cv::GaussianBlur(rangeImage, gaussImage, Size(5, 5), 1);

// applying the scaling factor.
double scalingFactor = 0;
for (int i = 0; i < HEIGHT; ++i)
    for (int j = 0; j < WIDTH; ++j)
    {
        scalingFactor = gaussImage.at<double>(i, j)/rangeImage.at<double>(i, j);

        // doing this in place to save memory could eliminate scales array totally
        for (int k = 0; k < 3; ++k)
            pointCloud[i][j][k] = pointCloud[i][j][k]*scalingFactor;
        if (isnan(pointCloud[i][j][0])){
            numNans++;
        }
    }

// point cloud is scaled at this point

// Start of Transform_Init will make a function later

// taking of the edges of the image
Mat reducedpc((HEIGHT-59) * (WIDTH-59), 3, CV_64F);

int counter = 0;
for (int i = 29; i < WIDTH-30; ++i)
    for (int j = 29; j < HEIGHT-30; ++j)
    {
reducedpc.at<double>((i-29)*(HEIGHT-59)+(j-29),0) = pointCloud[j][i][0];
reducedpc.at<double>((i-29)*(HEIGHT-59)+(j-29),1) = pointCloud[j][i][1];
reducedpc.at<double>((i-29)*(HEIGHT-59)+(j-29),2) = pointCloud[j][i][2];
if(isnan(pointCloud[j][i][0])){
    counter++;
}

//taking out the invalid values
Mat filteredReducedPC((HEIGHT-59)*(WIDTH-59)-counter,3,CV_64F);
counter = 0;
double average[3] = {0,0,0};
for(int i = 0; i<(HEIGHT-59)*(WIDTH-59);++i){
    if(!isnan(reducedpc.at<double>(i,0))){
        //might want to clean up with a for loop
        filteredReducedPC.at<double>(counter,0) = reducedpc.at<double>(i,0);
        filteredReducedPC.at<double>(counter,1) = reducedpc.at<double>(i,1);
        filteredReducedPC.at<double>(counter,2) = reducedpc.at<double>(i,2);
        average[0] += reducedpc.at<double>(i,0);
        average[1] += reducedpc.at<double>(i,1);
        average[2] += reducedpc.at<double>(i,2);
        counter++;
    }
}

// calculating the rotation matrix
//matrix of normals and cross multiplied with the cross vector
Mat normmat = planeFitSVD(average,filteredReducedPC);
double crossVect[3][1] = {{-1},{0},{0}};

Mat tmp = Mat(3,1,CV_64F,crossVect);
Mat y = normmat.cross(tmp);
double normy = 0;

for (int i = 0; i < 3; ++i){
    normy = normy + y.at<double>(i,0)*y.at<double>(i,0);
}
normy = sqrt(normy);
y = y.mul(1/normy);

Mat rotation(3,3,CV_64F);
rotation.row(0) = tmp.mul(-1).t();
rotation.row(1) = y.t();
rotation.row(2) = normmat.mul(-1).t();
Mat r = rotation*filteredReducedPC.t();

double mean[3][1] = {0,0,0};
for (int i = 0; i < r.size().width; ++i)
{  
    for (int j = 0; j < 3; ++j)
    {
        mean[j][0] += r.at<double>(j, i);
    }
}

mean[0][0] = mean[0][0]/r.size().width;
mean[1][0] = mean[1][0]/r.size().width;
mean[2][0] = mean[2][0]/r.size().width;

Mat meanMat = Mat(1,3,CV_64F,mean);
double translation[3][1] = {{{0},{0},{-meanMat.at<double>(0,2)}}};
//end of Transform_Init function  

double minPotholeSize = 0.05;

Mat goodData = Mat(HEIGHT*WIDTH-numNans,3,CV_64F);
int index = 0;
for(int i =0;i<WIDTH;++i){
    for(int j =0;j<HEIGHT;++j){
        if(!isnan(pointCloud[j][i][0])){
            goodData.at<double>(index,0) = pointCloud[j][i][0];
            goodData.at<double>(index,1) = pointCloud[j][i][1];
            goodData.at<double>(index,2) = pointCloud[j][i][2];
            index++;
        }
    }
}

Mat pointCloudTire = rotation * goodData.t();
for (int i = 0; i<goodData.size().height;i++){
    pointCloudTire.at<double>(2,i) = pointCloudTire.at<double>(2,i) + translation[2][0];
    average[0] += pointCloudTire.at<double>(0,i);
    average[1] += pointCloudTire.at<double>(1,i);
    average[2] += pointCloudTire.at<double>(2,i);
}

int normcount = 0;
pointCloudTire = pointCloudTire.t();
Mat normals = Mat(goodData.size().height,6,CV_64F);

//timing  
auto t2c = std::chrono::high_resolution_clock::now();
std::cout << "reading in and preping data took " << std::chrono::duration_cast<std::chrono::milliseconds>(t2c-t1c).count() << " milliseconds\n";

//alternative normal estimation faster but noisier  
// load point cloud
pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::PointCloud<pcl::PointXYZ>);
pcl::NormalEstimationOMP<pcl::::PointXYZ, pcl::::Normal> ne;
pcl::search::KdTree<pcl::::PointXYZ>::Ptr tree (new pcl::search::KdTree<pcl::::PointXYZ> ());
pcl::PointCloud<pcl::::Normal>::Ptr cloud_normals (new pcl::PointCloud<pcl::::Normal>);
cloud = MatToPointCloud(pointCloudTire.t());

// Create the normal estimation class, and pass the input dataset to it

ne.setInputCloud (cloud);
// Create an empty kdtree representation, and pass it to the normal estimation object.
// Its content will be filled inside the object, based on the given input dataset
//(as no other search surface is given).
ne.setSearchMethod (tree);

// Output datasets

ne.setKSearch (25);
// Compute the features
ne.compute(*cloud_normals);

for (int i = 0; i < goodData.size().height; ++i)
{
    // copy fast values here if need be
    normals.at<double>(i, 3) = (double)-cloud_normals->points[i].normal_x;
    normals.at<double>(i, 4) = (double)-cloud_normals->points[i].normal_y;
    normals.at<double>(i, 5) = (double)-cloud_normals->points[i].normal_z;
}

Mat normalsTire = normals.colRange(3, 6);
Mat normalVectorOriginal = planeFitSVD(average, pointCloudTire).mul(-1);
normalsTire.convertTo(normalsTire, CV_64F);
Mat NormalsUp = normalsTire * normalVectorOriginal;

int continueIter = 1;
double iter = 0;
double maxIter = 20;

double angleTol = 0.85;
double distTol = 0.005;
double nDist = 6;

double mUpperBound = (nDist*distTol - distTol) / (1-angleTol);
double bUpperBound = distTol - mUpperBound * angleTol;

double mLowerBound = (distTol - nDist*distTol) / (1-angleTol);
double bLowerBound = -distTol - mLowerBound * angleTol;

Mat roadsurface = pointCloudTire;
Mat newNormalVector, normalAlignment;
Mat pointCloudTireOrig = pointCloudTire;
pointCloudTire.copyTo(pointCloudTireOrig);
Mat normalsTireOrig;
normalsTire.copyTo(normalsTireOrig);

Mat inliers = Mat::ones(pointCloudTire.size().height, 1, CV_8UC3);
int numInliersOld = inliers.size().height;

Mat idxOutliersAbove, idxOutliersBelow, idxOutliers;

Mat bxtmp = Mat(pointCloudTire.size().height, 1, CV_64F);
Mat distToPlaneCentroid = Mat(pointCloudTire.size().height, 1, CV_64F);
std::atomic<int> value;
while (continueIter){
    double centroid[3] = {0, 0, 0};
    for (int i = 0; i < roadsurface.size().height; ++i)
    {
        centroid[0] += roadsurface.at<double>(i, 0);
        centroid[1] += roadsurface.at<double>(i, 1);
        centroid[2] += roadsurface.at<double>(i, 2);
    }
    newNormalVector = planeFitSVD(centroid, roadsurface).mul(-1);

    normalAlignment = normalsTireOrig * newNormalVector;

    idxOutliersAbove = Mat::zeros(normalAlignment.size().height, 3, CV_8U);
    idxOutliersBelow = Mat::zeros(normalAlignment.size().height, 3, CV_8U);
    idxOutliers = Mat::zeros(normalAlignment.size().height, 3, CV_8U);
    int numInliersNew = 0;

    pointCloudTire.col(0).forEach<double>(
        [&value, &pointCloudTire, &bxtmp, &centroid, &distToPlaneCentroid, &newNormalVector, 
         &normalAlignment, &idxOutliersAbove, &idxOutliersBelow, &inliers, &numInliersNew, 
         &angleTol, &distTol, &bUpperBound, &bLowerBound, &mUpperBound, &mLowerBound]
        (double & const int * position) -> void{
            int i = position[0];
            bxtmp.at<double>(i, 0) = pointCloudTire.at<double>(i, 0) - centroid[0];
            bxtmp.at<double>(i, 1) = pointCloudTire.at<double>(i, 1) - centroid[1];
            bxtmp.at<double>(i, 2) = pointCloudTire.at<double>(i, 2) - centroid[2];
            distToPlaneCentroid.at<double>(i, 0) = bxtmp.at<double>(i, 0) * newNormalVector.at<double>(0, 0) + bxtmp.at<double>(i, 1) * newNormalVector.at<double>(0, 1) + bxtmp.at<double>(i, 2) * newNormalVector.at<double>(0, 2);

            if ((normalAlignment.at<double>(i, 0) < angleTol && distToPlaneCentroid.at<double>(i, 0) >= distTol) ||
                (normalAlignment.at<double>(i, 0) >= angleTol && (distToPlaneCentroid.at<double>(i, 0) >= mUpperBound * normalAlignment.at<double>(i, 0) + bUpperBound))){
                idxOutliersAbove.at<char>(i, 0) = 1;
                idxOutliersAbove.at<char>(i, 1) = 1;
                idxOutliersAbove.at<char>(i, 2) = 1;
            }
        });

    value = value + 1;
if((normalAlignment.at<double>(i, 0) < angleTol && distToPlaneCentroid.at<double>(i, 0) <= -distTol) ||
(normalAlignment.at<double>(i, 0) >= angleTol && (distToPlaneCentroid.at<double>(i, 0) <=
(mLowerBound*normalAlignment.at<double>(i, 0) + bLowerBound))){
    idxOutliersBelow.at<char>(i, 0) = 1;
    idxOutliersBelow.at<char>(i, 1) = 1;
    idxOutliersBelow.at<char>(i, 2) = 1;
}
inliers.at<char>(i, 0) = (idxOutliersAbove.at<char>(i, 0) == 1) || idxOutliersBelow.at<char>(i, 0); 
inliers.at<char>(i, 1) = (idxOutliersAbove.at<char>(i, 1) == 1) || idxOutliersBelow.at<char>(i, 1); 
inliers.at<char>(i, 2) = (idxOutliersAbove.at<char>(i, 2) == 1) || idxOutliersBelow.at<char>(i, 2); 
if(inliers.at<char>(i, 0)==1){
    value++;
}
}

numInliersNew = value.load();
value.store(0);

//int numInliersNew = countNonZero(inliers)/3;
Mat newPC = Mat(numInliersNew, 3, CV_64F);
Mat newNorm = Mat(numInliersNew, 3, CV_64F);

for (int i = 0; i < inliers.size().height; ++i) {
    if(inliers.at<char>(i, 0)!=0){
        newPC.at<double>(counter, 0) = pointCloudTire.at<double>(i, 0);
        newPC.at<double>(counter, 1) = pointCloudTire.at<double>(i, 1);
        newPC.at<double>(counter, 2) = pointCloudTire.at<double>(i, 2);
        newNorm.at<double>(counter, 0) = normalsTireOrig.at<double>(i, 0);
        newNorm.at<double>(counter, 1) = normalsTireOrig.at<double>(i, 1);
        newNorm.at<double>(counter, 2) = normalsTireOrig.at<double>(i, 2);
        counter++;
    }
}

if(iter >= maxIter || abs(numInliersNew - numInliersOld)==0){
    continueIter = 0;
}
else{
    numInliersOld = numInliersNew;
    newPC.copyTo(roadsurface);
    newNorm.copyTo(normalsTire);
    iter++;
}
Mat xcoords = roadsurface.col(0);
Mat ycoords = roadsurface.col(1);
Mat zcoords = roadsurface.col(2);
double x_max, x_min, y_max, y_min;
Mat I = pointCloud2Image(xcoords, ycoords, zcoords, HEIGHT, WIDTH);
minMaxLoc(ycoords, &y_min, &y_max);
minMaxLoc(xcoords, &x_min, &x_max);

int numInliersBelow = countNonZero(idxOutliersBelow) / 3;
Mat xcoords2 = Mat(numInliersBelow, 1, CV_64F);
Mat ycoords2 = Mat(numInliersBelow, 1, CV_64F);
Mat zcoords2 = Mat(numInliersBelow, 1, CV_64F);
int imagecounter = 0;

for (int i = 0; i < idxOutliersBelow.size().height; ++i)
{
    if (idxOutliersBelow.at<char>(i, 0) == 1)
    {
        xcoords2.at<double>(imagecounter, 0) = pointCloudTireOrig.at<double>(i, 0);
        ycoords2.at<double>(imagecounter, 0) = pointCloudTireOrig.at<double>(i, 1);
        zcoords2.at<double>(imagecounter, 0) = pointCloudTireOrig.at<double>(i, 2);
        imagecounter++;
    }
}
double x_max2, x_min2, y_max2, y_min2;
Mat I_below = pointCloud2Image(xcoords2, ycoords2, zcoords2, HEIGHT, WIDTH);

double thresh_high = 0.6;
double thresh_low = 0.1;

for (int i = 0; i < I_below.rows; ++i)
{
    for (int j = 0; j < I_below.cols; ++j)
    {
        //probably should be else ifs will test later
        if (I.at<double>(i, j) > thresh_high || I.at<double>(i, j) < thresh_low)
        {
            I.at<double>(i, j) = 1;
        }
        if (I_below.at<double>(i, j) > thresh_high || I_below.at<double>(i, j) < thresh_low)
        {
            I_below.at<double>(i, j) = 1;
        }
        if (I_below.at<double>(i, j) == 1 && I.at<double>(i, j) == 1)
        {
            I_below.at<double>(i, j) = 1;
            I.at<double>(i, j) = 1;
        }
        if (I_below.at<double>(i, j) == 1 && I.at<double>(i, j) == 1)
        {
            I_below.at<double>(i, j) = 0;
        }
        if (I_below.at<double>(i, j) == 1 && I.at<double>(i, j) == 1)
        {
            I_below.at<double>(i, j) = 1;
            I.at<double>(i, j) = 1;
        }
    }
}
void matchPoint(int rows, int cols, int &x, int &y) {
    int div = 30;
    int rows = round(HEIGHT/div);
    int cols = round(WIDTH/div);

    double threshold = 0.8;

    for (int j = 0; j < div; ++j) {
        for (int i = 0; i < div; ++i) {
            if(i==div-1 && j==div-1) { // all blocks that are guaranteed to be divxdiv
                double index1 = 0;
                double index2 = 0;
                for (int m = j*rows; m < j*rows+rows-1 ; ++m) {
                    for (int n = i*cols; n < i*cols+cols-1; ++n) {
                        if(I.at<double>(m,n)!=1){
                            index1++;
                        }
                        if(I_below.at<double>(m,n)!=1){
                            index2++;
                        }
                    }
                }
                if(index1>index2 || (index1/index2)>threshold){
                    for (int m = j*rows; m < j*rows+rows-1 ; ++m) {
                        for (int n = i*cols; n < i*cols+cols-1; ++n) {
                            if(I_below.at<double>(m,n)!=1){
                                x = m;
                                y = n;
                            }
                        }
                    }
                }
            }
        }
    }
}

if(I_below.at<double>(m,n)!==1)
{
    index2++;
}
}

if(index1>index2 || (index1/index2)>threshold)
{
    for (int m = j*rows; m < j*rows+rows-1; ++m)
    {
        for (int n = i*cols; n < WIDTH-1; ++n)
        {
            I_below.at<double>(m,n)!==1;
        }
    }
}

else if(i=div-1 && j=div-1)//right edge
    double index1 = 0;
    double index2 = 0;
    for (int m = j*rows; m < HEIGHT-1; ++m)
    {
        for (int n = i*cols; n < i*cols+cols-1; ++n)
        {
            if(I.at<double>(m,n)!==1)
            {
                index1++;
            }
            if(I_below.at<double>(m,n)!==1)
            {
                index2++;
            }
        }
    }

if(index1>index2 || (index1/index2)>threshold)
{
    for (int m = j*rows; m < HEIGHT-1; ++m)
    {
        for (int n = i*cols; n < i*cols+cols-1; ++n)
        {
            I_below.at<double>(m,n)!==1;
        }
    }
}

else{//bottom right corner
    double index1 = 0;
    double index2 = 0;
    for (int m = j*rows; m < HEIGHT-1; ++m)
    {
        for (int n = i*cols; n < WIDTH-1; ++n)
        {
            if(I.at<double>(m,n)!==1)
            {
                index1++;
            }
            if(I_below.at<double>(m,n)!==1)
            {
                index2++;
            }
        }
    }
}
if(index1>index2 || (index1/index2)>threshold){
    for (int m = j*rows; m < HEIGHT-1; ++m)
    {
        for (int n = i*cols; n < WIDTH-1; ++n)
        {
            I_below.at<double>(m,n)=1;
        }
    }
}

imshow("test1",I);
waitKey(0);
imshow("test",I_below);
waitKey(0);
Mat map_below,bmap_below;
double thresh = 0.4;

Mat label,stats,center,originalImage;
I_below.copyTo(originalImage);
I_below = I_below.mul(255);
I_below.convertTo(I_below, CV_8U);
//fastNlMeansDenoising(I_below,I_below,100);
    //Gaussian Blur (Below) is faster
blur(I_below,I_below,Size(8,8));
// imshow("test",I_below);
// waitKey(0);

cv::threshold(I_below,I_below , 30, 255, CV_THRESH_BINARY);

    // imshow("test1",I);
    // waitKey(0);
    // imshow("test",I_below);
    // waitKey(0);
int dilation_type;
double dilation_size = 15;
Mat src;
bitwise_not(I_below,I_below);
dilation_type = MORPH_ELLIPSE;
Mat element = getStructuringElement( dilation_type,
    Size( 2*dilation_size + 1, 2*dilation_size+1 ),
    Point( dilation_size, dilation_size ) );
dilate( I_below, I_below, element );

    // strftime (buffer3,80,"SaliencyMapImages/saliency%Y-%m-%d %r.jpg",now);
    // imwrite(buffer3,I_below);
cv::connectedComponentsWithStats(I_below, label, stats, center, 4, CV_32S);
I_below.at<char>(round(center.at<double>(1, 1)), round(center.at<double>(1, 0))) = 255;
imshow("test", I_below);
waitKey(0);

int centroid[2] = {(int)center.at<double>(1, 0), (int)center.at<double>(1, 1)};

double pixel_width = pointCloud[centroid[1]][centroid[0]][2] * 2 * tan(35.3 * 180 / 3.14159) / WIDTH;
double pixel_height = pointCloud[centroid[1]][centroid[0]][2] * 2 * tan(35.3 * 180 / 3.14159) / HEIGHT;

double conv = 39.3701;
double lengthIn = -conv * pixel_height * stats.at<int>(1, 3);
double widthIn = -conv * pixel_width * stats.at<int>(1, 2);

double row = y_max - y_min;
double col = x_max - x_min;

double row_res = row / HEIGHT;
double col_res = col / WIDTH;
double mean_depth = 0;
double min_depth = 0;
double max_depth = 0;
double depthCounter = 0;
for (int i = 0; i < pointCloudTireOrig.size().height; ++i)
{
    if ((pointCloudTireOrig.at<double>(i, 1) < (y_max - (stats.at<int>(1, 1) * row_res))
        && pointCloudTireOrig.at<double>(i, 1) > (y_max - (stats.at<int>(1, 1) +
            stats.at<int>(1, 3)) * row_res)) && pointCloudTireOrig.at<double>(i, 0) >
        (x_min + (stats.at<int>(1, 0) + stats.at<int>(1, 2)) * col_res)) && pointCloudTireOrig.at<double>(i, 0) <
        (x_min + (stats.at<int>(1, 0) + stats.at<int>(1, 2)) * col_res)))
    {
        if (pointCloudTireOrig.at<double>(i, 2) > max_depth)
        {
            max_depth = pointCloudTireOrig.at<double>(i, 2);
        }
        if (pointCloudTireOrig.at<double>(i, 2) < min_depth)
        {
            min_depth = pointCloudTireOrig.at<double>(i, 2);
        }
        depthCounter++;
        mean_depth += pointCloudTireOrig.at<double>(i, 2);
    }
}
max_depth = abs(max_depth * conv);
min_depth = abs(min_depth * conv);
double depth = abs(mean_depth * conv / depthCounter);
double area = lengthIn * widthIn;
double volume = area * depth;
double check = 0;
double classification = 0;

if (lengthIn <= 3)
{
    classification = 0;
    check = 1;
}
else if (widthIn <= 3)
{
    classification = 0;
    check = 2;
}
else
{
    if (volume > 0 && volume <= 50)
    {
        classification = 1;
    }
    else if (volume > 50 && volume <= 150)
    {
        classification = 2;
    }
    else if (volume > 150 && volume <= 250)
    {
        classification = 3;
    }
    else if (volume > 250 && volume <= 350)
    {
        classification = 4;
    }
    else if (volume > 350)
    {
        classification = 5;
    }
}

// cout << "Full color image at " << buffer << endl;
// cout << "Saliency image at " << buffer3 << endl;
// cout << "Depth/ color image at " << buffer2 << endl;

cout << "The length of the defect is " << lengthIn << endl;
cout << "The width of the defect is " << widthIn << endl;
cout << "The max depth of the defect is " << min_depth << endl;
cout << "The mean depth of the defect is " << depth << endl;
cout << "The volume of the defect is " << volume << endl;
cout << "The class assigned to the defect is " << classification << endl;
t2c = std::chrono::high_resolution_clock::now();
sumtime += std::chrono::duration_cast<std::chrono::milliseconds>(t2c-t1c).count();
std::cout << "The program took " << std::chrono::duration_cast<std::chrono::milliseconds>(t2c-t1c).count() << " milliseconds\n";

} cout << sumtime;
// dev->stop();
// dev->close();
return 0;
}
Appendix G  - Post-Processing of Additional Road Defects

Figure 55: Gibbons St. - Actual Color Image

Figure 56: Gibbons St. - Point Cloud
Figure 57: Results of Clustering

(a) Gibbons St. - Before Clustering  
(b) Gibbons St. - After Clustering

Figure 58: Gibbons St. - Outliers Overlaid with Inliers
Figure 59: Defining the Outline of the Road Defect
Figure 60: Afton St. - Actual Color Image

Figure 61: Afton St. - Point Cloud
Figure 62: Results of Clustering

(a) Afton St. - Before Clustering    (b) Afton St. - After Clustering

Figure 63: Afton St. - Outliers Overlaid with Inliers
Figure 64: Defining the Outline of the Road Defect
Figure 65: Summit Hall - Actual Color Image

Figure 66: Summit Hall - Point Cloud
(a) Summit Hall - Before Clustering  
(b) Summit Hall - After Clustering 

Figure 67: Results of Clustering

Figure 68: Summit Hall - Outliers Overlaid with Inliers
Figure 69: Defining the Outline of the Road Defect
Figure 70: McLane Ave. - Actual Color Image

Figure 71: McLane Ave. - Point Cloud
Figure 72: Results of Clustering

Figure 73: McLane Ave. - Outliers Overlaid with Inliers
Figure 74: Defining the Outline of the Road Defect
Figure 75: Seventh St. - Actual Color Image

Figure 76: Seventh St. - Point Cloud
Figure 77: Results of Clustering

(a) Seventh St. - Before Clustering
(b) Seventh St. - After Clustering

Figure 78: Seventh St. - Outliers Overlaid with Inliers
Figure 79: Defining the Outline of the Road Defect
Figure 80: Sharon Ave. - Actual Color Image

Figure 81: Sharon Ave. - Point Cloud
(a) Sharon Ave. - Before Clustering                (b) Sharon Ave. - After Clustering

Figure 82: Results of Clustering

Figure 83: Sharon Ave. - Outliers Overlaid with Inliers
Figure 84: Defining the Outline of the Road Defect
Figure 85: Not a Road Defect #1 - Actual Color Image

Figure 86: Not a Road Defect #1 - Point Cloud
Figure 87: Results of Clustering

Figure 88: Not a Road Defect #1 - Outliers Overlaid with Inliers
Figure 89: Defining the Outline of the Road Defect
Figure 90: Not a Road Defect #2 - Actual Color Image

Figure 91: Not a Road Defect #2 - Point Cloud
(a) Not a Road Defect #2 - Before Clustering  
(b) Not a Road Defect #2 - After Clustering

Figure 92: Results of Clustering

Figure 93: Not a Road Defect #2 - Outliers Overlaid with Inliers
Figure 94: Defining the Outline of the Road Defect
Figure 95: Not a Road Defect #3 - Actual Color Image

Figure 96: Not a Road Defect #3 - Point Cloud
Figure 97: Results of Clustering

Figure 98: Not a Road Defect #3 - Outliers Overlaid with Inliers
Figure 99: Defining the Outline of the Road Defect
Figure 100: Not a Road Defect #4 - Actual Color Image

Figure 101: Not a Road Defect #4 - Point Cloud
Figure 102: Results of Clustering

(a) Not a Road Defect #4 - Before Clustering  
(b) Not a Road Defect #4 - After Clustering

Figure 103: Not a Road Defect #4 - Outliers Overlaid with Inliers
Figure 104: Defining the Outline of the Road Defect