1-8-2020

3D LiDAR Point Cloud Processing Algorithms

Bradford Scott Bondy
University of Windsor

3D LiDAR Point Cloud Processing Algorithms

By

Bradford Bondy

A Thesis

Submitted to the Faculty of Graduate Studies

through the Department of Electrical and Computer Engineering

in Partial Fulfillment of the Requirements for

the Degree of Master of Applied Science

at the University of Windsor

Windsor, Ontario, Canada

2020

3D LiDAR Point Cloud Processing Algorithms

by

Bradford Bondy

APPROVED BY:

_____

I. Ahmad
School of Computer Science

_____

M. Azzouz
Department of Electrical and Computer Engineering

_____

M. Khalid, Advisor
Department of Electrical and Computer Engineering

January 8, 2020

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

In the race for autonomous vehicles and advanced driver assistance systems (ADAS), the automotive industry has energetically pursued research in the area of sensor suites to achieve such technological feats. Commonly used autonomous and ADAS sensor suites include multiples of cameras, radio detection and ranging (RADAR), light detection and ranging (LiDAR), and ultrasonic sensors. Great interest has been generated in the use of LiDAR sensors and the value added in an automotive application. LiDAR sensors can be used to detect and track vehicles, pedestrians, cyclists, and surrounding objects.

A LiDAR sensor operates by emitting light amplification by stimulated emission of radiation (LASER) beams and receiving the reflected LASER beam to acquire relevant distance information. LiDAR reflections are organized in a three-dimensional environment known as a point cloud. A major challenge in modern autonomous automotive research is to be able to process the dimensional environmental data in real time. The LiDAR sensor used in this research is the Velodyne HDL 32E, which provides nearly 700,000 data points per second. The large amount of data produced by a LiDAR sensor must be processed in a highly efficient way to be effective.

This thesis provides an algorithm to process the LiDAR data from the sensors user datagram protocol (UDP) packet to output geometric shapes that can be further analyzed in a sensor suite or utilized for Bayesian tracking of objects. The algorithm can be divided into three stages: Stage One - UDP packet extraction; Stage Two - data clustering; and Stage Three - shape extraction. Stage One organizes the LiDAR data from a negative to a positive vertical angle during packet extraction so that subsequent steps can fully exploit the programming efficiencies. Stage Two utilizes an adaptive breakpoint detector (ABD) for clustering objects based on a Euclidean distance threshold in the point cloud. Stage Three classifies each cluster into a shape that is either a point, line, L-shape, or a polygon using principal component analysis and shape fitting algorithms that have been modified to take advantage of the pre-organized data from Stage One.

The proposed algorithm was written in the C language and the runtime was tested on a two Windows equipped machines where the algorithm completed the processing, on average, sparing 30% of the time between UDP data packets sent from the HDL32E. In comparison to related research, this algorithm performed over seven hundred and thirty-seven times faster.

DEDICATION

To my wife, son, and mother, for all their love, kindness and support.

ACKNOWLEDGMENTS

I would like to thank my supervisor professor Dr. Mohammed Khalid for providing me the opportunity to research under him.

Marko Jovanovic, for all our technical conversations throughout the years, thank you for your guidance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS/ SYMBOLS

ABD	Adaptive Breakpoint Detector
ACC	Adaptive Cruise Control
ADAS	Advanced Driver Assistance System
AEB	Autonomous Emergency Braking
ASIC	Application Specific Integrated Circuit
BLIS	Blind Spot Indication System
BRAM	Block Random Access Memory
FPGA	Field Programmable Gate Array
FOV	Field of View
GPU	Graphical Processor Unit
GNSS	Global Navigation and Satellite System
IMU	Inertial Measurement Unit
INS	Inertial Navigation Unit
LASER	Light Amplification by Stimulated Emission of Radiation
LiDAR	Light Data and Ranging
RADAR	Radio Detection and Ranging
SAE	Society of Automotive Engineers
SLAM	Simultaneous Localization and Mapping
TDC	Time to Digital Conversion

CHAPTER 1

Introduction

1.1. Motivation

In recent years, the automotive industry has attracted great attention from the technical community, resulting in vast changes to the trade. Over the last decade, automotive start-up companies have rapidly emerged, as manufacturers race to develop advanced driver assist systems and autonomous vehicles through technological innovation. There has been an unprecedented amount of research in machine learning algorithms and artificial intelligence that is focused on object recognition through convolutional neural networks.

The automotive sensor suite presents unique challenges. For example, weather is unpredictable and can change from sunny to overcast, which can turn into rain, hail, fog, snow, or even a blizzard, sometimes within the same trip. Similarly, vehicle cabin temperatures can fluctuate from a maximum of 76 degrees Celsius [1] to the lowest recorded minimum temperature, which according to the Guinness World Records is - 67.7 degrees Celsius, in Russia [2].

Severe weather and temperature changes are not the only challenges for the electronic components and designers. Achieving the necessary speed of the electronics and data processing time are yet another great design challenge. Timing of decision-making in a vehicle can be the difference of arriving home safe to one's family or not returning home at all. For an autonomous vehicle to function safely in the public, near instantaneous decision-making is critical. There has been much research into different physical processing platforms to which will deliver the most efficient and timely decision-making process. Different hardware platforms range from multi core processors, to Graphics Processing Units (GPU) that exploit parallel processing, to Field Programmable Gate Arrays (FPGA), which benefits from both paralleling and pipelining. In order to exploit the full capabilities of these hardware platforms, appropriate algorithm development must take place.

Not all algorithms are the same. The University of Berkeley California has focused much research on algorithm development through parallel programming models deemed "the 13 dwarves", where each dwarf classifies an algorithm based on patterns of computation and shows which hardware platform would optimize the performance of execution [3].

Another challenge to the automotive sensor suite is the influx of motor vehicles in the market and the rising costs of maintenance. At inception, motor vehicles promised humankind liberating physical mobility, but now seem to be responsible for mass traffic jams, long commutes, and the need for flexible work arrangements. In 1913, the first Ford Model T rolled off the assembly line, igniting a new revolution of affordable mass-produced transportation. By the 1950's, almost every North American home had one vehicle per household, with homes in most older neighborhoods featuring a single driveway with a single car garage. In North America today, it is commonplace to have one vehicle for every person in the household who drives. Yet, a Business Insider study shows that Americans only drive a total of 300 hours per year [4]. This means, on average, each vehicle is in use only 3.4% of the time per year. Considering the expenses associated with motor vehicle ownership, including insurance, fuel, maintenance, repairs, and depreciation cost, the cost of ownership related to the infrequent amount of time spent in these vehicles is disproportionate. Perhaps, a pay per use model could be a more efficient method of spending for the consumer [36].

The need for advanced driver assistance systems and autonomous vehicles in the market can be appreciated in reviewing the results of a Stanford study, which shows that 90% of all automobile accidents are caused by human error [36]. If the human aspect could be removed from controlling the vehicle, could road collisions reduce to one tenth of current numbers? When autonomous vehicles come to fruition, a paradigm shift could occur, altering the landscape as we know it. Will modern homes have a garage or even a driveway?

For a vehicle to become fully autonomous, a sensor suite must be employed. Common approaches use RADAR, LiDAR, Cameras, and Ultrasonic sensors. Not any one of these sensors has proven to be exceptional on their own in a harsh automotive environment; but, working together, this group of sensors can function together acceptably, in any environment.

This thesis research specifically focuses on LiDAR point cloud processing segmentation and shape extraction.

1.2. Related Work

Research in the area of automotive LiDAR is rapidly expanding, with vehicle manufacturers competing to develop autonomous vehicles. New algorithms and white papers are published frequently with findings, improvements and better methods.

For this research, several algorithms were implemented in order to achieve the completed process, from the unpackaging of the data packet provided from the LiDAR sensor, to the process of shape extraction from clusters.

In [5], L. Zheng and Y. Fan propose an integrated circuit and system design for the Velodyne VLP-16 to decode the 3D LiDAR information transmitted over UDP.

In [6], Tobias Nystrom Johansson & Oscar Wellenstam proposed a pre-single line shape extraction, which later merges the single lines into clusters and extracts the final shapes. This research was conducted using MATLAB. The path Johansson and Wellenstam have taken shows promise to an algorithm implementation that needs to be accelerated.

In [7], Su-Yong An, Jeong-Gwan Kang, Lae-Kyoung Lee, and Se- Young Oh, introduce an ABD in two dimensions. The adaptive breakpoint detector varies the threshold depending on distance from the LiDAR sensor.

In [8], Kim, Beomseong & Choi, Baehoon & Yoo, Minkyun & Kim, Hyunju & Kim, Euntai introduce an algorithm for segmentation using a multi-layer LASER scanner. The algorithm utilizes a candidate set for testing LASER points. The study utilized the same ABD method in [7].

1.3. Scope and Limitations

This paper's research area focuses on the time to unpack the UDP data packet, process segmentation clustering and perform shape extraction of LiDAR data. The LiDAR sensor data selected for this research was the Velodyne HDL 32E. Recorded HDL 32E LiDAR data was obtained from the download sections of Velodyne website. These algorithms were tested with a limited scope of prerecorded data from what was readily available from the Velodyne website. If the proposed algorithm was operated with a physical hardware LiDAR sensor, additional processing time may be required.

Simultaneous Localization and Mapping (SLAM) of the points was not considered in the scope of this research. The sample of recorded LiDAR data was purposely chosen to be flat to minimize the impact of not recalibrating the points for vehicle movements. As this research will only focus on the throughput and processing of the LASER data, only Velodyne HDL 32E UDP data packets from port 2368 will be considered in this research.

The shapes extracted from this research could be furthered processed in the LiDAR sensory chain utilizing Bayesian tracking [35].

1.3.1. Research Area

The image below illustrates a typical path for processing LiDAR data in an automotive environment. The red boxes highlight the area of research focus in this thesis.

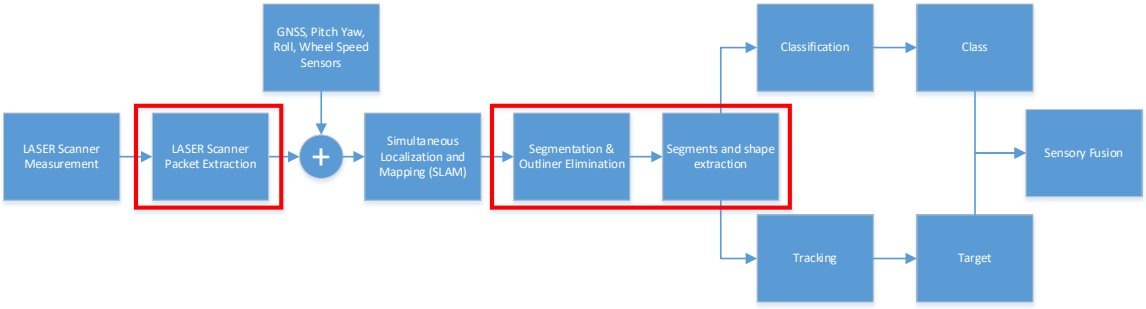

Figure 1-1-1 An example of a LiDAR processing path in an automotive environment

1.4. Contribution

The primary contributions of this thesis are:

- A method for parsing ethernet UDP LiDAR packets provided by Velodyne HDL 32E LiDAR sensor that organizes data by vertical and azimuth angle
- A modified method for clustering data when organized in a vertical and azimuth angle
- Adaptation of the two-dimensional adaptive breakpoint detector clustering algorithm to incorporate detections in three dimensions
- A modified Graham scan algorithm that exploits semi-preorganized data
- Timing analysis comparison of algorithms and methods

1.5. Thesis Outline

This thesis is assembled as follows: Chapter 2 introduces the autonomous and ADAS features found in automotive vehicles today. Chapter 2 expands to an overview of the current sensors used in the automotive industry to achieve these advanced features. Chapter 3 provides a system overview of the hardware and processes used in this research, as well as methods. Chapter 4 provides a thorough description of the implementation of the LiDAR data obtainment, algorithms used and methods of implementation. Chapter 5 provides manual evaluation of algorithms and the timing analysis

results. Chapter 6 provides a discussion of the tuning parameters needed for the complete algorithm and future work. Chapter 7 presents the conclusion.

CHAPTER 2

Background

2.1. Automotive ADAS & Autonomous Introduction

ADAS are features within the vehicle that aids the driver in vehicle navigation, safety, and alerts. The level of how much control the feature has on the vehicle is defined by the level of autonomy standardized by the Society of Automotive Engineers (SAE) [9]. Figure 2-1 illustrates how the SAE has defined six levels of automation, level zero being no automation and level five being a fully automated vehicle.

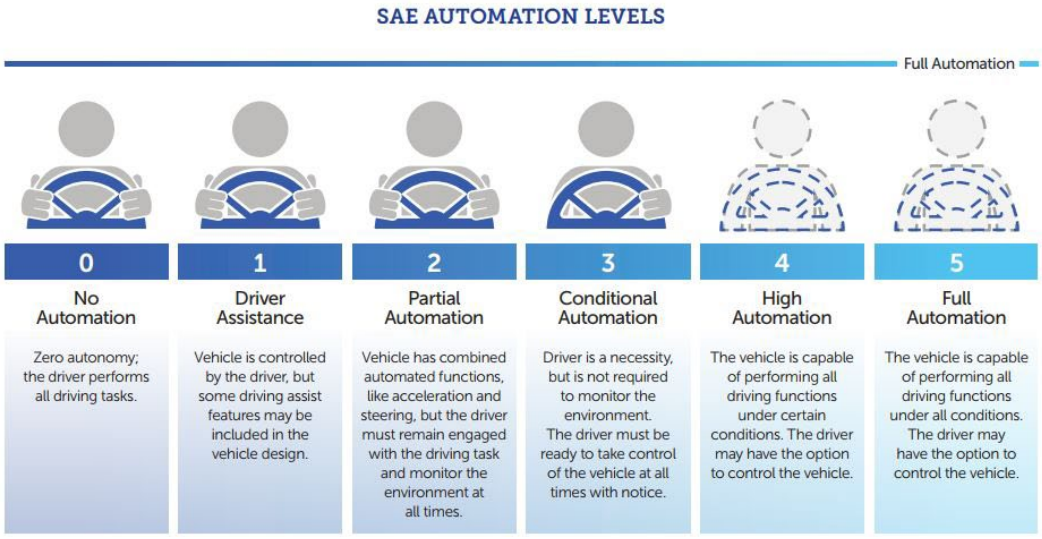

Figure 2-1 The six levels of autonomy standardized by the Society of Automotive Engineers, taken from [9]

Level two and three automation features can be found in today's automobiles.

Common ADAS features utilized in vehicles today include:

- Adaptive Cruise Control (ACC) – ACC will increase or decrease velocity of the vehicle and apply the brake if an obstacle is in front of the vehicle. ACC allows the vehicle to maintain a velocity in traffic based on distance. ACC main hardware components are front facing camera and the vehicle front fascia mounted radar module.
- Autonomous Emergency Braking (AEB) - AEB will automatically apply the brakes on the vehicle when an object located in front of the vehicle comes within a critical stopping range. The AEB features rely on the front facing radar module.
- Park Aid – Park Aid allows the vehicle to park itself, in a parking lot, or parallel park the vehicle on the street. The Park Aid feature relies on ultrasonic sensors.

- Lane Centering – The Lane Centering feature allows the vehicle to steer itself, keeping the vehicle between the lane markings on the road. The Lane Centering features utilizes the front facing camera for visualization.
- Road Sign Detection - The Road Sign Detection feature locates roadway speed signs and displays them to the driver on a human machine interface, eliminating the question of what the speed on a roadway is. The Road Sign Detection feature utilizes the front facing camera for sign detections.
- Blind Spot Indicator System (BLIS) – The BLIS feature alerts a motorist of approaching objects or vehicles in the motorist's blind spots. The BLIS feature utilizes RADAR modules located in the rear fascia corners of the vehicle. A yellow light in the side mirrors has become common place for a warning to motorist that an object exists in the blind spot.
- Pedestrian Detection - The Pedestrian Detection feature alerts the driver to a pedestrian suddenly walking out in front or behind a vehicle, which is optimal in urban areas. The pedestrian detection systems rely on ultrasonic sensors for detection and cameras for object classification.

2.2. Introduction to Sensor Suite Found in the Automotive Industry

For a fully automated vehicle to work seamlessly, a network of different sensor technologies is employed for the task. The most common sensors found in the automotive industry today are as follows:

2.2.1. Ultrasonic Sensors

- Excellent for detecting objects in the immediate vicinity
- Objects generally detected include; automobiles, pedestrians, curbs, etc.
- Ultrasonic sensors play an important role in Park Aid and automated parking
- Effective sensor range is approximately 2 meters

2.2.2. Image Sensors (Camera)

- Traffic sign detection
- Lane departure/ markings
- Object identification i.e. other vehicles
- Detection of colours and fonts
- Sensor range approximately is 120 meters

2.2.3. RADAR Sensors

- Different range radars sensors are placed on the exterior of the vehicle
- Short and medium range radar sensors are used for blind spot vehicle detection
- Long range radar can be used for ACC & AEB to identify vehicles' frontal distance

- Radar modules can track the speed of other vehicles in real time
- Sensor Range is approximately 250 meters

2.2.4. LiDAR Sensors

- LiDAR sensors can accurately identify objects
- Usually paralleled with other sensors' data for confirmations
- Sensor Range is approximately 200 meters

2.2.5. Cloud

- As the previously described sensors cannot reach beyond 250 meters, the autonomous vehicle can use information from the cloud to predict maneuvers
- Receive traffic congestion reports
- Sensor range is greater than 250 meters

2.2.6. Comparison of Sensor Technologies

Figure 2-2 illustrates how not one single sensor is appropriate for every task that an autonomous vehicle would need to achieve to be self-sufficient [10].

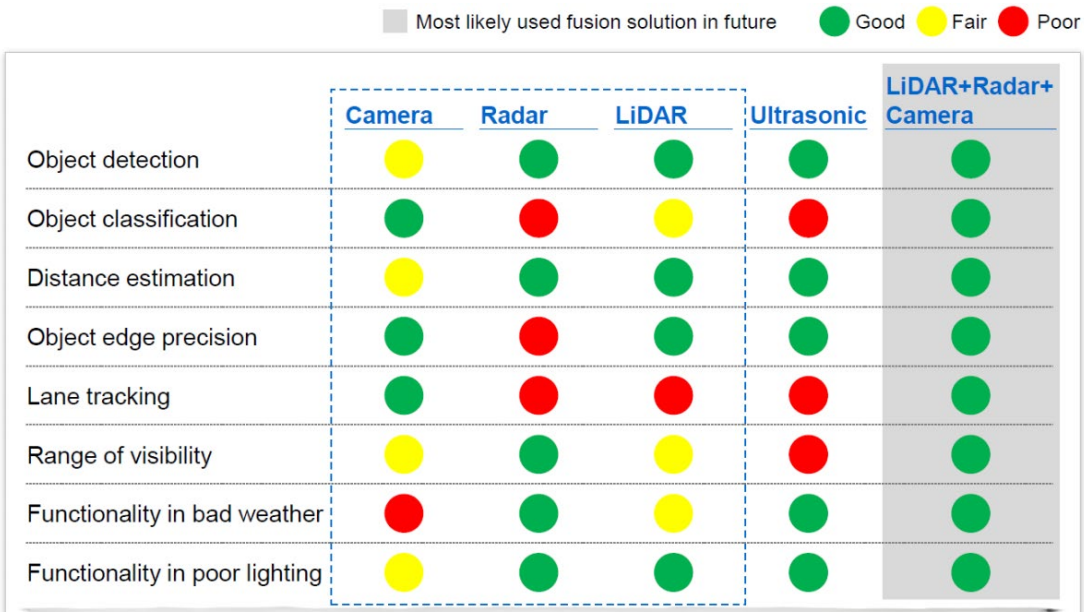

Figure 2-2 Comparison of performance levels of different automotive sensors, taken from [10]

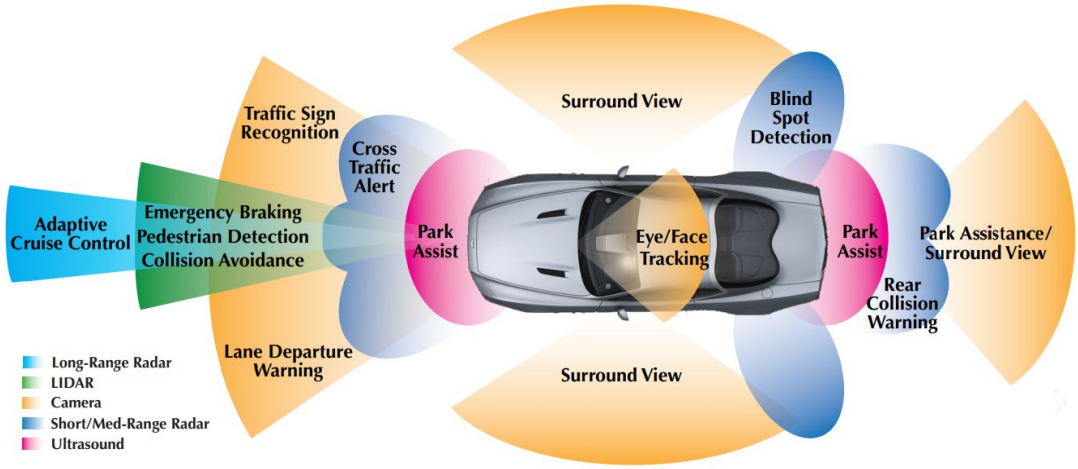

Figure 2-3 Topology of a common sensor suite in today's ADAS vehicles, taken from [10]

2.3. Hardware Principle of LiDAR

LiDAR is an active remote sensor that uses light as its medium that can be used for distance measurement, detecting objects, and position reference. LiDAR sends a LASER beam of light from the LiDAR device, which receives said beam. The time duration between transmission and receipt can provide distance information. The beam that is emitted by the LiDAR device is extremely focused and can detect small objects with precision. Commercial LiDAR devices is a relatively new technology made commercially available in 1995 [11].

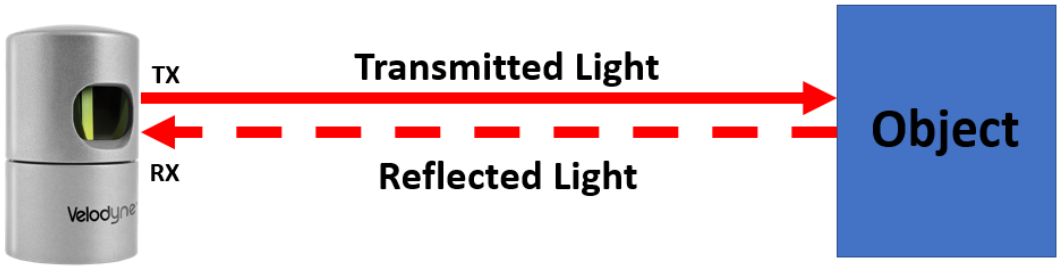

Figure 2-4 Example of a LiDAR sensor transmitting and receiving a LASER

As the LASER must travel to and from the object, the speed of light multiplied by time of flight must be divided by two to obtain distance.

$$Distance\ of\ Object = \frac{Speed\ of\ Light \cdot Time}{2}$$

Equation 1, LiDAR distance to object formula

2.4. Components of LiDAR

2.4.1. Mirror Systems

There are two well-established types of LiDAR systems manufactured currently - nodding and polygonal mirror systems. The nodding mirror type of sensor features an optical rotary encoder on the mirror pivot mount for precise feedback. The "nodding" of the mirror creates the field of view (FOV) for the vertical axis.

To sweep the LASER sensor through the horizontal axis or azimuth, the nodding mirror system base must be mounted to another motor, where another optical encoder is mounted for precise feedback. The LiDAR LASER sensors are rotated at 600 - 1200 revolutions per minute (RPM) which creates an accurate point cloud [12].

This paper will only focus on the nodding mirror type, which is used in the Velodyne HDL 32E.

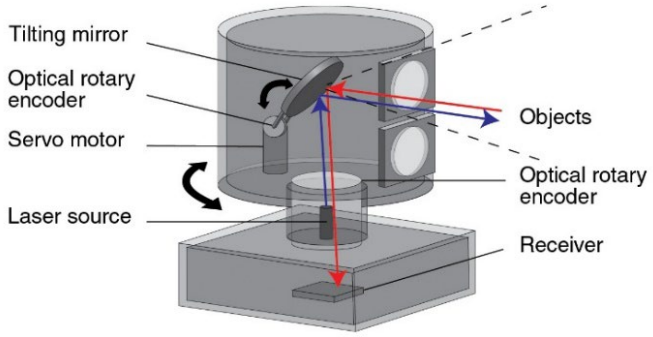

Figure 2-5 Example of a nodding mirror type LiDAR sensor, taken from [12]

Figure 2-6 Example of a polygonal type LiDAR sensor, taken from [13]

2.4.2. Detector

The photodetectors internal to the sensor use silicon avalanche photodiodes to convert light energy (photons) into an electrical signal [14].

2.4.3. Timing Electronics

Critical timing of the LASER pulses is essential to accurate measurement. Two methods for the critical timing capture are either a high-speed analog to digital converter (ADC) or a time-to-digital converter (TDC). Approaches to a TDC include using a dedicated microprocessor or microcontroller with a FPGA for logic control. The TDC application has become so popular that dedicated integrated circuits (IC's) are now manufactured [15].

2.4.4. Optical Design

The minimum sensing distance FOV must overlap the transmitted and the received LASER in the lenses, defining the minimum sensing distance. The optical design will need to focus the energy on the photo diodes' active area in order to maximize the energy received. Further development of the optical system is needed to prevent the LASER from divergence which would drastically reduce sensing distance. The parameter to correct this is the astigmatism of the Light Emitting Diode (LED). A corrected system is referred to as collimated [15].

2.5. LiDAR LASER Wavelengths

LiDAR utilizes different LASER wavelengths depending on the application. The wavelengths seen in the industry commercially available today range between 532 nm to 1550 nm.

LiDAR wavelengths and use cases can be summarized into three different categories.

1. Bathymetric application - 532 nm wavelength to penetrate the water way and measure the seafloor which is blue green in the visible light spectrum
2. Automotive application - 905 nm wavelength for obstacle detection which is infrared non-visible
3. Topography mapping - 1500 nm wavelength for topology mapping of land which is infrared non-visible

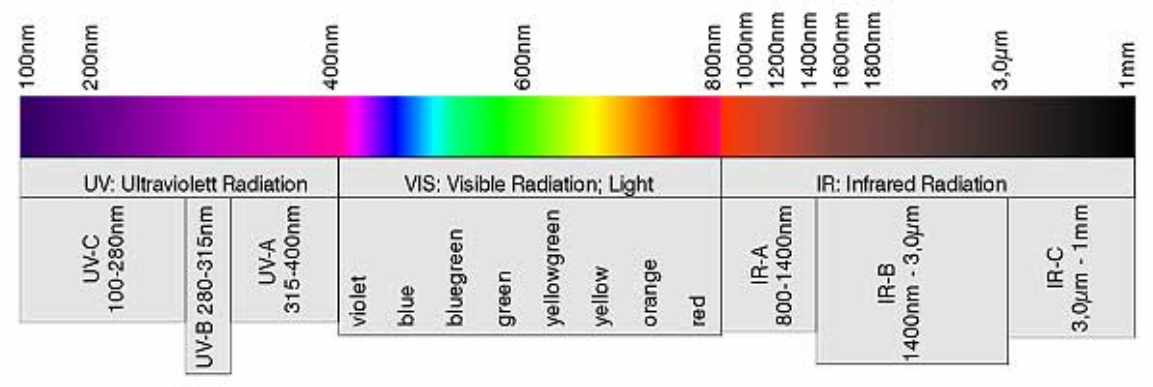

Figure 2-7 Wavelength range of optical radiation, taken from [16]

Solving for the frequency of LASER emitted for automotive at 905 nm shown below

$$f = \frac{C}{\lambda} = \frac{3 \cdot 10^8 m/s}{905\ \text{nm}} = 31.58 \cdot 10^{15} = 31.58\, pHz$$

Equation 2, LiDAR frequency from wavelength formula

Water absorption is the main factor for use case and wavelengths. [17] found that LiDAR sensors that use a wavelength of 1550 nm exhibited a water absorption rate more than twice as great as a LiDAR sensor that uses 905 nm. The result of a high-water absorption rate is a degraded point cloud in adverse weather conditions such as rain, fog, and snow.

2.6. LASER Safety

"All commercially sold LiDAR products must achieve eye-safety certification via compliance with the U.S. Food & Drug Administration eye-safety performance standard which conforms with the International Electrotechnical Commission ("IEC") 60825 standard. If sensors are designed to meet eye safety standards, both wavelengths can be used safely." [18]

2.7. Detection Schemes

Early model LiDAR sensors could only detect one object per LASER pulse. Late model LiDAR sensors can now detect multiple returns per single LASER pulse. A LiDAR sensor mounted on an airplane scanning overtop of a forest could reflect the LASER off the leaves several times before hitting the ground. Image 2-8 illustrates a LASER pulse which picks up four returns - three branches of the tree and the ground. Recently, topological LiDAR scans have revealed ancient hidden cities for archeologists.

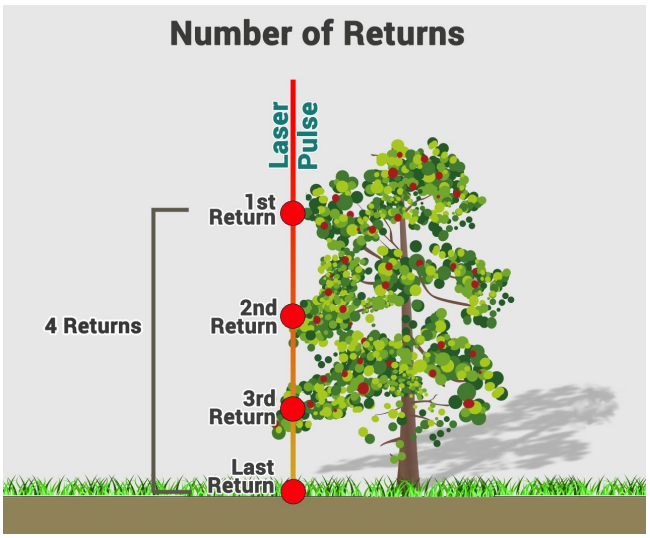

Figure 2-8 Multiple LASER returns from a single LASER fire, taken from [19]

2.8. Point Cloud

A point cloud is the image that the LiDAR sensor creates by scanning the surrounding area. The point cloud is made from a collection of individual LASER pulses. LiDAR data is received as polar or spherical coordinates with known angle, beam intensity and return distance. The LASER pulses from a LiDAR sensor can be thought of as a collection of points in Cartesian two dimensional (x, y) or three dimensional (x, y, z) coordinates. Image 2-9 illustrates a point cloud from a rotating three-dimensional multilayer LIDAR sensor.

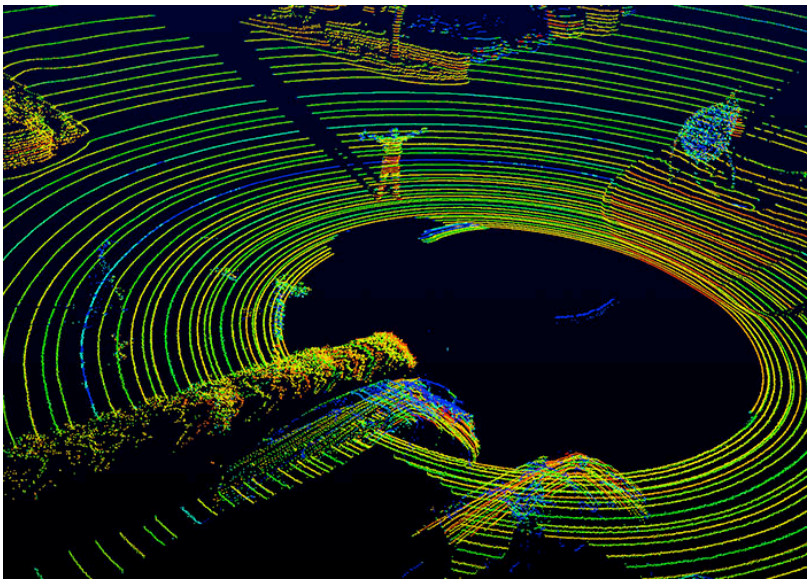

Figure 2-9 A point cloud from a rotating 360-degree LIDAR sensor, taken from [20]

2.9. GPS and IMU

Typical combination in LiDAR sensory fusion applications such as simultaneous localization and mapping (SLAM) utilize Global Navigation Satellite System (GNSS) with an inertial navigation system (INS). The INS houses the inertial measurement unit (IMU) system, which can help guide and keep track of the vehicle's pose through any terrain as the vehicle comes in and out of satellite communication.

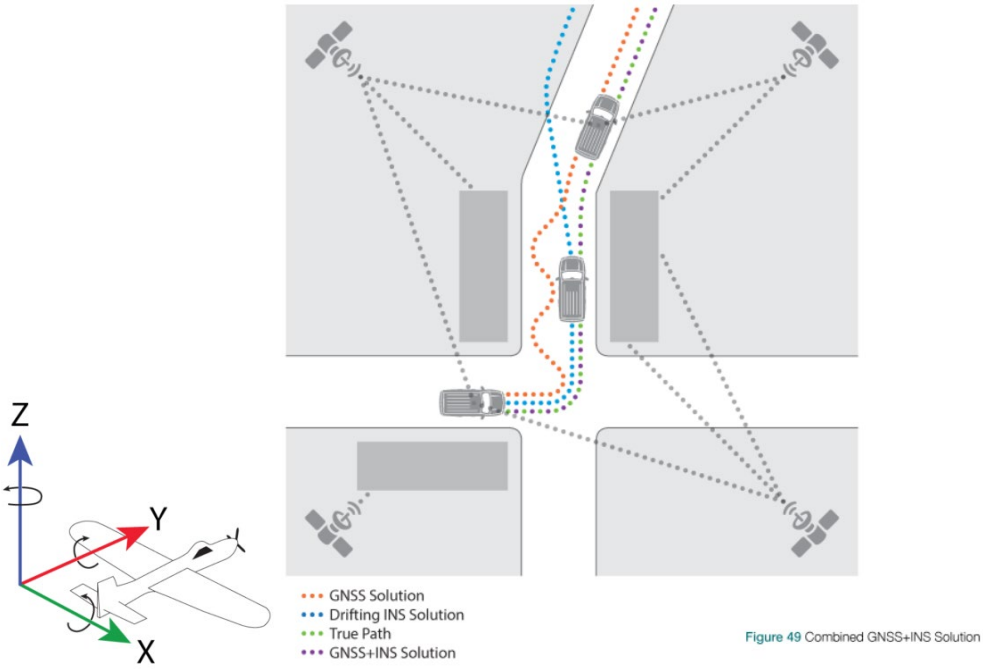

Figure 2-10 Illustration of a global navigation satellite system used in an automotive application, taken from [21]

2.10. Simultaneous Localization and Mapping (SLAM)

Although SLAM will not be covered in this research, a general perspective will be described, with the importance.

SLAM is a method for mapping the surrounding area while keeping track of the vehicle or robot's position. In a perfect environment, a vehicle would always travel straight, and the road would always be perfectly flat. In the real world, the road curves, hills produce inclines or declines, highways bank their curves and roadways are designed with a drainage gradient in mind. These imperfections add greatly to the challenge of determining the vehicle's location relative to its surroundings. Knowing exactly how the vehicle is situated is known as the robot's pose. Figure

2-10 illustrates a GNSS system and how the vehicle would operate with its onboard odometry. Today, a vehicle has many odometry systems already in place to aid in the robot's pose, including a restraint control module which constantly measures the vehicle pitch, yaw and roll; wheel speed sensors which determine actual distance travelled; and, onboard global positioning systems as well as GNSS systems. These systems all work together to provide an accurate robot pose estimation. To illustrate this concept with Velodyne HDL32-E (the LiDAR Sensor used in this research), Figures 2-11 and 2-12 below reveal the potential problems.

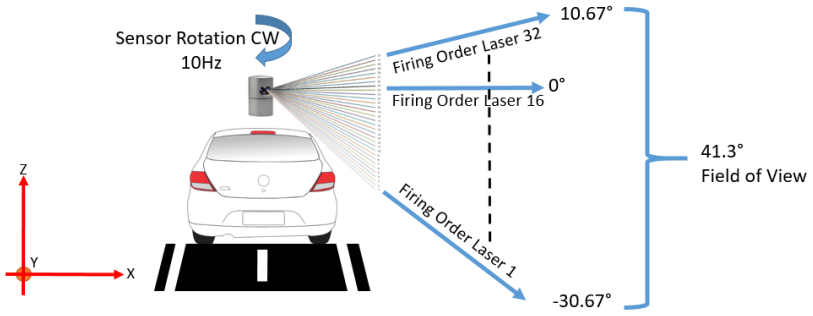

Figure 2-11 Both the vehicle and roadway are on level in relation to axis

Figure 2-11 the above image illustrates how, when everything is level, the conversion from the LiDAR sensors spherical coordinates to Cartesian is straightforward and transfers over. One can notice the 0-degree angle provided by the LiDAR sensor is parallel to the compared Cartesian X axis.

However, when the vehicle is driving through banked turns, inclines, declines or over bridges, constant compensation of all data points must be considered and compensated to map correctly. Sudden movements of the vehicle will result in sudden movements of the onboard sensors. The sudden movements in LiDAR could cause distorted segmentation and reality if the LASER returns are not calibrated.

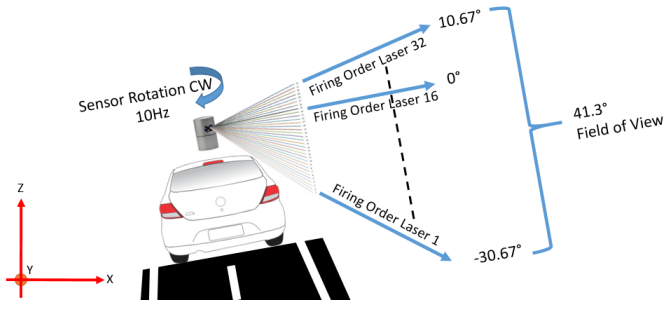

Figure 2-12 Both the vehicle and roadway are unlevel in relation to axis

Figure 2-12 the above image illustrates a vehicle driving in a banked turn where one can notice the 0-degree angle provided by the LiDAR sensor is inaccurate compared to the Cartesian axis.

The SLAM problem becomes a greater challenge as the robot or vehicle is moving in three dimensions dynamically, as LiDAR, Radar, and Ultrasonic Sensors would produce invalid size and or distance of objects with respect to the environment.

Common algorithms used to the SLAM problem include and are not limited to particle filter, extended Kalman filter, Covariance intersection, and Graph SLAM [22].

CHAPTER 3

System Overview

3.1. Hardware Equipment for Processing

Chapter 3 describes the hardware and equipment used in the development of this research and testing. In addition to the hardware listed in this chapter, an additional Windows machine was used for execution run time testing of the final algorithm, as described in Chapter 5.

3.1.1. Computer for Processing and Development

- Dell Latitude E5570
- Intel I7- 6820HQ CPU at 2.7 GHz Processor
- SSD hard drive model SK Hynix SC311 SATA 512 GB
- RAM 16 GB

3.1.2. Computer Operating System

- Windows 10 Enterprise Edition 64 bit

3.1.3. Software Compiler

- Code Blocks 17.12

3.2. LiDAR Hardware

3.2.1. Lidar Sensor

- Velodyne HDL32-E

The LiDAR sensor used in this research is the Velodyne HDL-32E. The HDL-32E can create a three-hundred-and-sixty-degree point cloud image. The creation of this three-dimensional point cloud image is made possible by using thirty-two vertically aligned LASER emitter and detector pairs, placed in a rotating nodding mirror designed housing to capture the surrounding environment.

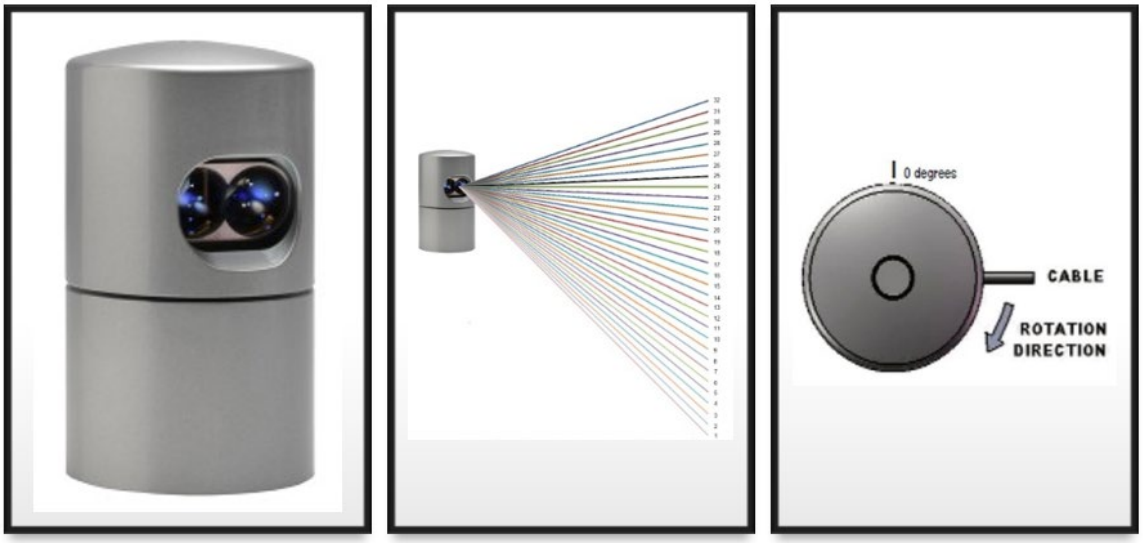

Figure 3-1 Velodyne HDL 32E LiDAR sensor, taken from [23]

3.2.2. Velodyne HDL32E Specifications

Velodyne HDL 32E [23]

- Maximum number of scans per second is 700,000 LASER returns
- 100 MB/s Ethernet Connection, UDP packets
- Data packet 0.55296 ms or 1808.45 packets/second
- 32 LASERs, Class 1 Eye Safe
- 10 Hz Frame Rate
- Accuracy +/- 2 cm
- 360° Horizontal Field of View (FOV)
- 41.3° Vertical Field of View (FOV)
- Usable returns up to 70 meters

3.2.3. Azimuth Resolution Calculation

The Velodyne HDL 32E can fire one independent LASER at a time, as every LASER must be transmitted and received before firing the next LASER. The sequence of LASER fires happens while the outside case of the HDL 32E rotates at 10 Hz or 600 rpm. For accurate recreation of the point cloud, the angular rotation of each LASER must be added to the rotational byte which is obtained from the start of every 32 LASER firing sequence within the UDP data packet.

$$Azimuth_{Resolution} = \frac{rev}{min} \times \frac{1}{60} \frac{min}{s} \times 360 \frac{degrees}{rev} \times \frac{46.08 \times 10^{-6}\ s}{firing\ cycle}$$

Equation 3, Velodyne HDL 32E azimuth resolution calculation

Equation 3 may be simplified as follows:

$$Azimuth_{Resolution} = \frac{0.166\ degrees}{firing\ cycle}$$

Equation 4, Simplified Velodyne HDL 32E azimuth resolution formula

3.2.4. Rotational Speed and Field of View

Figure 3-2 illustrates a hypothetical pictorial view of a mounted Velodyne HDL 32E on an automotive vehicle. The rotational speed of the HDL 32E of this research is 10 Hz or 600 revolutions per minute. The field of view (FOV) of 41.3 degrees is made possible from the 32 vertical firing LASERs.

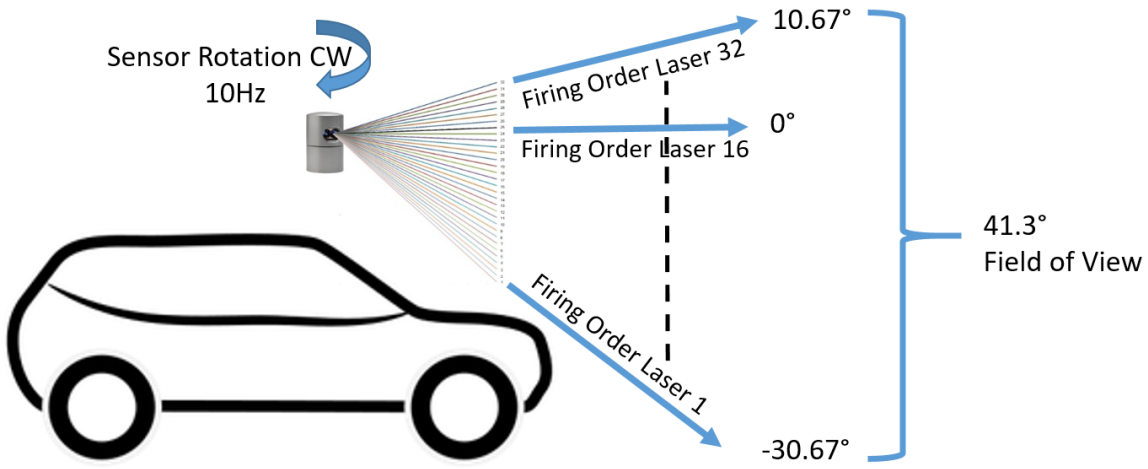

Figure 3-2 Field of view and rotation of the Velodyne HDL 32E

3.3. Processing Time Benchmark

The Velodyne HDL 32E LiDAR sensor sends a new UDP data packet every 552.96 µS. For the processing time to be useful in a real-world application, all LiDAR UDP packet data processing must be completed before the following UDP data packet arrives. The time between UDP data packets will serve as a benchmark for evaluating the algorithms in this research. Note that this research only covers a portion of all the software processes needed in an actual vehicle LiDAR

system. Additional algorithms such as SLAM would need to be added to the processing time of the algorithms in this research. The processing time in this research will need to be considerably less than the time duration between UDP packets in order to accompany the additional algorithms needed for a complete LiDAR processing design. Figure 3-3 is an illustration of the timing benchmark of UDP data packets sent from the HDL 32E, showing the LiDAR data must be processed before the arrival of the next packet.

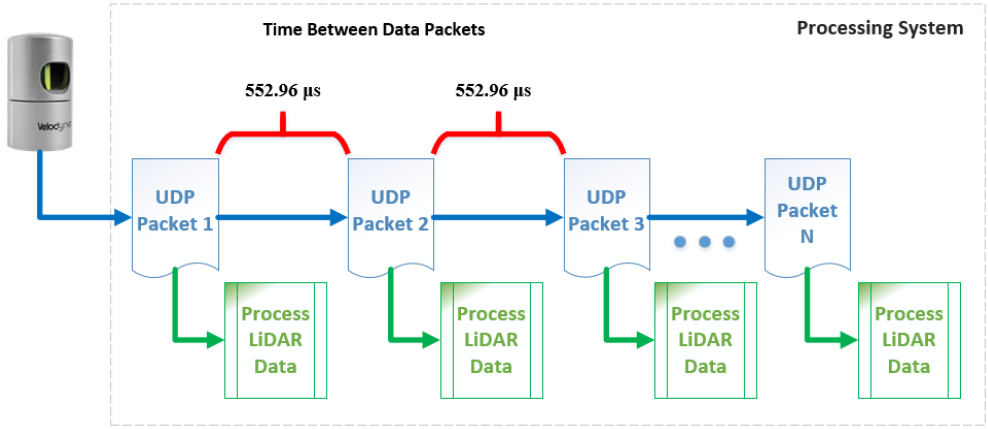

Figure 3-3 Velodyne HDL 32E UDP data packet timing delivery benchmark

3.4. Vehicle Coordinates

The vehicle coordinate system used throughout this research is shown in Figure 3-4, where the positive X axis is in front of the vehicle, the positive Y axis is to the right side of the driver and the positive Z axis is straight up.

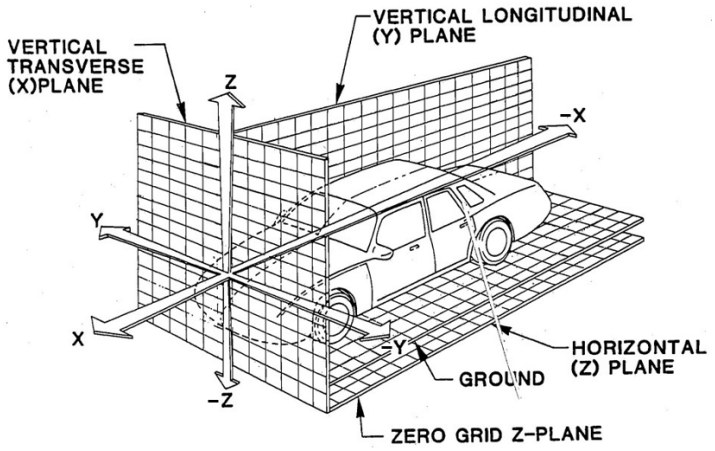

Figure 3-4 Vehicle coordinates, taken from [24]

CHAPTER 4

Implementation

4.1. Three Stage Process

The implementation of the LiDAR data processing can be subdivided into three stages. The coding of this research project contains three function calls from the main program. These three function calls are independent of one another but dependent on the work of the previous function. The three stages of the code are as follows:

Stage One - Packet Parsing

- Organizes and parses the LiDAR UDP data packet
- Removes any erroneous data
- Completes the ground point removal
- Conversion of spherical coordinates to Cartesian
- Stores data in an organized memory array

Stage Two - Data Clustering

- Computes adaptive breakpoint detection
- Evaluates LASER returns beyond nearest neighbors for a cluster match
- Assigns every valid LASER return a cluster number
- New cluster numbers are assigned if the LASER return does not find a cluster match

Stage Three - Shape Extraction

- Every Cluster is evaluated and assigned a shape type
- There can only be 4 outcomes as there are four shapes
    - 1 = Point, 2 = Line, 3 = L Shape, 4 = Convex Hull
- A Cluster with less than one point is automatically classified as a point (Shape 1)
- Principal component analysis is performed to find Line shapes (Shape 2)
- Clusters that have not been assigned a shape number are tested for the L shape (Shape 3)
- If the L shape test fails, the cluster must be a convex hull (Shape 4)
- Convex hull processing is performed on the remaining clusters that failed the L-Shape test

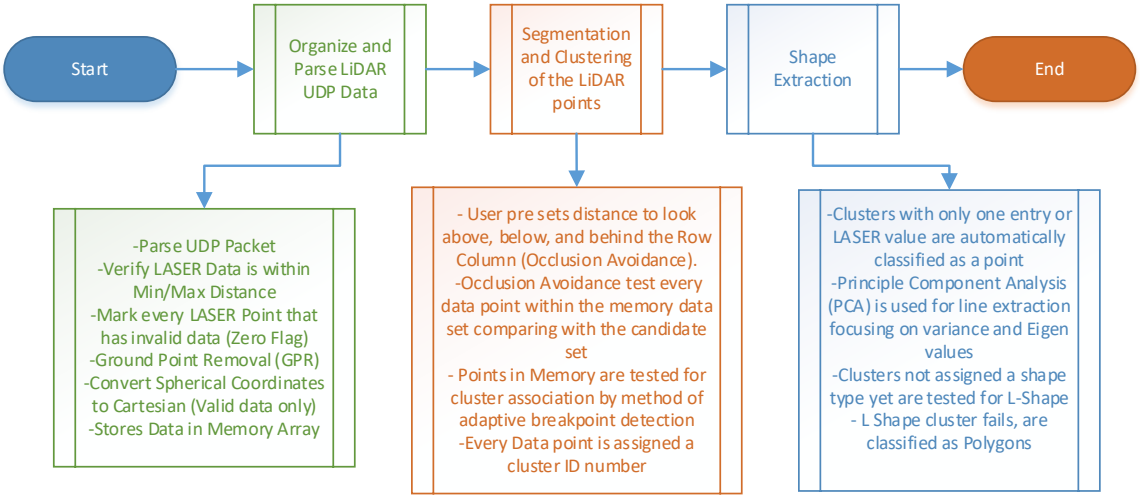

Figure 4-1 Visual representation of the 3 function calls from the main program software

4.2. Preliminary Work: Obtaining the LiDAR Data

Velodyne has pre-recorded LiDAR files readily available on their website. The pre-recorded LiDAR files come in .pcap file format. The data set used in this research was labeled "HDL32-V2_R into Butterfield into Digital Drive.pcap" link available from the Velodyne website where the file is stored at data.kitware.com [25].

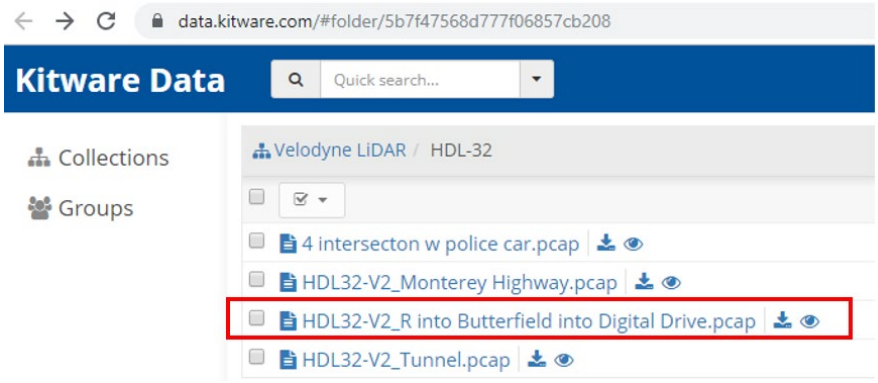

Figure 4-2 Velodyne pre-recorded LiDAR .pcap file download

Velodyne was contacted to request sensor mounting height and vehicle type. Velodyne confirmed that the HDL32-E sensor was mounted on a Ford Fusion vehicle with the sensor mounting height being ~1.73 m. The mounting height of the sensor was crucial for setting up the ground point removal process.

The data set “Butterfield into Digital Drive” .pcap file contains 89,691 UDP data packets. There are two types of data packets produced by the Velodyne HDL-32E. The first data packet located on port 2368 contains the LASER firing data. The second packet located on port 8308 contains the GPS and positioning data. This research will only focus on the throughput and processing of the LASER data packets from port 2368.

4.2.1. Pre-Processing the LiDAR data

The program Wireshark was implemented to open the .pcap file and remove all the 8308 port files. The custom data set of the first 200 UDP packets from port 2368 were saved a “.C” Array. By having all the LASER data packet information available in the computer memory, a true performance speed test of the algorithms can be measured without throttling.

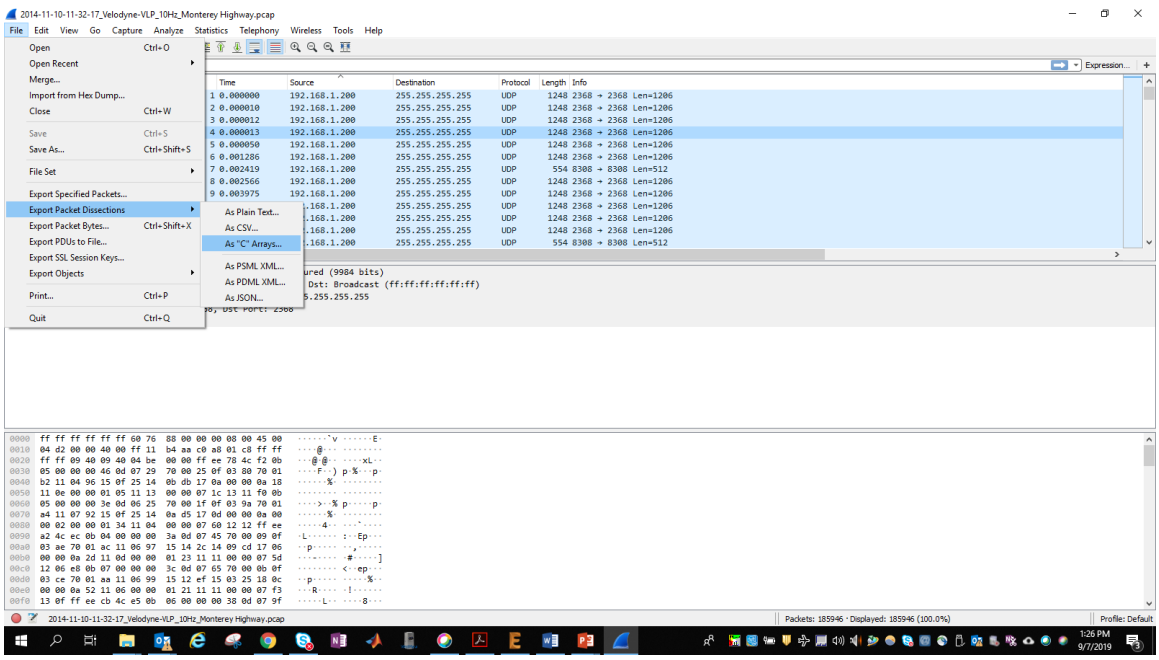

Figure 4-3 Wireshark exporting a C file array

4.3. Stage One – Packet Parsing

In the first stage, the objective is to arrange the LASER data returns vertically and horizontally, as the human eye would view the environment, while removing any invalid data. Invalid data is any LASER returns beyond the max distance of the LiDAR sensor (70 m) or any return below the minimum distance of the LiDAR sensor (1 m). Any LASERs that face at a negative angle compared

to the horizon maximum distance will be reduced, as the negative angle LASERs beam will hit the ground before ever reaching their maximum capable distance. The maximum distance will be computed based upon vehicle height for all downward facing LASERs. If the LASER return is deemed to have hit the ground, that LASER data will be considered invalid data. This step is called ground point removal (GPR). If the LASER return data falls in the range of being greater than the minimum distance and less than the maximum distance, Cartesian coordinates will be computed and all LASER data will be stored in memory for further processing.

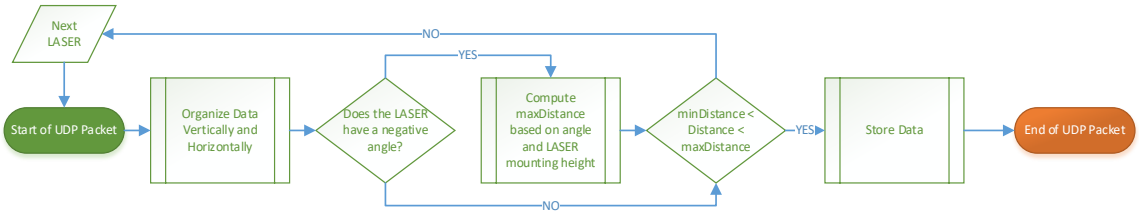

Figure 4-4 Stage 1 algorithm flow chart

4.3.1. Data Storage

Data storage for the processing of LiDAR points is stored in a static array, 32 rows high by 960 columns wide. The array is 32 rows high to allow a location for every LASER vertically for the Velodyne HDL 32E, which has thirty-two LASERS. Every UDP packet sent from the HDL 32E has 12 azimuth sets of 32 vertical LASER fires, meaning that every UDP packet has 384 pieces of LASER return data. With the rotational velocity of the LIDAR sensor being 10 Hz, every data packet will cover ~4.87° of azimuth rotation. A full three-hundred-and-sixty-degree rotation will require ~74 UDP Packets in order to complete a scan of the surrounding environment. With every rotating pass, the stored data must be overwritten to update objects and movements of both the vehicle and its surroundings. To provide a slight buffer for overlap, the static memory is expanded from 74, to be able to hold 80 UDP packets of data. Eighty packets multiplied by 12 LASER fires per packet results in an array that will hold 960 Columns. Every location within the array stores vital data about the LASER return such as a zero flag, distance, rotational angle, intensity, X, Y, Z Cartesian points, and cluster number. The vertical angle is not stored, as the vertical angle corresponds to the row value of the array, where “Row 0” holds the most negative vertical angle and “Row 31” holds the most positive vertical angle (Low to High, sequentially). An implementation of the SLAM algorithm would require storing the vertical angle as the movements from the vehicle will continuously require the adjustments of points. In this research, memory allocation is aimed at static memory and not dynamic, as the end goal should be to move this

research to a FPGA or a GPU. FPGAs deal with static memory more efficiently as the data can be stored on board in the block random access memory (BRAM).

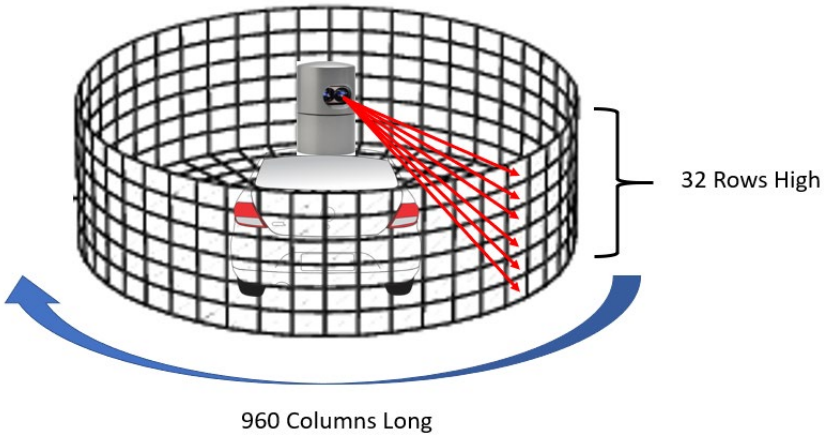

Figure 4-5 Visual representation of the memory array, images used from [26]

4.3.2. HDL 32E Usable Data Range

Velodyne states that LASER's returns less than one-meter or returns greater than seventy meters is unusable data for the HDL-32E [23]. Any LASER return that falls into the unusable category immediately has a zero flag which is set equal to one (1 = no data, 0 = data). In further processing steps, the zero flag is always examined first, and the data is ignored for further processing during subsequent steps if the zero flag is equal to one.

The image below illustrates the Velodyne HDL-32E usable data and non-useable data regions. Note that a distance threshold greater than a meter for usable data would likely be used in order to clear the vehicles hood and body components, dependent on mounting position.

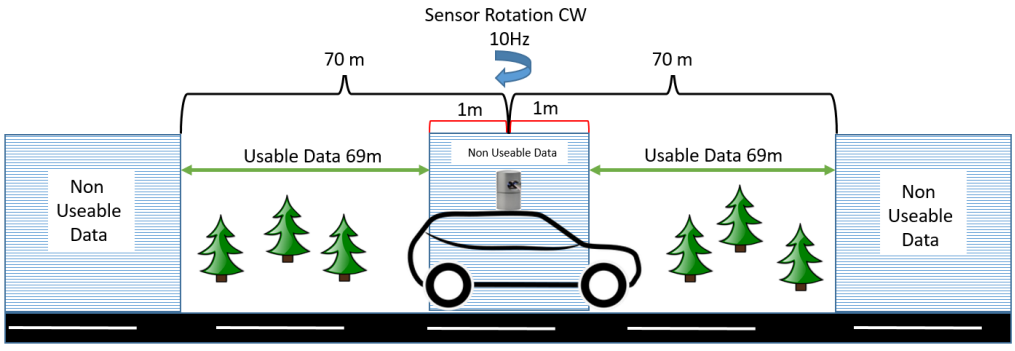

Figure 4-6 Visual representation of the Velodyne HDL-32E usable data range, images used from [26]

4.3.3. Ground Point Removal (GPR)

The physical mounting height of the LiDAR sensor and the downward firing angle of the LASERs are necessary for calculating the distance to the ground. The ground serves as nonvaluable data and it is desirable to eliminate any LASER return data that is associated with the ground, to reduce algorithm processing time. Image 4-7 illustrates a hypothetical image of the two lowest firing LASERs reflecting off the ground before they would reach the maximum potential distance of 70 m (the maximum sensing distance for the Velodyne HDL32-E).

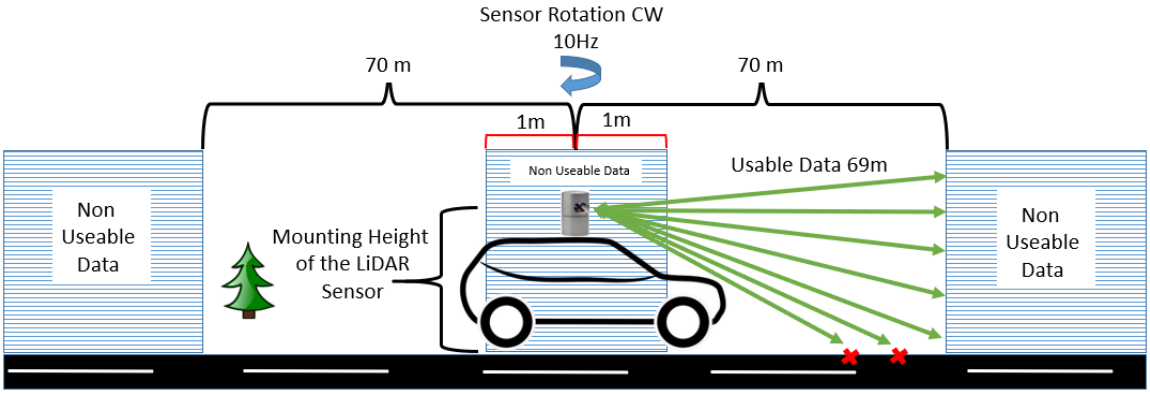

Figure 4-7 Visual representation of ground point removal, images used from [26]

The selected data set "HDL32-V2_R into Butterfield into Digital Drive.pcap" has a known sensor mounting height of 1.73 meters. The maximum distance related to angle can be calculated using the following formula.

$$Laser\ Max\ Distance = \frac{Mounting\ Height\ of\ LiDAR\ Sensor}{\sin(|Laser\ firing\ angle|)}$$

Equation 5, GPR LASER max distance formula

Applying Equation 5, the following chart calculates the maximum distance achievable based upon the LASER downward firing angle. The produced values "LASER Max Distance" is utilized in the processing for ground point removal, where, if the LASER return is greater than the computed value, it is ignored. This method will work with extremely flat circumstances; however, this method would not be the most robust with hills. The chart below only shows Laser Firings from 0 degrees to -30.67°, as positive vertical angles are expected to be able to reach the maximum LASER sensing distance (70 m).

Velodyne Laser Firing Order	DSR #	Vertical Angle (Degrees)	Radian Angle (Radians)	LASER Max Distance (Meters)
16	15	0	0	70
14	13	-1.33	-0.023212879	70
12	11	-2.67	-0.046600291	37.14
10	9	-4	-0.06981317	24.80
8	7	-5.33	-0.093026049	18.62
6	5	-6.66	-0.116238928	14.92
4	3	-8	-0.13962634	12.43
2	1	-9.33	-0.162839219	10.67
31	30	-10.67	-0.186226631	9.34
29	28	-12	-0.20943951	8.32
27	26	-13.33	-0.232652389	7.50
25	24	-14.67	-0.256039801	6.83
23	22	-16	-0.27925268	6.28
21	20	-17.33	-0.302465559	5.81
19	18	-18.67	-0.325852971	5.40
17	16	-20	-0.34906585	5.06
15	14	-21.33	-0.372278729	4.76
13	12	-22.67	-0.395666141	4.49
11	10	-24	-0.41887902	4.25
9	8	-25.33	-0.4420919	4.04
7	6	-26.66	-0.465304779	3.86
5	4	-28	-0.488692191	3.68
3	2	-29.33	-0.51190507	3.53
1	0	-30.67	-0.535292482	3.39

Table 4-1 Ground point removal calculations for the lower 22 LASER returns

4.3.4. HDL 32E UDP Packet Decoding

The transmission medium from the LiDAR sensor is a 100 MB/s ethernet connection, which sends UDP packets containing distance, azimuth rotational angle, vertical angle, LASER return intensity, and GPS Time Stamp.

Figure 4-8 illustrates the HDL 32E port 2368 UDP data packet. Every 2368 UDP Data packet consists of a 42-byte ethernet header. The two-byte LASER block ID on the HDL 32E always remains 0xEEFF, as the HDL 32E is equipped with only 32 LASER banks. The Velodyne HDL 64 has two 32 banks of LASER and the LASER block ID will alternate between 0xEEFF & 0xDDFF, depending on LASER bank in use at the time. The azimuth rotational angle is supplied from two-bytes in the form of integer with values between 0-35999 and must be divided by 100 to be converted into degrees. The next three bytes are repeated 32 times (once for every LASER onboard of the HDL 32E). This process is repeated 11 more times (12 in total) for every UDP

packet, completing 384 LASERs fires for every UDP data packet. At the end of every UDP packet, the status bytes are sent, containing the GPS time stamp Status Type and Status value. All data bytes of two bytes or more are sent using Little Endian Byte Order data structure (least significant byte first).

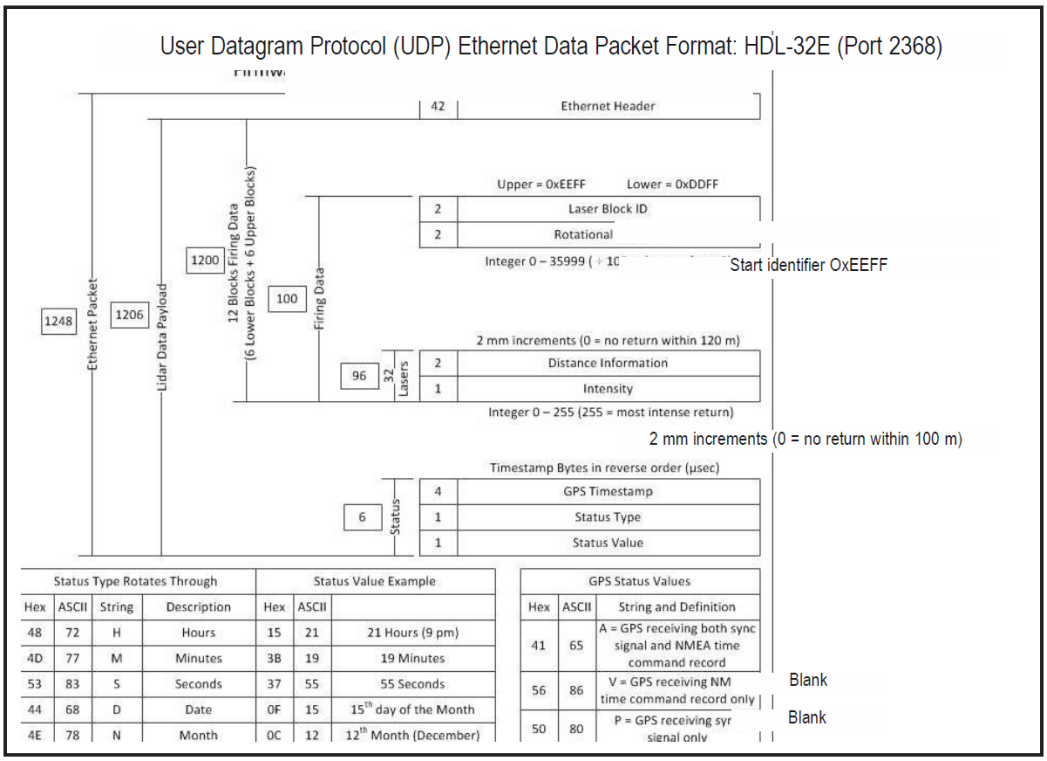

Figure 4-8 Velodyne HDL 32E UDP data packet, taken from [23]

4.3.5. LASER Firing Sequence

A LiDAR packet parsing algorithm was constructed in order to add order to the LASER returns from Velodyne HDL 32E. This step is initialized at the beginning in the first processing stage so that all subsequent processes can exploit the organized nature of the data. The organization of the data is an effort to make all further processing of algorithms more efficient and less costly of computational processing time. Processing of further steps can exploit the use of loops in smaller regions of the memory array as opposed to having to check all data locations repeated throughout the processing stages. Velodyne purposely designed the firing order of the HDL 32E LiDAR sensor to be interleaved to avoid potential ghosting caused by retro-reflectors. The LASER firing sequence of the HDL 32E is not ordered from ascending or descending vertical angle. Table 4-3 shows the LASER firing order from the HDL 32E user manual [23].

Firing order	DSR #	Vertical angle
1	0	-30.67
2	1	-9.33
3	2	-29.33
4	3	-8.00
5	4	-28.00
6	5	-6.66
7	6	-26.66
8	7	-5.33
9	8	-25.33
10	9	-4.00
11	10	-24.00
12	11	-2.67
13	12	-22.67
14	13	-1.33
15	14	-21.33
16	15	0.00
17	16	-20.00
18	17	1.33
19	18	-18.67
20	19	2.67
21	20	-17.33
22	21	4.00
23	22	-16.00
24	23	5.33
25	24	-14.67
26	25	6.67
27	26	-13.33
28	27	8.00
29	28	-12.00
30	29	9.33
31	30	-10.67
32	31	10.67

Table 4-2 HDL 32E LASER firing order

Table 4-3 in the column vertical angle shows the first LASER to fire is −30.67°; the second is -9.33°; the third is -29.33°, and so on. One can see that these LASER fires are not in vertical ascending or descending order. Algorithm 1 in this research provides the sequence needed to be able to order these points. Figure 4-9 illustrates the resulting memory array after the LASER return data is arranged and is packaged as human beings see the world in the memory storage array.

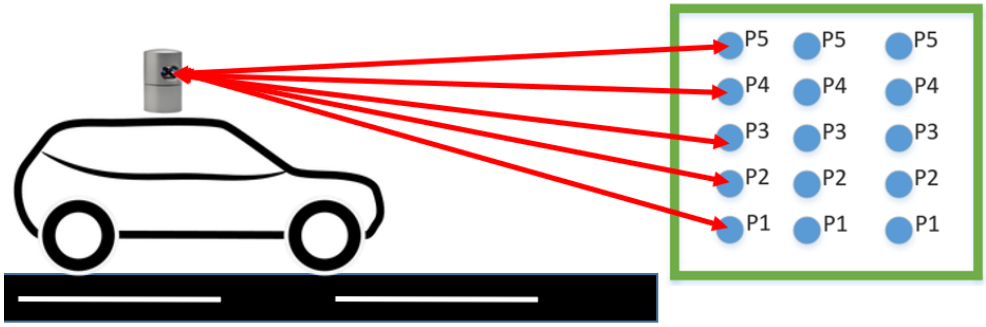

Figure 4-9 Visual representation of the vehicle storing LiDAR data arranged vertically, Images used from [26]

4.3.6. Ordering the Row Storage Location

Table 4-2 is a snippet from the chart developed to reorder the LASER return data sent from the HDL 32E. The pattern becomes revealed when the points are ordered from most negative to most positive angle. (The full chart can be found in Appendix A).

Access Pattern	Byte Order Access	Length	Firing 0rder	DSR #	Vertical Angle	Radian Angle
Laser Block ID	42	2 Bytes	***	***	***	***
Rotational	44	2 Bytes	***	***	***	***
Distance Information	46	2 Bytes	1	0	-30.67	-0.535292482
Intensity	48	1 Byte				
Distance Information	52	2 Bytes	3	2	-29.33	-0.51190507
Intensity	54	1 Byte				
Distance Information	58	2 Bytes	5	4	-28	-0.488692191
Intensity	60	1 Byte				
Distance Information	64	2 Bytes	7	6	-26.66	-0.465304779
Intensity	66	1 Byte				

Table 4-3 Organized LIDAR LASER point data by vertical Angle

4.3.7. Ordering the Column Storage Location

As discussed in Section 4.3.1, the memory array has 960 columns which can only hold 80 UDP data packets of LiDAR LASER return data. Equation 6 is derived to order the LASER points in the appropriate columns and never overrun the static memory array of 960 columns. This formula allows the overwrite of data every 80 UDP data packets.

$$Pkt\ Count = Packet\ Number\ to\ be\ parsed\ (0,1,2,\dots,N)$$

$$MaxPackets\ in\ memory = 80, as\ we\ store\ 80\ packets\ one\ complete\ rotation$$

$$Laser\ Packet = Laser\ Value\ (0,1,2,\dots,11)$$

$$Laser\ Scans\ in\ a\ Packet = 12\ (Every\ UDP\ packet\ has\ 12\ laser\ scans\ (HDL\ 32E\ ))$$

$$Column\# = resultant\ location\ to\ store\ value$$

$$Column\# = \left\{\left[Pkt\ Count - \left(MaxPackets\ in\ memory \cdot \left\lfloor \frac{Pkt\ Count}{MaxPackets\ in\ memory} \right\rfloor\right)\right] \cdot Laser\ Scans\ in\ a\ Packet\right\} + Laser\ Packet$$

Equation 6, Column number for memory allocation formula

4.3.8. Spherical to Cartesian Conversion

LiDAR data is retrieved from the sensor in spherical coordinates and must be converted to Cartesian in order to implement some of the algorithms, which will be covered in later sections. The formulas to convert to Cartesian coordinates are shown below as per manufacturer data sheet [23].

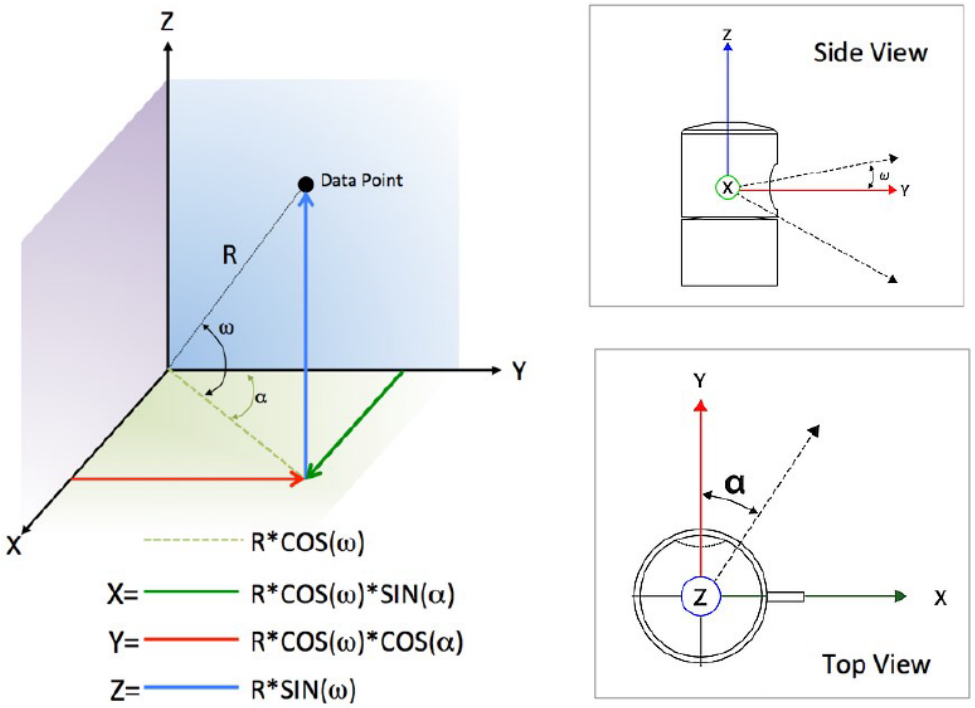

Figure 4-10 Spherical to Cartesian coordinates, taken from [23]

Equation 7, Spherical to Cartesian conversion

4.3.9. Algorithm 1: HDL 32E, LiDAR Packet Parser

Algorithm 1 converts the UDP LASER packet data from the Velodyne arrangement to order from the most negative angle LASER in array row location "0" incrementing up the array, populating all incremental LASER angles, finally reaching array row location "31" that houses the most positive LASER angle. The first for loop initiates at byte 42, where a little-endian swap is called and the "LASER Id" is organized as bytes 0x42 0x41. The "Rotation" adds 2 to the "pkt Byte" where the little-endian swap is once again called to organizing the rotation angle as bytes 0x44 0x43. Now that the LASER Id and rotation angle is implemented, 32 LASER returns of data must be organized in the memory array. LASER(i) is always initialized to 0 as for every UDP packet 12 columns of data will be populated in the array. The second for loop initializes at byte 46 in the LiDAR UDP data packet. This byte is little-endian swapped with the result of the distance from the first LASER return in the packet, becoming 0x46 0x45. The function call is made to Algorithm 2, which validates distance for maximum and minimum accepted LASER returns and converts data to Cartesian. When returning from the function call, the LASER(i) is incremented, which controls the row in the memory array for storage. The third for loop is initialized at byte 49 and increments

by 6. The process is repeated 16 times before exiting the third for loop. Once this third for loop has completed, all the LASER data of 32 returns is organized in a memory array. The GPS and status bytes are then extracted, utilizing similar methods, with the exception that the GPS value is 4 bytes in length. This process is completed 12 times in total for every UDP LASER packet, in order to extract all LASER returns.

Algorithm 1: HDL 32E, LiDAR Packet Parser

Input: LiDAR UDP packet (pkt)
Output: Organized LiDAR Points by Vertical Angle in Memory
Data: Testing set Port 2368 Packets

```
1  LASER(i) ← i = 0, 1, ..., 31                          // 32 LASERS per set
2  LsrCnt(j) ← j = 0, 1, ..., 11                         // 12 sets of LASERS per packet
   /* Beginning of the loop                                                          */
3  for (pktByte = 42; pktByte < 1143; pktByte += 100) do
4      Call Function (Little endian byte swap cacatenate)          // Laser Id
5      Rotation = pktByte + 2
6      Call Function (Little endian byte swap cacatenate)          // Rotation
7      Rotation = Rotation/100                  // Rotation value to degrees
8      for (pktByte = 46 +(100 * LsrCnt(j)); pktByte < 142 +(100 *
        LsrCnt(j)); pktByte += 6 ) do
9          Call Function (Little endian byte swap cacatenate)      // Distance
10         Call Function (Convert Data to Cartesian (Algorithm 2))
11         LASER(i) + +                                          // Increment LASER
12     for (pktByte = 49 +(100 * LsrCnt(j)); pktByte < 142 +(100 *
        LsrCnt(j)); pktByte += 6 ) do
13         Call Function (Little endian byte swap cacatenate)      // Distance
14         Call Function (Convert Data to Cartesian (Algorithm 2))
15         LASER(i) + +                                          // Increment LASER
16     LsrCnt(j) + +                                       // Increment LASER Set
17 GPS LowByte = 1242
18 Call Function (Little endian byte swap cacatenate)              // Rotation
19 GPS HighByte = 1244
20 Call Function (Little endian byte swap cacatenate)              // Rotation
21 GPS TimeStamp = GPS high and Low byte cacatenate
22 StatusTypeByte = pktByte1247                          // One Byte in Length
23 StatusValue = pktByte1246                             // One Byte in Length
```

4.3.10. Algorithm 2: Validation, Cartesian, and Storage of Data

Algorithm 2 works hand in hand with Algorithm 1 as a function call. To reduce repetitive code in Algorithm 1, Algorithm 2 was developed. The LASER row location is passed to Algorithm 2 from Algorithm 1 as LASER(i). The Max distance variable is always set to the maximum allowable distance of the HDL 32E LiDAR sensor (70m). If the LASER to be examined is a negative angle value, such as the lower 23 values of the 32 LASERS, the MaxDistance variable must be changed for ground point removal using Equation 5 to calculate the MaxDistance for the downward facing angle at test. Once the MaxDistance is computed for negative angle firing LASERS or left at 70m for positive firing LASERS, the data is validated for an acceptable value. If the LASER distance return value does not fit in between minimum and maximum values of acceptance, the row and column location of the memory's zero flag is set to 1. A zero-flag set to one allows all other subsequent processing steps to avoid further processing at this location. If the data is acceptable the Cartesian coordinates are calculated and stored at the memory location.

Algorithm 2: Validates data based on distance,converts to Cartesian, and stores data

Function: Convert Data to Cartesian
Input: LiDAR UDP Packets (pkt)
Output: Organized LiDAR Points by Vertical Angle in Memory

```
1  LASER(i) ← i = 0, 1, ..., 31
2  pkt(i) for i = 1 to N                          // Parse Packets 1 to N
   /* Ground Point Removal Step (GPR)                                    */
3  if LASER is facing Downward then
4      CalculateMax(θ)Distance
5      MaxDistance = Max(θ)Distance               // Need to calculate the Max
         distance if facing the ground
   /* if valid data is present at this location store values             */
6  if MinDistance < Distance < MaxDistance then
7      Calculate X, Y, Z values
8      Intensity = pktByte +2                     // LASERS return intensity
9      Store Distance, Rotation(theta), Column Value, X, Y ,Z, Intensity
   /* Set Zero Flag 1 = no data, 0 = data                                */
10 else
11     Zero Flag = 1
```

4.3.11. Result of Algorithms 1 and 2 Processing

After Algorithms 1 and 2 have been executed, a single UDP LiDAR data packet is stored in the static memory array, where twelve sets of thirty-two LiDAR LASER returns are stored. Each of the twelve sets of thirty-two LASER returns is organized from most negative to positive vertical angle. Every LASER set (1-12 per UDP packet) is a new column in the memory array which increments through the static memory array using Equation 6. The storage of the LiDAR return data is best described as a human being would see the environment (Figure 4-10).

4.4. Stage Two – Data Clustering

Clustering of the LASER data is imperative to detect objects from the point cloud. Common clustering methods found in LIDAR are Density-Based Spatial Clustering of Applications with Noise (DBSCAN) method proposed by [27] and ABD originally proposed by [7], which has been gaining popularity. [8] had shown great potential of eliminating ghost while using the ABD clustering method. In this section, the clustering of the data will utilize the methods adopted by [8], utilizing a candidate set to seek cluster matches. However, modification of the clustering algorithm proposed in [8] exploits the organized data from stage one and employs the use of "FOR" loops. The ABD, a two-dimensional detection scheme, is also adopted to three dimensions. A cluster array is established for keeping data about the cluster, utilizing a struct. The struct is accessed by the cluster number and contains information of number of points in a cluster, cluster shape type, left most column, right most column, lowest row value, highest row value, lowest most vertex, highest most vertex, left most vertex and right most vertex. In addition to the general information about the cluster, the outermost rows and columns are utilized for retrieving the data for shape extraction in Stage Three. The vertex locations are used for when an L-shape needs to be fitted.

4.4.1. Clustering of Objects

The objective of clustering objects is to be able to identify individual objects in the point cloud. The shape of an object can tell the autonomous vehicle system a great deal. The ABD method is employed to cluster points by building a threshold that increases or decreases in size, depending on the distance of the LASER return. If the distance between two points is beyond the threshold, a break point is detected, and a new cluster begins.

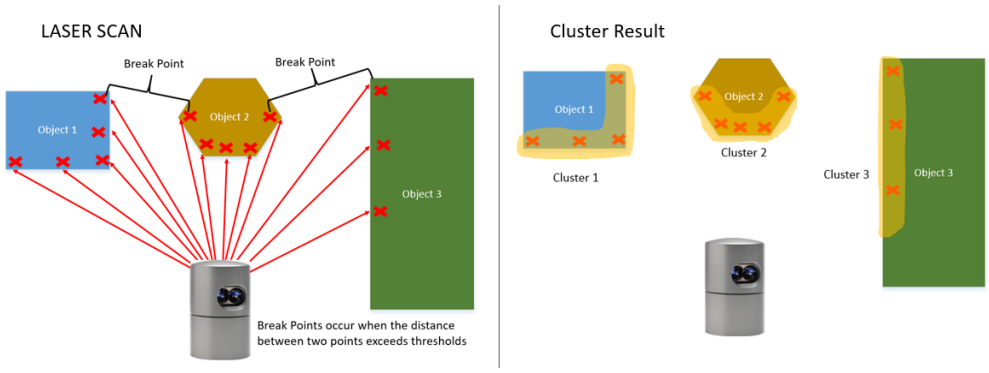

Figure 4-11 LASER scan with clustering of points

A dynamic threshold depending on distance is imperative in the world of LiDAR, as LASERS are fired at angles, where the greater the distance the further separated the beams become. Figure 4-12 illustrates how a distance threshold must be dynamic at varying LASER returns distances. The blue circle is an appropriate threshold distance at the reduced distance LASER returns. However, this threshold would not function for LASER returns of a greater distance. Nor would the greater distance threshold be appropriate for the reduces LASER returns.

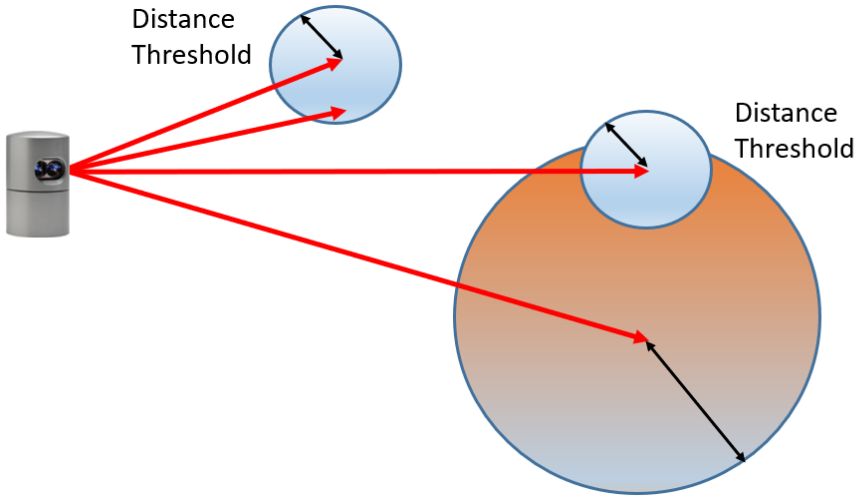

Figure 4-12 Distance thresholds must adapt based on distance from sensor

4.4.2. Adaptive Breakpoint Detector (ABD)

The following adaptive breakpoint detector proposed by [25] varies the threshold circle depending on the distance of the previous point.

Where,

$$D_{max} = r_{i-1}^{t} \cdot \frac{\sin(\Delta\theta)}{\sin(\lambda - \Delta\theta)} - 3\sigma_r$$

$$D_{max} = Threshold\ circle\ radius$$

$$r_{i-1}^{t} = radius\ of\ previous\ point$$

$$\Delta\theta = Angle\ between\ points$$

$$\sigma_r = noise\ from\ sensor(HDL\ 32E\ \sigma_r = 2cm)$$

$$\lambda = maximum\ incidence\ angle$$

Equation 8, ABD max distance calculation

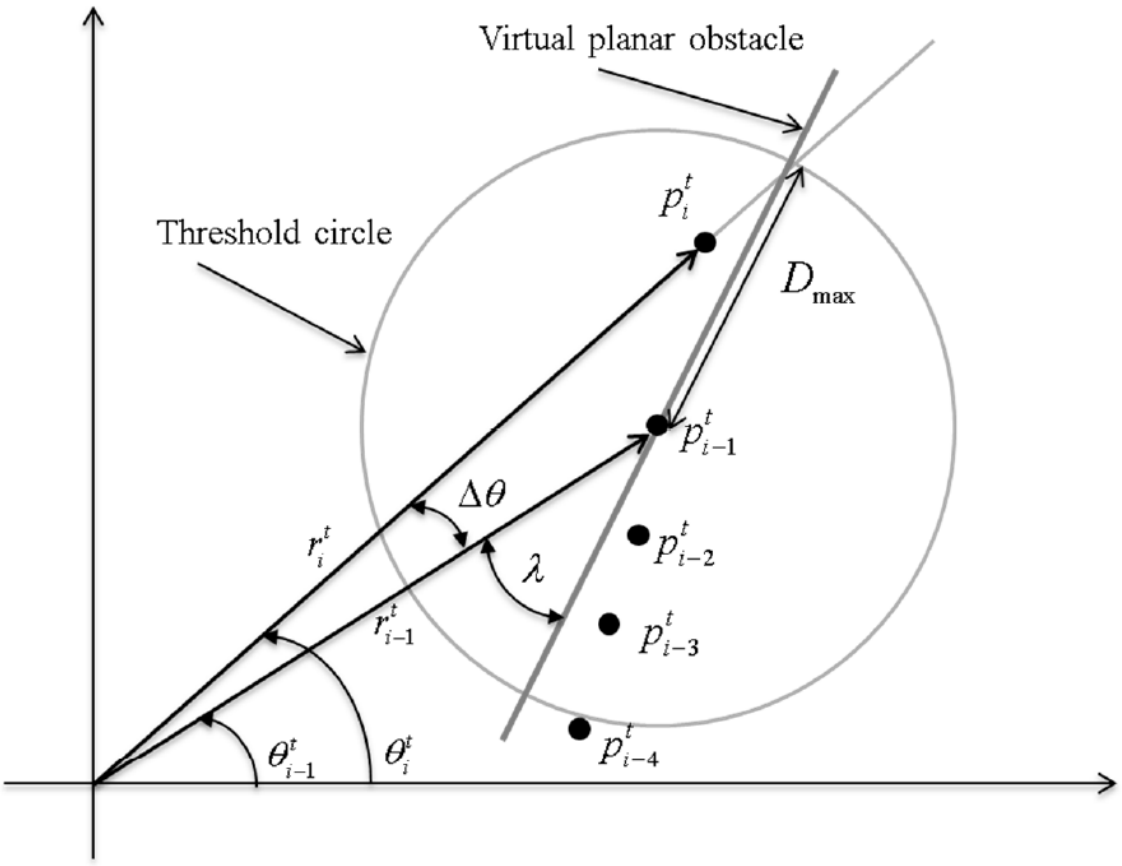

Figure 4-13 ABD threshold diagram, taken from [7]

λ must always be greater than the largest angle between the two points ($\Delta\theta$) under test to keep the ratios positive. Being that the user can program how many LASER returns to look above, below, and left of the point under test, the value of λ must be set appropriately.

4.4.3. Dealing with Occlusions

Occlusions occur when a smaller object resides in front of a larger object. The image below illustrates an example of a bird in front of the vehicle. In the case of the vertical scan, the most upper and lower points are part of the car and therefore should be a part of the same cluster, whereas the two points in the center are the bird and should form a new cluster. In order to do this, one employs the ability to look above and below multiple points to see if the point in question is in fact part of another cluster. The horizontal scan image shows a similar case but with respect to the azimuth or horizontal scan where the leftmost LASER should be in the same cluster as the rightmost LASER.

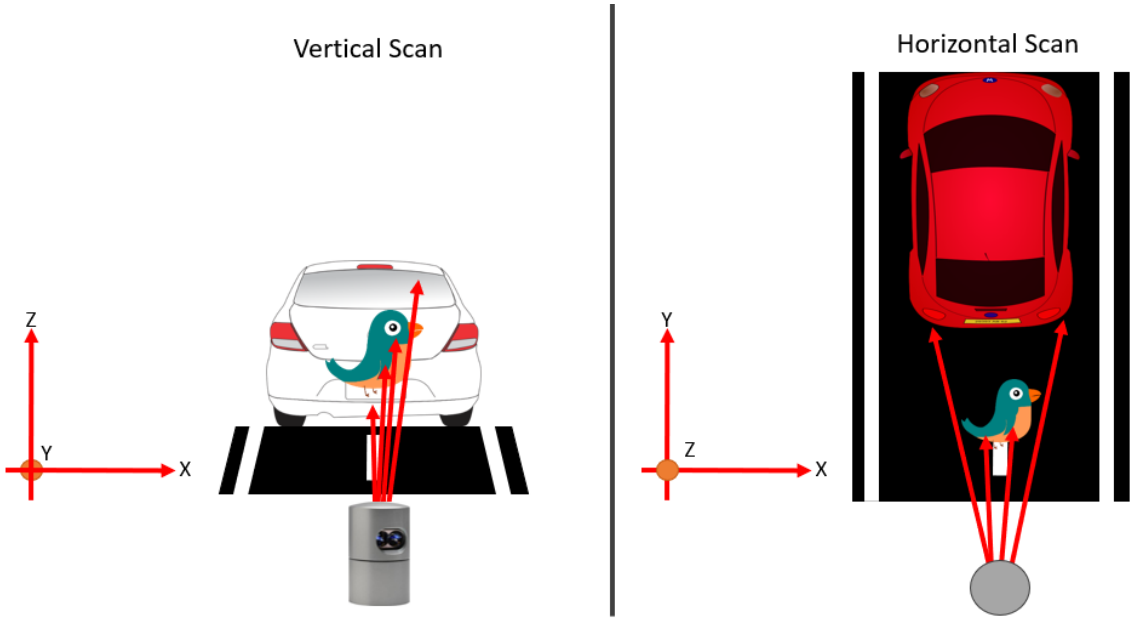

Figure 4-14 Horizontal and vertical example views of occlusions

4.4.4. Clustering of Data

[6] proposes a layer wise segmentation process which shape extracts each layer, merging multi-layer clusters to a single cluster later in the process. [8] argues that layer wise segmentation does not work well and requires multi-layer segmentation. [8] proposes a multi-layer segmentation algorithm utilizing a candidate set for test. [8] cannot utilize nested for loops in clustering as the data provided by the LiDAR sensor is unorganized and sometimes is missing data due to an out of range LASER or an invalid result. This research modifies the clustering algorithm of [8] and looks

to exploit the fact that in Algorithm 1 of this research, the data points were organized by layer and rotation in static memory.

The candidate test system is implemented in this research in an array of 32 rows by 1 column.

Rules for clustering:

- If data is found in a row, the data must be moved to the corresponding row of the candidate set
- The candidate set will be searched below and above the data row value set by the user to search for a cluster match (i.e. if row one is looking for a cluster match and does not find a match by row five… it is most likely a waste of process to search all 32 rows)
- Every data point will be assigned a cluster number
- All candidate set values must be assigned a cluster number
- All values moved to the candidate set will update the cluster number in the memory location also
- Points are tested sequentially from the bottom to maximum row in the image below
- Once a column is tested the next column is tested, points are tested sequentially from the lowest to the maximum row
- The distance between the points under test and candidate set must be less than the distance threshold to be considered the same cluster

Figure 4-15 shows a hypothetical data set under test. LASER returns in the memory array are tested sequentially from row one to Row 6. The leftmost image illustrates how the latest inspected point is moved into the candidate set row. The center image illustrates the point under test and the three options that would be tested if the user chose to select to test +/-1 row. The third attempt shows the match signaled by a green check mark on Row 3. The rightmost image shows how the previous point under test was updated to the cluster and the candidate set was updated. Now the row has been incremented to the next point under test.

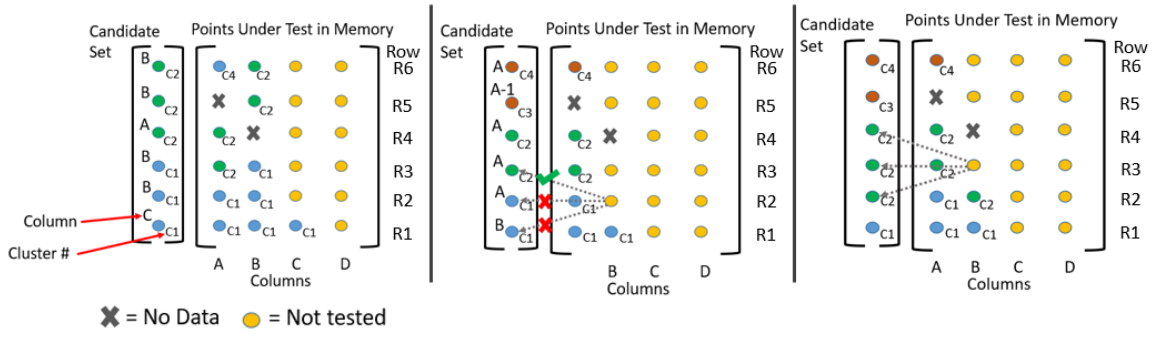

Figure 4-15 Clustering algorithm visualization with candidate set

4.4.5. Transitioning Between Data Packets

Under ideal circumstances, all clusters/objects would remain in one UDP packet. However, in the real world, an object or cluster could span across many UDP data packets sent from the LiDAR sensor. The candidate set allows the new data packet to look at the previous clusters for potential matches along the seams of adjoining packets. Every UDP packet that becomes available is parsed, stored in memory, clustered/segmented, then shape extracted. This creates a dynamic shape updating system where one data packet could first conclude a specific cluster number to be one of the four shape types proposed in this research, then, when this cluster overlaps to another packet and more information is provided (i.e., more LASER returns), the shape could change as more data about the object becomes available.

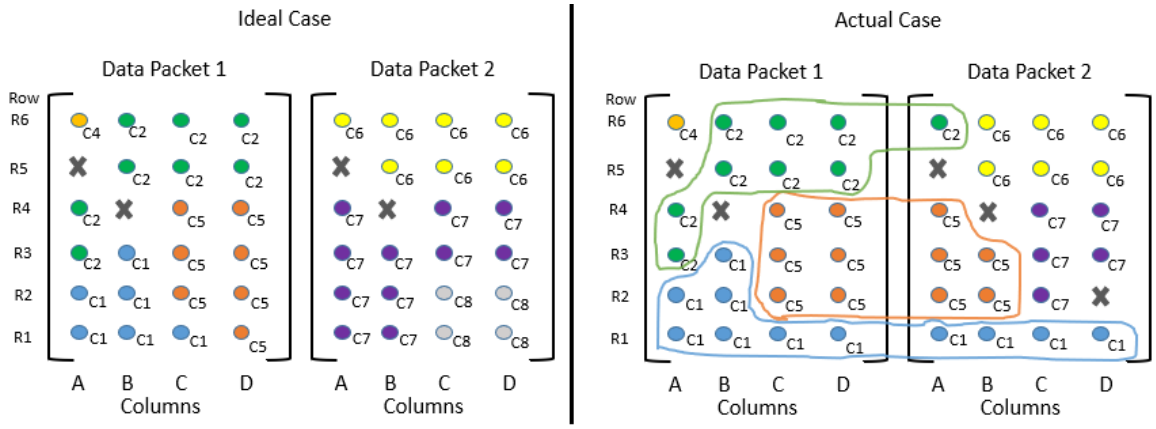

Figure 4-16 Clusters that overlap UDP data packets, Left image illustrates the ideal case where objects/clusters are contained to each data packet & the right image shows the actual case where objects/clusters overlap between data packets

4.4.6. Adopting the Adaptive Breakpoint Detector to Three Dimensions

In Section 4.4.2, the ABD was used to determine the threshold between two LASER points on a two-dimensional plane. Figure 4-17 illustrates a hypothetical diagram, extracting the ABD threshold in a single column. The angle ($\Delta\theta$) between the two points can be calculated by the vertical angle (θ), where, in this example, the difference between each row is 5°. If the case was different and the ABD threshold was to be solved between rows 1 and 3, the angle ($\Delta\theta$) would be the difference of 10°. This is still a case where the ABD detector is operating in two dimensions, as the columns have not been considered yet.

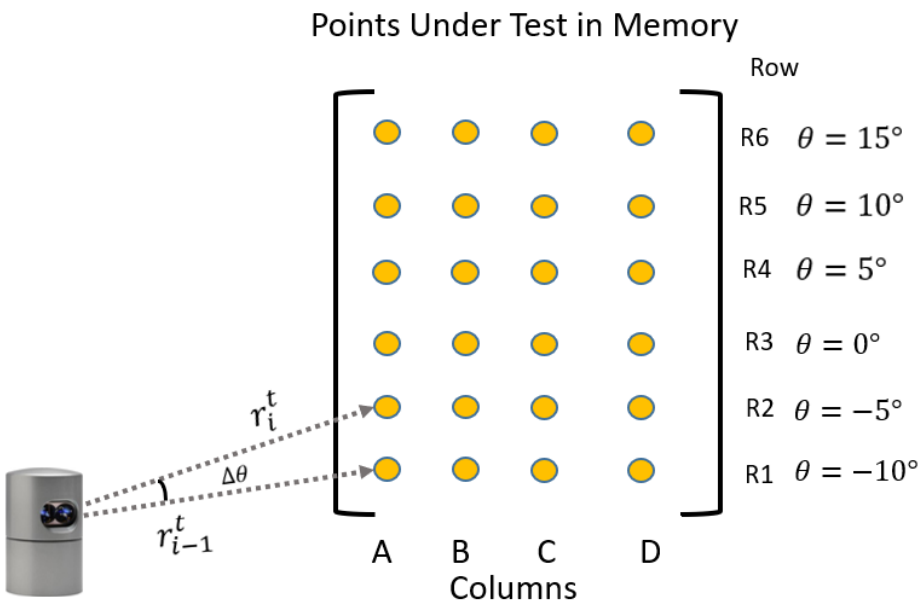

Figure 4-17 Applying ABD in two dimensions

However, when dealing with LASER points in three dimensions, the same ABD can be applied to any two points if you consider a moveable plane between any two points. The movement of this plane enables the calculation of the distance threshold between any two points, no matter the row and column combination. The row angle (θ) is provided by the LASER firing angle of the Velodyne HDL 32E, which is 1.33° between every LASER row. The column angle is determined by the "rotational angle" provided when the packet is parsed from bytes 45 and 46 respectively, which is incremented by a 100 (145 &146, 245 & 246, …) for eleven additional times for every UDP data packet (12 in total). Figure 4-18 illustrates a hypothetical image of LIDAR data where the vertical LASER increments upward by 5° (θ) and increases the azimuth angle by 2° ($\emptyset$). This method is shown as a static environment, as slight deviations to the LASER returns are not considered. This method is still applicable to the post-processing of a SLAM algorithm if the information of angular difference between the two points under test is known.

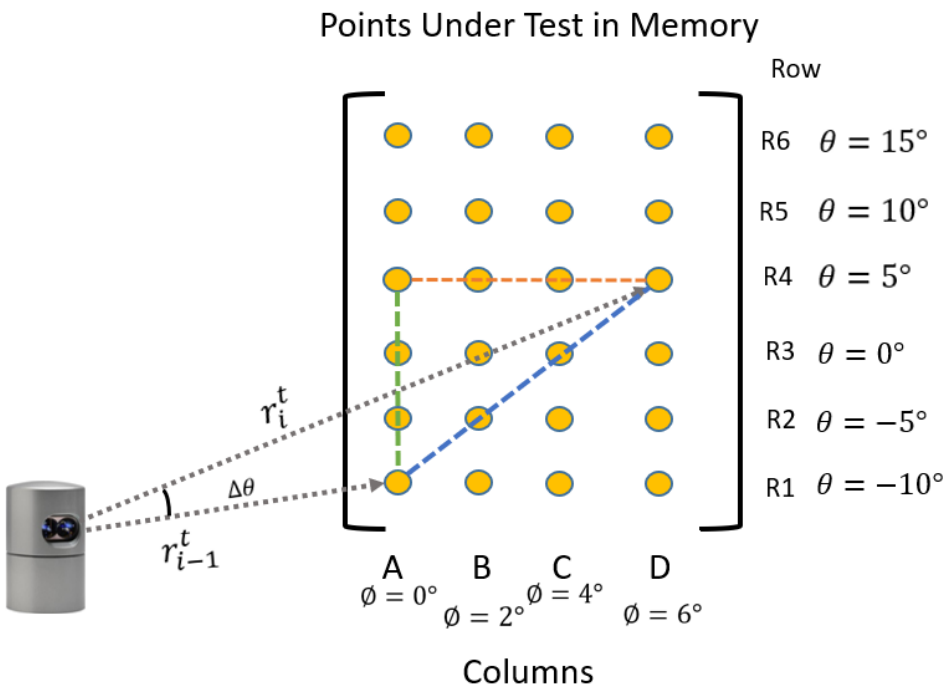

Figure 4-18 ABD adapted to 3-dimensional coordinates

The equation below shows how the angle between any two points is calculated.

$$\Delta\theta = \sqrt{\left(\theta_{r_i^t} - \theta_{r_{i-1}^t}\right)^2 + \left(\emptyset_{r_i^t} - \emptyset_{r_{i-1}^t}\right)^2}$$

Equation 9, Angle between any two points formula

Substituting Equation 9 into Equation 10, the maximum distance that will be accepted to a cluster can be solved by multiplying the previous points radius(r_{i-1}^t) or distance between the LiDAR sensor and the previous points.

$$D_{max} = r_{i-1}^t \cdot \frac{\sin(\Delta\theta)}{\sin(\lambda - \Delta\theta)} - 3\sigma_r$$

Equation 10, Maximum distance for ABD

To calculate the Euclidean distance between two points on a polar plane, the following equation is utilized.

$$\|Distance\ (p_A, p_B)\| = \sqrt{(r_{i-1}^t)^2 + (r_i^t)^2 - 2(r_{i-1}^t)(r_i^t) \cdot \cos(\Delta\theta)}$$

Equation 11, Formula for calculating the Euclidean distance between two points in a polar plane

Testing the points to see if the value falls below the threshold is resolved if Equation 12 is true.

$$\|Distance\ (p_A, p_B)\| < D_{max}$$

Equation 12, Equation for testing if the distance between two points is less than Dmax

4.4.7. Algorithm 3: Clustering and Segmenting

Algorithm 3 performs Stage 2 of the proposed algorithm in this research the clustering stage. This algorithm initializes the count at 0 (variable Cnt) and increments to 11 for the 12 columns contained in an UDP data packet. A nested for loop of 32 rows to increment through every LASER return from the HDL 32E. These FOR loops will increment through a total of 384 LASER returns per UDP data packet. Every LASER return goes through the same test, and initially the user determines how many rows to search above and below the row at test. The user provides a numeric value for the variable "Maxdistance" initialized in the tuning parameters. The "Maxdistance" variable is added and subtracted to the row under test to produce "RowMax" and "RowMin" Values which are contained between 0-31. The "RowMax" and "RowMin" values are used in the while loop to navigate search locations in the candidate set. Candidate set locations are first tested if data exist in the location by use of the zero flag. If no data exist, the "RowMin" value increments up toward "RowMax". If data exist in the candidate set, the distance between the column in the candidate set is tested with the memory array. If the distance calculated is greater than the "Maxdistance" set by the user, the test on this point is skipped and the value increments up toward "RowMax". If the Column is less than the "Maxdistance" variable the ABD threshold is computed along with the Euclidean distance between the two points under test. If the distance between the two points is greater than the threshold, the point in the candidate set is skipped and the value increments up toward "RowMax" to test another point. If the Euclidean distance between the two points is less than the ABD threshold, the cluster number is assigned to the location in the memory array. The leftmost and rightmost column, and the lowest and highest rows are updated if the location exceeds previous entries in the cluster array. Lastly the memory array LASER return data is copied to the candidate set row of the row location undertest of the memory array. The While loop is then broken and the next LASER return value in the UDP data packet which is stored in the memory array, is tested. If no matches were found in the candidate set, a new cluster is assigned where the left and right row, low and high column are assigned the row and column location of the first point. The memory array LASER return data is copied to the candidate set of the same row of the row location undertest in the memory array.

Algorithm 3: Modified Clustering Algorithm for Organized LiDAR Data

Input: Pkt Number, Maxdistance (Distance to search above, below, and left of the Point under test, ClusterCount)

/* Data is contained in the memory array and must be accessed */

Output: Points from the UDP packet under test are assigned a cluster number

```
1  Cnt = 0,                                                  // Set counter to 0
2  for (Cnt; Cnt < 12;Cnt += 1) do
3      Column = "Equation 6"(Column Number for Memory Allocation formula)
4      Row = 0;
5      for (Row; Row < 32; Row += 1 ) do
           /* Test for Lowest row to search (Reduce Occlusions)                */
6          if (Row - Maxdistance < 0) then
7              RowMin = 0;
8          else
9              RowMin = Row - Maxdistance;
           /* Test for Highest row to search (Reduce Occlusions)               */
10         if (Row + Maxdistance > 31) then
11             RowMax = 31;
12         else
13             RowMax = Row + Maxdistance;
14         while RowMin <= RowMax do
               /* 0 = data, 1 = no data (zero flag                             */
15             if (CandidateSet[Rowmin].ZeroFlag = 0 and (CandidateSet -
                MemoryArray Column < Maxdistance)) then
                   /* Only test if data exist and column is in range           */
16                 Compute Distance threshold (ABD)
17                 Compute Distance between points
18                 if (Distance between points < Distance threshold) then
19                     Memory Array = Candidate set Cluster number
20                     Update Left and Right Column Value
21                     Update High and Low Row Value
22                     Copy Memory array Point info to the matching Row
                        of the CandidateSet
23                     Break;
                       /* Break, Leave once a match is found                   */
               /* If no data is found or at first run to populate the
                  candidate set                                                */
24             if (RowMax == RowMin) then
25                 ClusterCount += 1; /* New cluster                           */
26                 Memory Array [Row][Column] Cluster Number =
                    ClusterCount
27                 Update Left and Right Column Value
28                 Update High and Low Row Value
29                 Copy Memory array Point info to the matching Row of
                    the CandidateSet
```

4.4.8. Result of Algorithm 3 Processing

After Algorithm 3 has been applied to the stored data from Algorithms 1 and 2. Every LASER return is assigned a cluster value. A cluster array is populated which contains data about the cluster such as cluster number, left and right most column, lowest and highest row. This locational data will be used later for shape extraction stages.

4.5. Stage Three - Shape Extraction

The shape extraction step increments through a series of tests to determine the shape type to be assigned to the cluster. Four different shapes are classified: a point, line, L-Shape, and a convex hull (polygon). The Cartesian z component of the cluster is ignored, and the shape extraction is executed on a two-dimensional plane. Every shape, except for clusters with only one LASER return, must be computed with principal component analysis (PCA) to determine if the shape is a point with multiple LASERs in the cluster, line, L-Shape, or a convex hull. Shapes are returned as a numeric value contained in the struct under shape identification number. The LASER return data for all shapes is retained in the memory array in order to keep the memory static. Clusters are marked by the lowest row, highest row, left most column, and right most column. When the cluster needs to be accessed for shape extraction, the data is written to the shape evaluation array.

Shape ID Number	Shape
1	Point
2	Line
3	L-Shape
4	Polygon

Table 4-4 Shape type in relation to assigned number

4.5.1. Shape Extraction from the Memory Array

The third stage in this algorithm is to perform shape extraction on every cluster produced by the data set. The cluster array is reviewed for all new clusters to be assessed or updated clusters that need to be reassessed. An example of an updated cluster would be a previously found cluster in a previous UDP data packet that extended into the latest UDP packet. The cluster array provides the information needed to retrieve the data for shape extraction, including left and right most column as well as lowest and highest row. A nested for loop is performed which increments through the data in the memory array, which moves any value found to be a part of the cluster value to the shape extraction array. Not all elements in the search area set by the highest and lowest row as well as the leftmost and rightmost column necessarily belong to the cluster; however, this process allows for static memory. Figure 4-19 illustrates how the shape extraction stage sources the cluster data from the memory array. The roadmap set forth by the row and column markers. Notice cluster 2 is to be retrieved but data from other clusters and invalid data could still exist in this area.

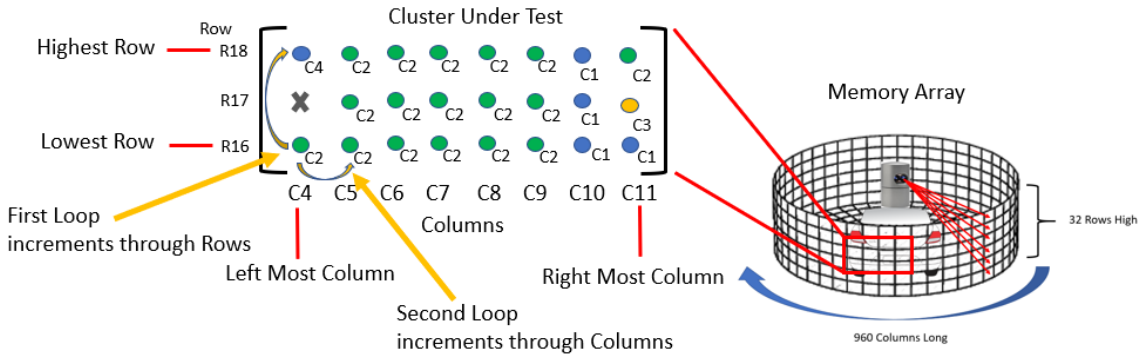

Figure 4-19 Illustration of a cluster extracted from the memory array

4.5.2. Decision Tree

The flowchart below illustrates the process of decision making for shape extraction.

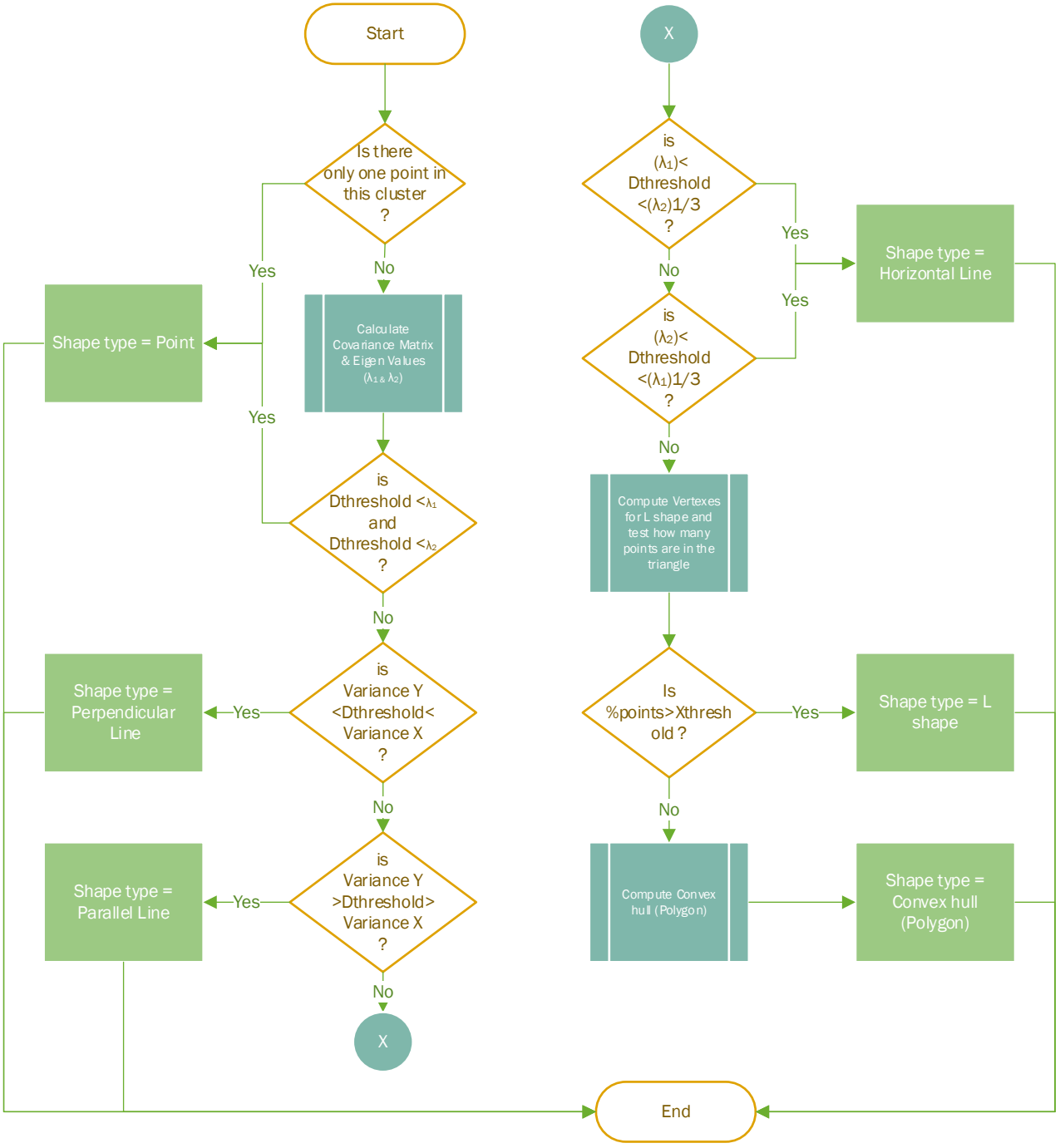

Figure 4-20 Decision tree for shape extraction

4.5.3. Principal Component Analysis (PCA)

PCA is a statistical process used for converting a set of data points into a linear transformation of two orthogonal magnitudes and vectors (eigenvalues and eigenvectors). The magnitude of the two produced eigenvalues can offer great insight into determining if LiDAR data correlates to a certain shape, whereas the vector can lead us to the direction of the shape. PCA is particularly useful when the cluster of LiDAR data does not run parallel with either the x or y axis. In order to solve the orthogonal components, a system of linear equations must be solved. Below the mathematical procedures are shown.

Consider data set A is under test

$$Data_{set} = A = \begin{bmatrix} ax_{11} & ay_{12} & az_{13} \\ ax_{21} & ay_{22} & az_{23} \\ \vdots & \vdots & \vdots \\ ax_{n1} & ay_{n1} & az_{n1} \end{bmatrix}$$

Ignoring the z component and setting all z components equal to zero the data set becomes,

$$Data_{set} = A = \begin{bmatrix} ax_{11} & ay_{12} \\ ax_{21} & ay_{22} \\ \vdots & \vdots \\ ax_{n1} & ay_{n1} \end{bmatrix}$$

Traditionally, calculating the deviation matrix "a" to normalize the data is performed with Equation 13,

$$a = A - [1] \cdot A \cdot \left(\frac{1}{n}\right)$$

Equation 13, Deviation matrix

To become,

$$Data_{set} = a = \begin{bmatrix} ax_{11} & ay_{12} \\ ax_{21} & ay_{22} \\ \vdots & \vdots \\ ax_{n1} & ay_{n1} \end{bmatrix} - \begin{bmatrix} 1_{11} & 1_{12} \\ 1_{21} & 1_{22} \\ \vdots & \vdots \\ 1_{n1} & 1_{n1} \end{bmatrix} \cdot \begin{bmatrix} ax_{11} & ay_{12} \\ ax_{21} & ay_{22} \\ \vdots & \vdots \\ ax_{n1} & ay_{n1} \end{bmatrix} \cdot \left(\frac{1}{n}\right)$$

Where,

$$[1] = Unity\ Matrix$$

$$n = the\ number\ of\ row\ samples$$

However, in practical application to code efficiently the following steps are implemented.

First, the mean of the data for both the X and Y values must be calculated.

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

Equation 14, Mean formula

The mean is then subtracted from the data set to center the data about the zero-crossing axis (normalize). One could conclude that both methods yield the same results, except that one is easier to program.

$$Data_{set} = a = \begin{bmatrix} ax_{11} - \bar{X} & ay_{12} - \bar{Y} \\ ax_{21} - \bar{X} & ay_{22} - \bar{Y} \\ \vdots & \vdots \\ ax_{n1} - \bar{X} & ay_{n1} - \bar{Y} \end{bmatrix}$$

Equation 15, Means values are subtracted from the appropriate columns

Figure 4-17 illustrates an example procedure of removing the z component and normalizing the data to zero with a polygon shape.

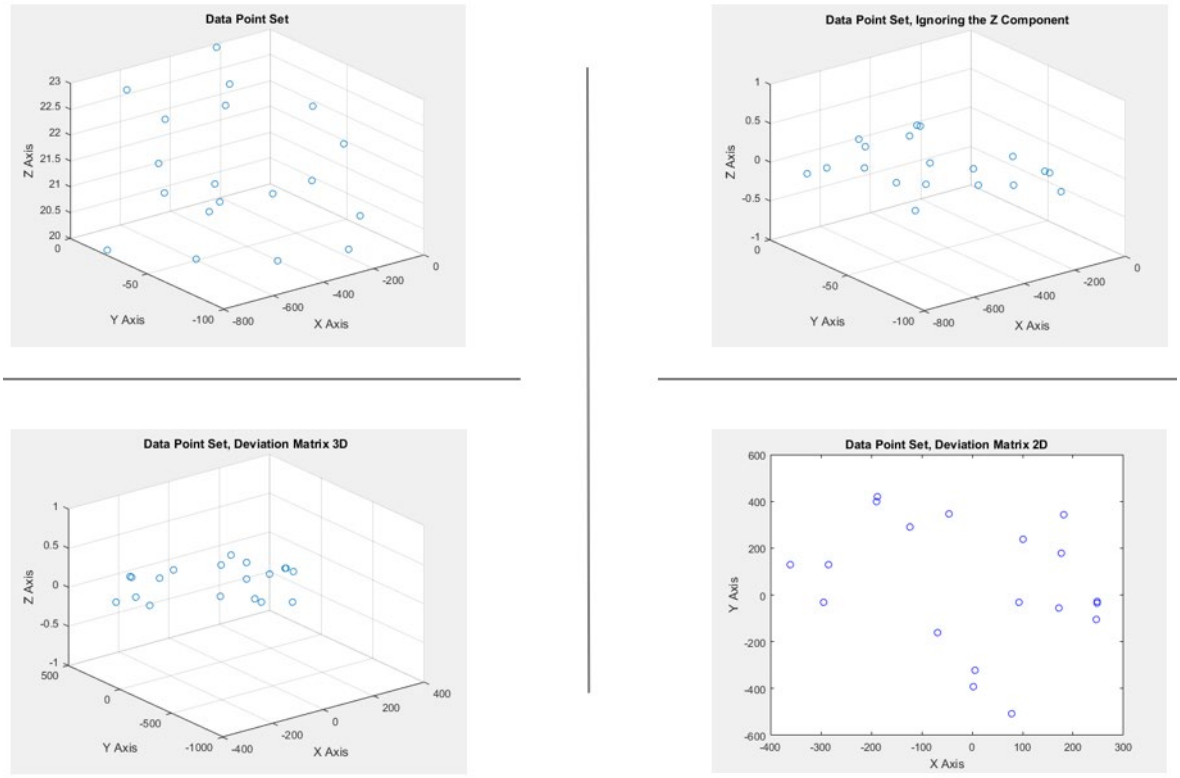

Figure 4-21 Visualization of data with removed z component and normalized data

The Covariance matrix is than calculated by multiplying matrix "a" with the transposed version of itself "a^T",

$$covarianceMatrix = a \cdot a^T$$

Equation 16, Covariance matrix

The covariance matrix equals the deviation matrix multiplied by the transposed deviation matrix.

$$covarianceMatrix = \begin{bmatrix} cov(x,x) & cov(x,y) \\ cov(y,x) & cov(y,y) \end{bmatrix}$$

Equation 17, 2 X 2 covariance matrix

The covariance matrix location (1,1) provides the variance in the x direction ($cov(x,x)$) while location (2,2) provides variance in the y direction ($cov(y,y)$).

Solving for eigenvalue and eigenvectors

$$A\bar{v} = \lambda\bar{v}$$

Equation 18, Eigenvector and Eigenvalue formula

Solving the identity matrix multiplied by lambda,

$$\lambda \mathrm{I} = \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}$$

Equation 19, Eigenvalue multiplied by the identity matrix

Where,

$$\mathrm{A} - \lambda \mathrm{I} = \begin{bmatrix} cov(x,x) - \lambda & cov(x,y) \\ cov(y,x) & cov(y,y) - \lambda \end{bmatrix}$$

Equation 20, Covariance matrix subtracted the λ identity matrix

In order to solve for the eigenvalues and eigenvectors, the Jacobi cyclic method was employed [33]. The Jacobi method is a common algorithm for solving the eigenvalue problem and is accessible in almost any numerical methods book. An advantage of the Jacobi cyclic method over the classic Jacobi is that each element is rotated only once (called a sweep) and the convergence is generally quadratic, and ordering can be implemented row wise. The Jacobi cyclic method was preferred as both eigenvalues are revealed, whereas methods such as Rayleigh and the power method converge to only the dominant eigenvalue. Both eigenvalues are required in this research for the principal component analysis stage for shape extraction.

4.5.4. Point

Any cluster that contains only one LASER return is automatically classified as a point. There could be a group of points that could still be classified as a point if the variance of the points is less than the determined threshold, or λ_1 and $\lambda_2$are less than a threshold.

Points return the x mean and y mean of the cluster that was tested

$$Point = P_{mean} = \begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix}$$

Equation 21, Point formula that return x and y mean

Example of shapes considered to be points are shown below,

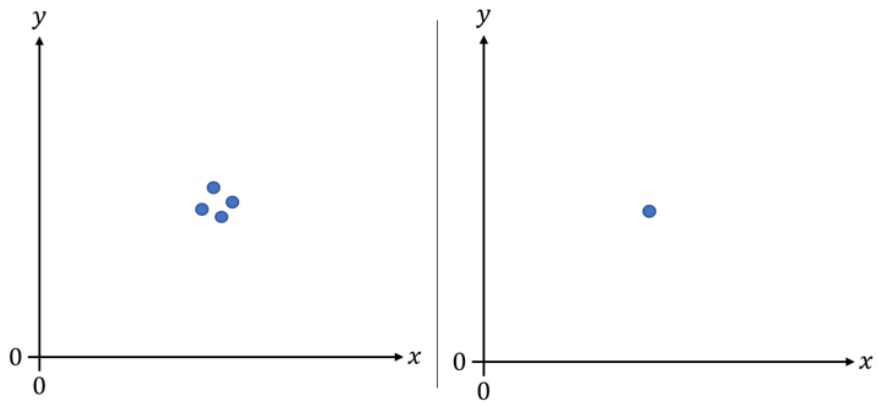

Figure 4-22 Illustrates how a point shape can be considered a densely packed group of LASER points or a single LASER point

4.5.5. Line

When the decision has been made that the cluster set at test is not a point shape, the algorithm then tests for a line. Figure 4-23 illustrates how the variance ($\sigma_x^2 \; and \; \sigma_y^2$) can be used to determine a line when the data resides parallel with an axis. However, when the line is diagonal, the variances ($\sigma_x^2 \; and \; \sigma_y^2$) are not a good determinate of the shape. Eigenvalues are then used on the shape to determine if the data set under test can be considered a line. The eigenvalues help determine if a diagonal line can be considered from the data set, if one eigenvector is above the threshold and the other eigenvalue is below one third, the threshold a line shape is considered. Eigenvectors operate like a weather vane, where the eigenvector assigned to the largest eigenvalue will point in the main direction of the data and the smaller eigenvalue associated to the other eigenvector shows the orthogonal magnitude [28].

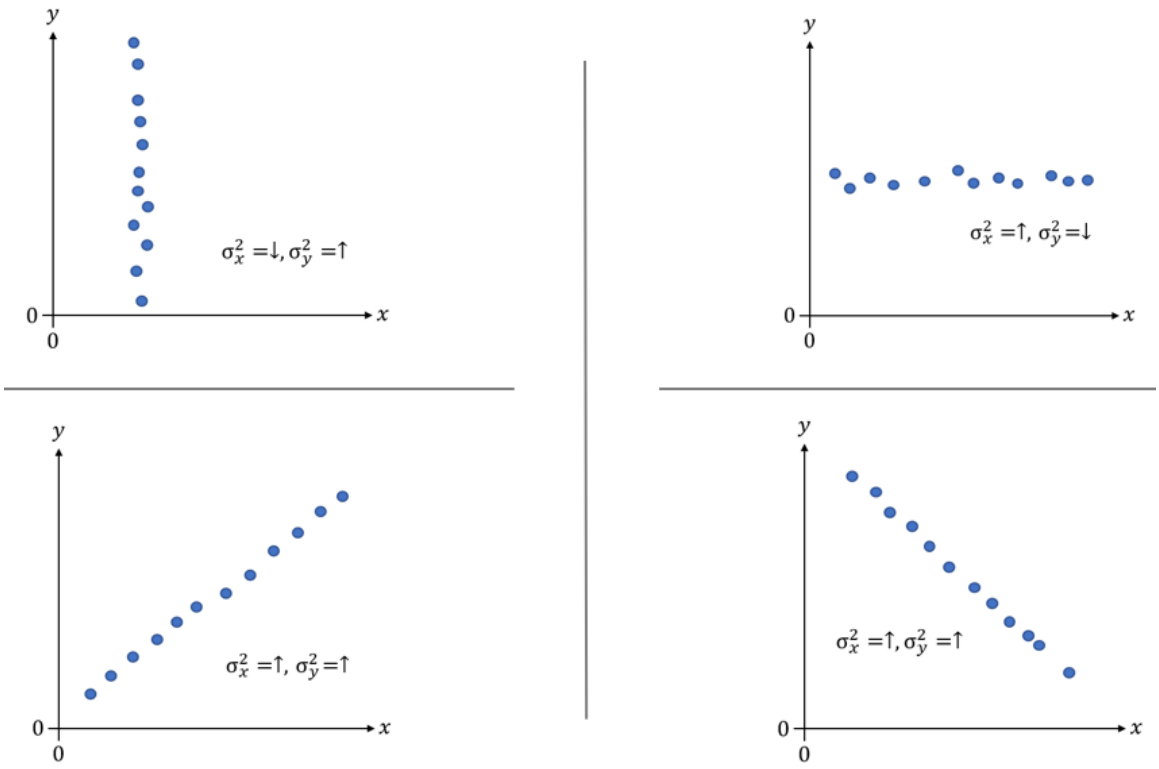

Figure 4-23 Illustration of the relationships between variance and data

It can be seen from Figure 4-23 that the two lower diagonal lines accessing the variances will not provide adequate information to determine the shape type. Figure 4-24 shown below is the same two diagonal lines with eigenvalues applied. One can conclude the eigenvalues are a good indicator for diagonal lines.

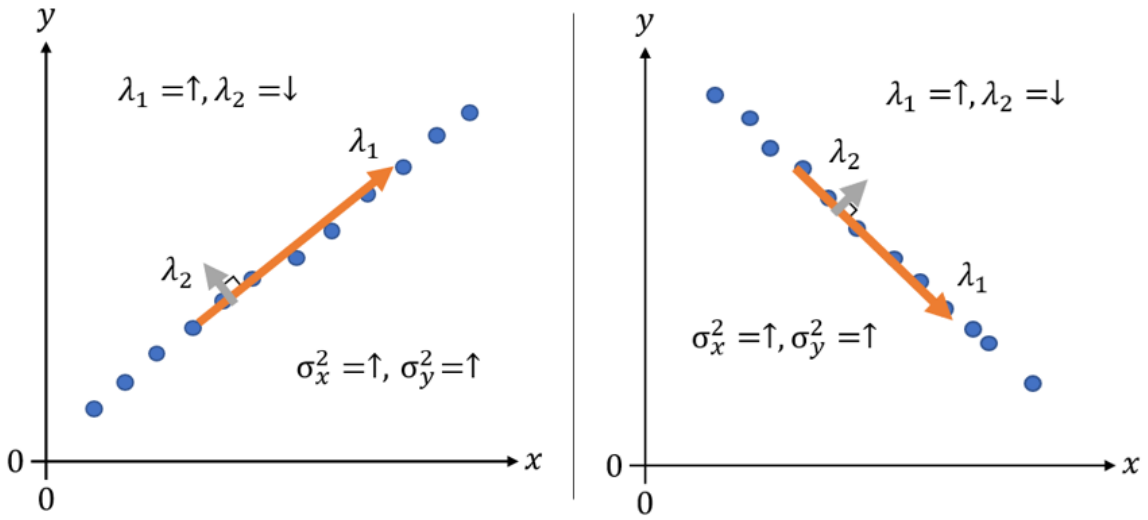

Figure 4-24 Eigenvalues in comparison with variance of diagonal lines

When the data is dispersed nearly uniformly, one can see that both eigenvalues become large along with the variances.

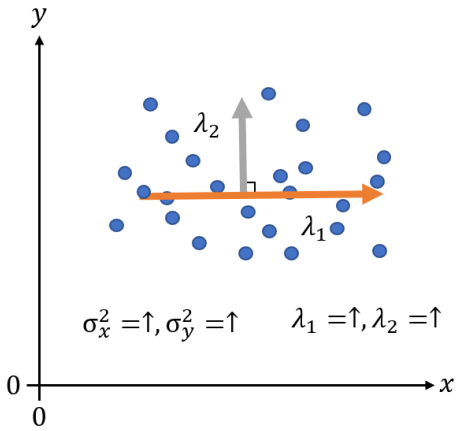

Figure 4-25 Example data when both Eigenvalues are large

Once the decision is made that the data set under test is a line shape, further processing is implemented to retrieve the line of best fit. The choice of linear regression is Theil Sen Robust line segmentation which is less sensitive to outliers than least square regression. Shown below is a diagram that illustrates how sensitive least squares linear regression is to outliners.

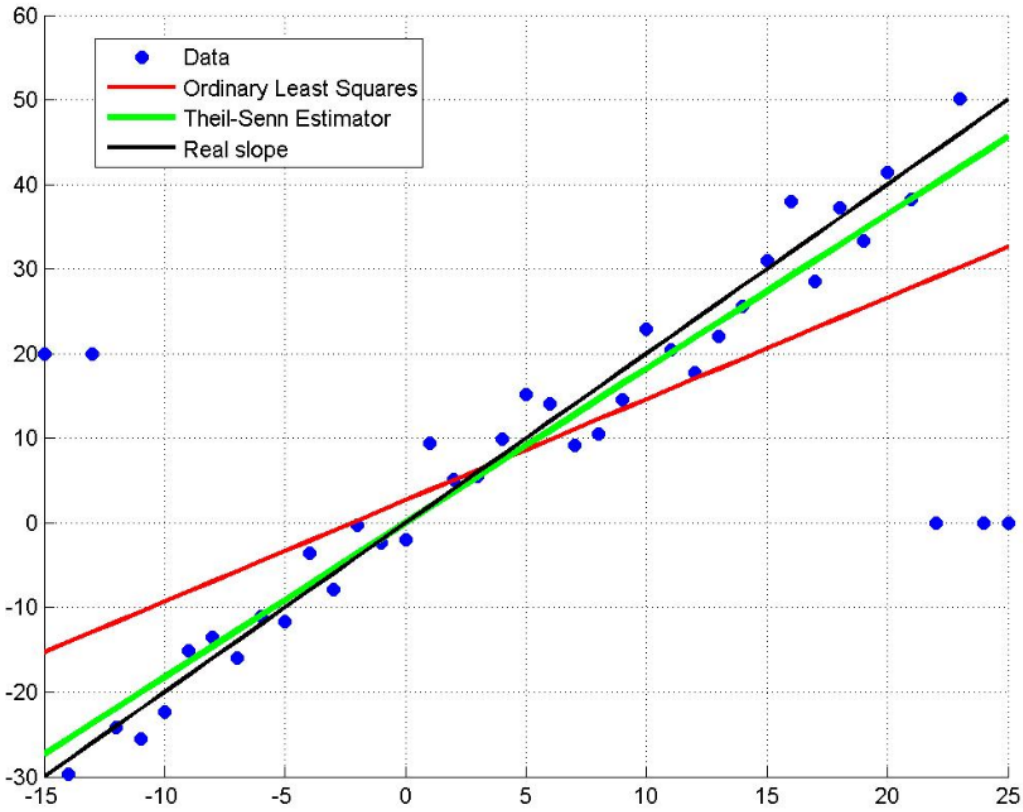

Figure 4-26 Thiel Sen compared to least squares linear regression, taken from [29]

Thiel Sen Robust line segmentation can more accurately predict the line of best fit as the slope value is measured between all sets of adjacent points $Point_i \; and \; Point_{i+1}$ to $Point_{n-1}$ where n is the number of samples. Once all the slopes are calculated, they are arranged in ascending order. The median value is selected as the slope [30] [31].

$$m_{ij} = \frac{(Y_j - Y_i)}{(X_j - X_i)}, for \; i = 1 \; to \; n-1, and \;\; j = 2 \; to \; n$$

Equation 22, Thiel Sen slope calculation

$$n = \; the \; number \; of \; data \; points$$

The denominator of the slope calculation runs the possibility of equating to zero if $X_j and \; X_i$ are equal, which would result in an undefined number. In this case, where the subtraction of $X_j and \; X_i$ equal zero, the calculation of the slope is avoided, and the number of slope entries is reduced by one.

4.5.6. L-Shape

The L-shape is a shape of high value, as vehicles, barriers, and buildings form the L-shape. During the clustering phase, four points were stored as potential vertices for L-shape extraction by saving the greatest $x\ and\ y$ values, and the minimum $x\ and\ y$ values. Depending on the rotational quadrant of the HDL 32E, these saved values become potential candidates for vertices of the L-shape. [32] proposes the algorithm for fitting an L-shape. The L-shape image below illustrates how the four most exterior points are labeled ($V_{Left}, V_{Right}, V_{Up}, \& V_{Down}$). The first step is to find the two vertices that nearly overlap. In the image shown below, it is evident that V_{Left} and V_{up} nearly overlap. These points are labeled Vs1 and Vs2 and the geometric center is found and set as V_A. Whereas, the two other vertices (V_{Right} & V_{Down}) are labelled Vr1 and Vr2.

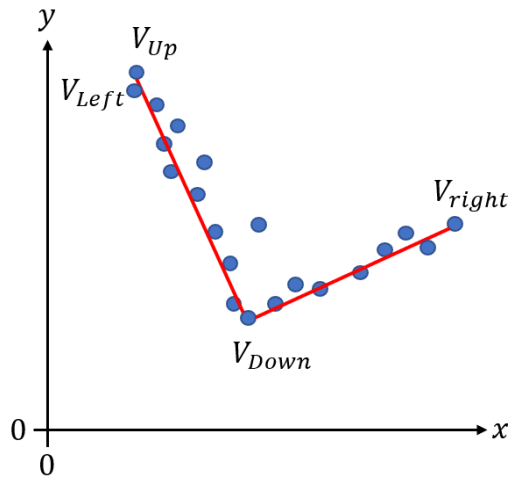

Figure 4-27 L-shape fitting illustration

The angle is then computed by going from V_A to V_{r2} to V_{r1}, then by going to V_A to V_{r1} to V_{r2}. Whichever pattern exhibits the larger angle, the patterns last term is labeled V_B. In the example below (Figure 4-28) $V_B = V_{r2}$.

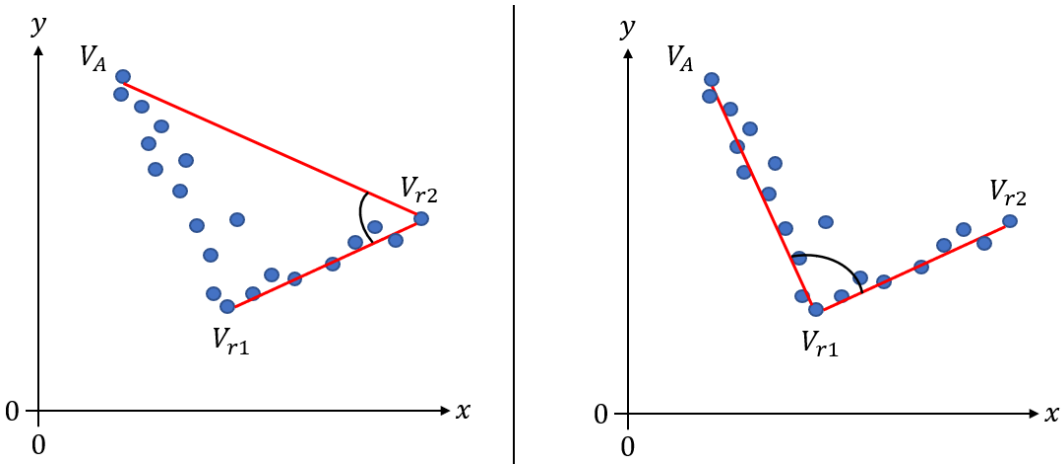

Figure 4-28 Visualization of L-shape maximum angle finding

[32] provides a second algorithm than can be implemented in order to find the data point that produces the closest angle to ninety degrees.

Once the L-shape vertices are known with the angle, the vertices are used to create a triangle. The triangle is then converted to Barycentric coordinates to see how many points reside inside the triangle. To know if a point resides in the triangle, we must solve for V_1 and V_2. Once V_1 and V_2 are solved, three conditions must be satisfied for a point to reside in a triangle, shown in Equation 25.

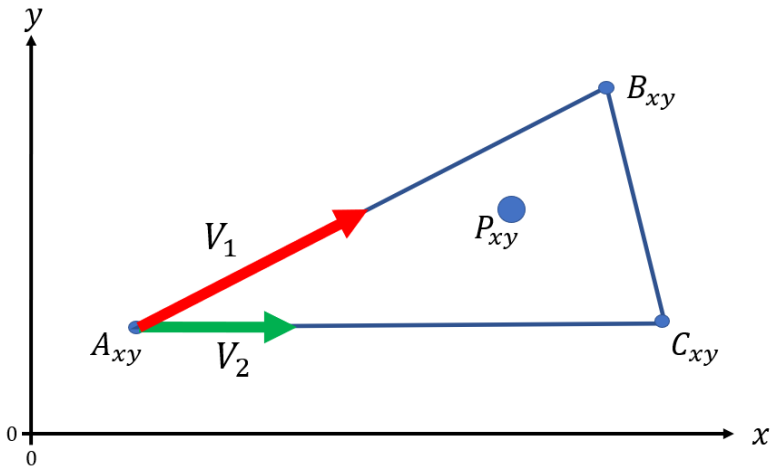

Figure 4-29, Barycentric coordinates

$$V_1 = \frac{A_x(C_y - A_y) + (P_y - A_y)(C_x - A_x) - P_x(C_y - A_y)}{(B_y - A_y)(C_x - A_x) - (B_x - A_x)(C_y - A_y)}$$

Equation 23, Barycentric coordinates for V1

$$V_2 = \frac{P_y - A_y - V_1(B_y - A_y)}{C_y - A_y}$$

Equation 24, Barycentric coordinates for V2

Conditions for a point to exist inside the triangle must be true

$$V_1 \geq 0$$

$$V_2 \geq 0$$

$$V_1 + V_2 < 1$$

Equation 25, Barycentric conditions

For a data set to be accepted as an L-shape, the percentage of points within the triangle must exceed the threshold set by the user,

$$Points_{Percentage} < Threshold$$

Equation 26, Triangle percentage test

4.5.7. Algorithm 4 and 5: Modified Graham Scan, Convex Hull

A common algorithm for finding the outer edge points in a data set is known as Graham scan. Graham scan solves the convex hull which can be interchangeably called the polygon. Traditional applications of Graham scan always start at the lowest "Y" value regardless of the edge position. This method is an excellent approach when the data set is scattered, and no information is previously known [34]. The implementation in this research attempts to exploit the previous arrangement of the LIDAR data completed in Stage One. The added benefit is that horizontal data is already known by column values. As the memory cluster array provides the leftmost and rightmost column, it is known that the outer edges must be in these columns. The row information does not add value, as the Z component is ignored for shape extraction. The modified version of Graham scan always starts at the leftmost column. Every column is treated as scattered data, knowing the point that is closest and furthest from the vehicle must exist in the row, if data from the cluster exist in the row. All rows are sorted finding the smallest "Distance" value (retained from the cylindrical coordinates) and placing the closest value on the stack, and the column than increments to the right one place sorting for the smallest "Distance" value and placing the second value on the stack. The process is repeated for the third column where this value is not placed on

the stack until the calculated cross product results in a CCW rotation. If a CW is detected, the previous point is popped off the stack. And the test proceeds until the rightmost column is calculated. The rightmost column is tested for more than one value if the rightmost column has more than one data point, the rows are sorted for the greatest value where the point is than tested and accepted if CCW rotation is present. The rightmost column is then decremented until the value equals the leftmost column, sorting for the highest value Y point in each column along the way, that satisfies the CCW acceptance test. At the end a polygon is formed with the outermost points following the form of a CCW rotation.

The cross product is used to determine the angle between the two vectors produced by the three points.

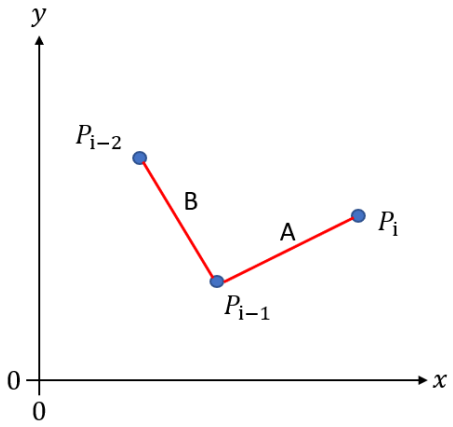

Figure 4-30 Cross product calculation

Where the cross product can be solved by the following,

$$\overline{P_{\iota-1}P_{\iota}} \times \overline{P_{\iota-1}P_{\iota-2}} = \left(x_{(p_{i-1}-p_i)} \cdot y_{(p_{i-1}-p_{i-2})}\right) - \left(x_{(p_{i-1}-p_{i-2})} \cdot y_{(p_{i-1}-p_i)}\right)$$

Equation 27, Cross product test for three points in two dimensional cartesian coordinates

If the result is positive, we can conclude the tested points are making a counterclockwise (CCW) turn, if the result is negative a clockwise (CW) turn has been made. If all results are positive the algorithm will move forward through the points creating a stack to store the points. As soon as a clockwise detection is made the last point on the stack is removed and the current point under test is reexamined for a CW or CCW rotation with the last two data points placed on the stack. If the rotation returns CW the next point on the stack is removed and retested and checked until a CCW rotation is satisfied.

Algorithm 4: Modified Graham Scan

Input: Left and Right most column, Lowest and highest row

```
/* Data is contained in the memory array and must be accessed */
```

Output: All outer edge points that form a convex hull

```
1  start = Left most column, count = 0, dataflag = 0
/* Testing the bottom half of the convex hull */
2  for (start; start < Right most Column; start += 1) do
3      rowLow = Lowest most row
4      for (rowLow; rowLow < highestrow+1; rowLow += 1 ) do
5          if the row and column cluster number = cluster number under
            test then
6              if row and column Y value is < lowest value then
7                  lowest value = row and column value
                   /* sort row for lowsest value */
8                  dataflag = 1;
                       // Need to calculate the Max distance if facing the ground
9      if dataflag == 1 then
10         Call CW CCW test function
/* Trivial case where if the right most side has more than two values the
   highest Y value must be part of the polygon shape */
11 if Right most column has more than two row values then
12     Sort for highest Y value Push highest Y value on stack  Call CW
        CCW test function
/* second half testing the top half of the convex hull */
13 start = Right most column - 1
14 for (start; start >= Left most Column; start -= 1) do
15     rowLow = Lowest most row
16     for (rowLow; rowLow < highestrow+1; rowLow += 1 ) do
17         if the row and column cluster number = cluster number under
            test then
18             if row and column Y value is > Highest Y value then
19                 Highest value = row and column value
                   /* sort row for Highest value */
20                 dataflag = 1;
                       // Need to calculate the Max distance if facing the ground
21 if Left most column has more than two row values then
22     Sort for highest Y value Push highest Y value on stack  Call CW
        CCW test function
```

Algorithm 5: Executes a cross product test for Clock wise (CW) or Counter Clock Wise(CCW) rotation and maintains the stack

```
Function: CW CCW test function
Input: Stack data and new point under test
Output: Resolved stack
1  LASER(i) ← i = 0, 1, ..., 31
2  pkt(i) for i = 1 to N                                  // Parse Packets 1 to N
   /* Ground Point Removal Step (GPR)                                          */
3  count += 1;
                                          // increment count data has been found
4  if count < 2 then
5      Push value on stack
6      dataflag = 0;   // the first two values must be pushed on the stack as
         the cross product requires 3 point for test
7  else
8      while dataflag == 1 do
9          cross product test with last two points on the stack and point
             under test for CCW rotation
10         if direction == CCW then
11             Push value on stack
12             dataflag = 0;
13         else
               /* direction == CW                                              */
14             Pop previous value off the stack
             // the first two values must be pushed on the stack as the cross
           product requires 3 point for test
```

4.5.8. Result of Algorithms 4 and 5 Processing

After Algorithms 4 and 5 have been executed by the modified Graham scan algorithm, the outer perimeter LASER return values are made available. These outer perimeter points could be used in conjunction with sensory fusion algorithms which is beyond the scope of the research presented here.

CHAPTER 5
Results

5.1. Manual Evaluation of Algorithms

5.1.1. Dynamic Cluster Update System

As stated in Section 4.4.5, when a UDP data packet undergoes the three stages of the algorithm the cluster and attributes (Top and lowest most row, left and right most row, Vertex locations, and shape type) are subject to change based on newly available data. As future UDP data packets are extracted, the new data has the potential to become part of a cluster from the previous data packet. When this newly arrived data is found, the new cluster information must be updated and the whole cluster must be reassessed. Figure 5-1 provides an example of Cluster #1's attributes (shown left) when only a single data packet is extracted, and when two UDP data packets are extracted (shown right). It is noticeable that multiple attributes have been updated.

Cluster # 1, Single UDP Data Packet

```
 CLuster Number = '1', CLuster Points = '19',
Shape Type = '3',
CLuster left col = '0',
CLuster right col = '11',
 CLuster low row = '0',
CLuster high row = '1',
 vertex Up X = '-2522.349854',
 vertex Up Y = '-2622.025635',
 vertex down X = '-2595.983398',
 vertex down Y = '-2812.255371',
 vertex left  X = '-2661.808350',
 vertex left  Y = '-2766.995117',
 vertex Right X = '-2412.572998',
 vertex Right Y = '-2704.959717',
```

Cluster # 1, Two UDP Data Packets

```
 CLuster Number = '1', CLuster Points = '43',
Shape Type = '3',
CLuster left col = '0',
CLuster right col = '23',
 CLuster low row = '0',
CLuster high row = '1',
 vertex Up X = '-2632.348389',
 vertex Up Y = '-2519.044922',
 vertex down X = '-2595.983398',
 vertex down Y = '-2812.255371',
 vertex left  X = '-2775.213623',
 vertex left  Y = '-2655.760986',
 vertex Right X = '-2412.572998',
 vertex Right Y = '-2704.959717',
```

Figure 5-1 Dynamic cluster update, cluster #1 is updated between data packets

5.1.2. Ground Point Removal (GPR)

GPR is driven specifically by the mounting height of the LiDAR sensor. The image below illustrates two UDP data packets approximately ten degrees of azimuth sensor rotation. The asterisk is used to show where data is present, and a blank space is used where invalid data points are present. The image on the left illustrates where ground points are present whereas the image on the right shows the same data packets with the ground point removal activated. The GPR is only applied to the 22 negative vertical angle LASERS (0-21). The other 10 LASERS positive angle or

0 degrees are evaluated using the max LASER distance of the Velodyne HDL 32E which is 70 meters.

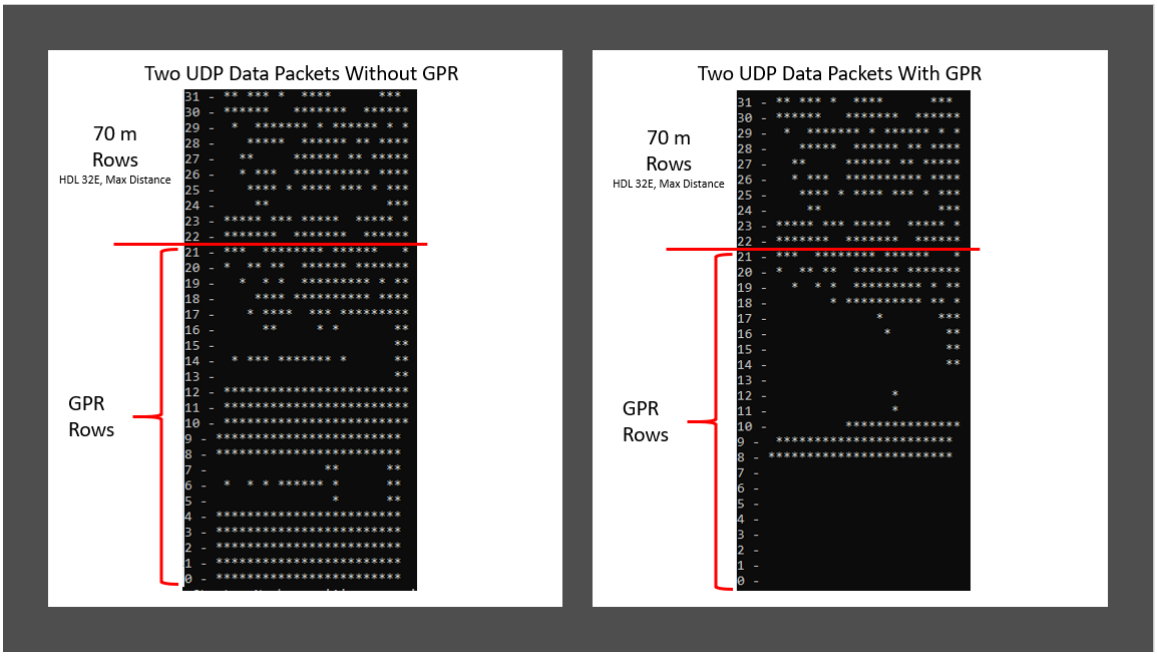

Figure 5-2 Ground point removal, the left side without GPR and the right side with GPR where "" = data and " " = no data*

5.1.3. Segmentation

Segmentation is the process of grouping LiDAR data together where every group known as a cluster pertains to a single object. As discussed earlier, the determination factor is the distance between LASER returns utilizing the ABD [7]. If an object contains more data than just what belongs to the object, the cluster is said to be over segmented. An example of an over segmented object would be if a car and a tree were considered part of the same cluster. If the object is a greater size than a single cluster, the cluster is said to be under segmented. An example of under segmentation would be if the vehicle had multiple clusters assigned to it, when the result should be only one cluster. Segmentation values can be tuned by adjusting the tunable parameters from the ABD. Tuning the LiDAR for segmentation would require access to a real hardware LiDAR sensor and the ability to scan a room or environment with known distance to objects and to evaluate and adjust parameters as needed. [6] [8] Use of the methods of adaptive breakpoint detection for clustering has proven to be a robust method of segmentation. The image below illustrates the segmentation of two Velodyne UDP packets from the HDL 32E. The data points are arranged in vertical and horizontal order just like photograph.

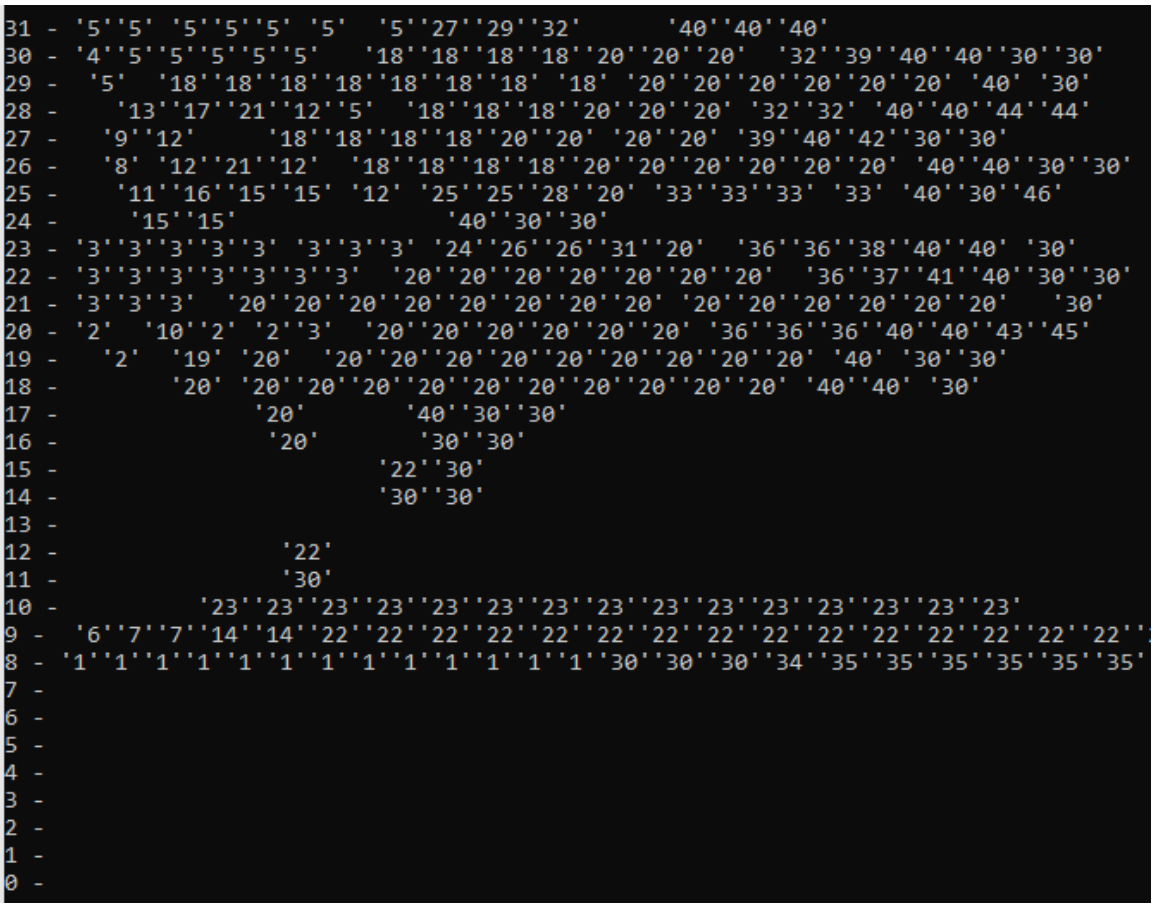

Figure 5-3 Two UDP packets segmented, numeric value reveals cluster identification number

5.1.4. Point Shape

Any cluster with a single point is considered a point. Points are possible with multiple entries of LASER data if the calculated variance of both X and Y are less than the threshold entered by the user, or the calculated Eigenvalues both one and two are less than the specified threshold set up by the user.

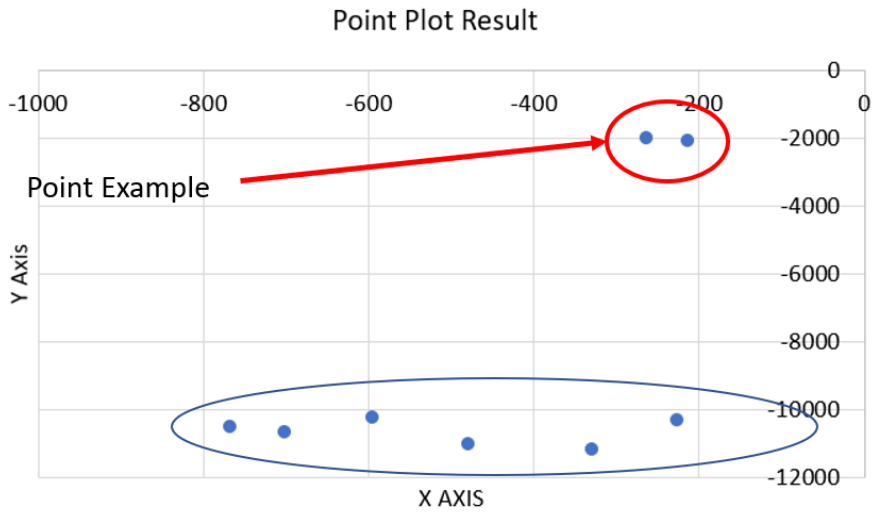

Figure 5-4 Multiplied LASER returns that are classified as a point

5.1.5. Line Shape

Line shapes are found by examining the variance in both the X and Y direction as well as examining the computed Eigenvalues. The image below illustrates data points that were classified as a line.

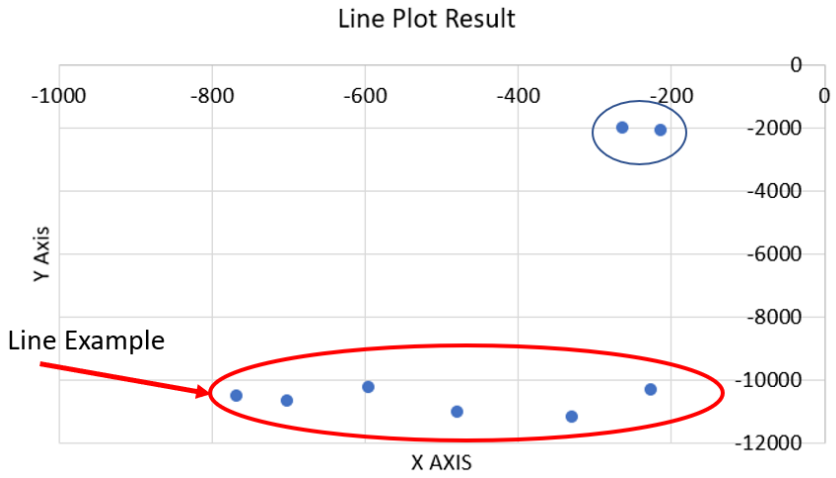

Figure 5-5 Example of data classified as a line

5.1.6. L Shape

The image below illustrates LASER return points that were clustered and considered an L shape.

```
matrix location 0, X = -355, Y = -8068
matrix location 1, X = -412, Y = -8062
matrix location 2, X = -468, Y = -8059
matrix location 3, X = -529, Y = -8059
matrix location 4, X = -588, Y = -8051
matrix location 5, X = -640, Y = -7976
matrix location 6, X = -642, Y = -8003
matrix location 7, X = -646, Y = -8046
matrix location 8, X = -648, Y = -8081
matrix location 9, X = -698, Y = -7979
matrix location 10, X = -699, Y = -7998
matrix location 11, X = -699, Y = -8000
matrix location 12, X = -702, Y = -8024
matrix location 13, X = -706, Y = -8080
matrix location 14, X = -756, Y = -7972
matrix location 15, X = -757, Y = -7985
matrix location 16, X = -760, Y = -8013
matrix location 17, X = -765, Y = -8063
```

Figure 5-6 Data points that were classified as an L-shape

The following information was computed and calculated for the most left, right, up, and down vertex.

```
 CLuster Number = '18',  row with the most data points = '0', CLuster Points = '18',
Shape Type = '3',
CLuster left col = '4',
CLuster right col = '11',
 CLuster low row = '26',
CLuster high row = '30',
 CLuster distance= '0',
 vertex Up X = '-756.443115',
 vertex Up Y = '-7972.639160',
 vertex down X = '-648.787292',
 vertex down Y = '-8081.308594',
 vertex left  X = '-765.035950',
 vertex left  Y = '-8063.204102',
 vertex Right X = '-355.114532',
 vertex Right Y = '-8068.815918',
```

Figure 5-7 Vertex points that were calculated for the L-shape test

The result of the algorithm is illustrated in the image below.

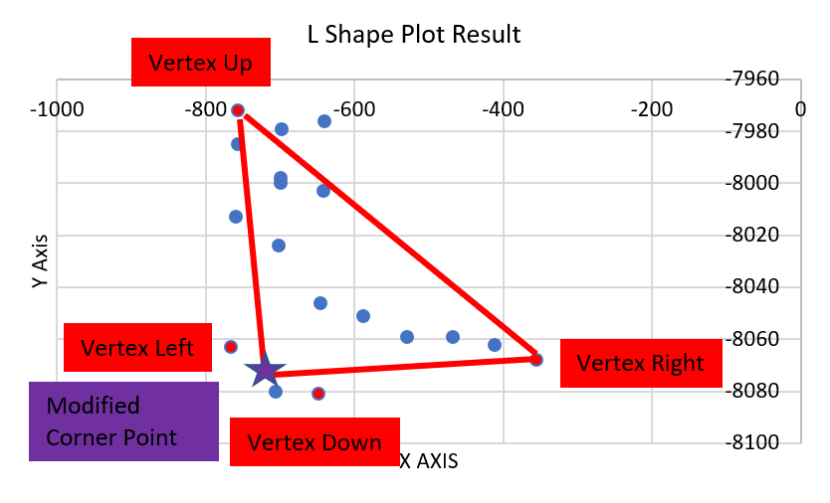

Figure 5-8 Manual evaluation of L-shape

5.1.7. Convex Hull

The image below is a plotted representation of data points returned from a cluster whose shape was considered a polygon.

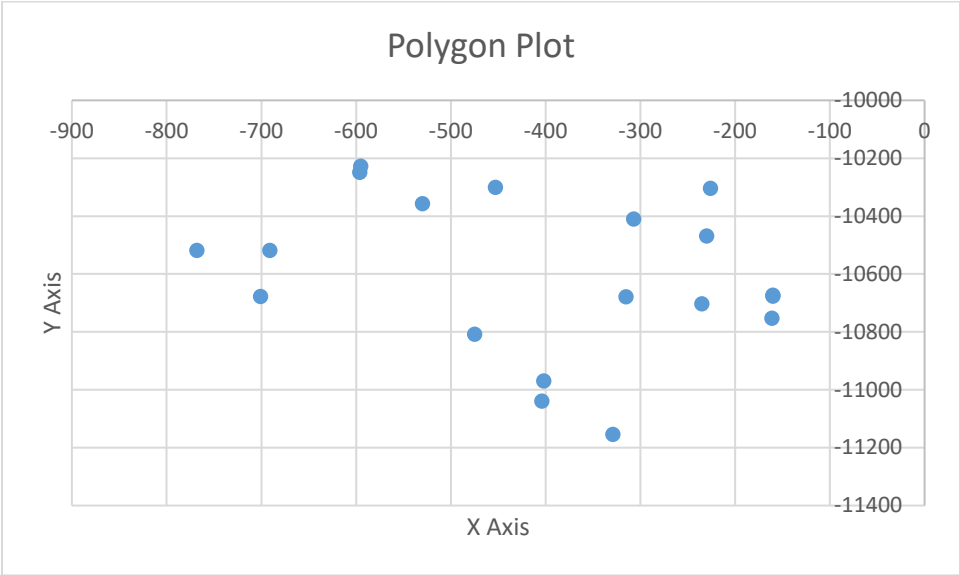

Figure 5-9 Returned plotted data points for a polygon

The goal of the polygon or convex hull shape extraction algorithm is to extract the data points that form the exterior of the hull. The red line illustrates the outermost points in the hull and the points that should be returned to prove the algorithm is correctly functioning.

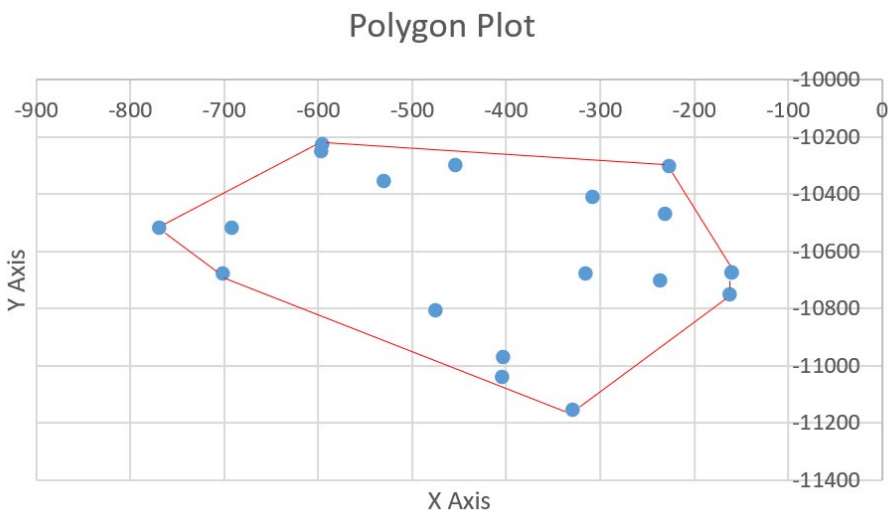

Figure 5-10 Convex hull/ polygon perimeter path marked

The following points are the returned points from the convex hull algorithm.

```
X point = -160.273788, Y point = -10677.191406, Col location = 0
X point = -161.425095, Y point = -10753.889648, Col location = 1
X point = -329.128113, Y point = -11155.145508, Col location = 2
X point = -701.798828, Y point = -10678.831055, Col location = 3
X point = -768.844177, Y point = -10519.942383, Col location = 4
X point = -595.156006, Y point = -10228.700195, Col location = 5
X point = -226.648911, Y point = -10304.730469, Col location = 6
```

Figure 5-11 Returned data points to form the convex hull. The plot of the points below proves the algorithm is functioning as intended returning the outermost points.

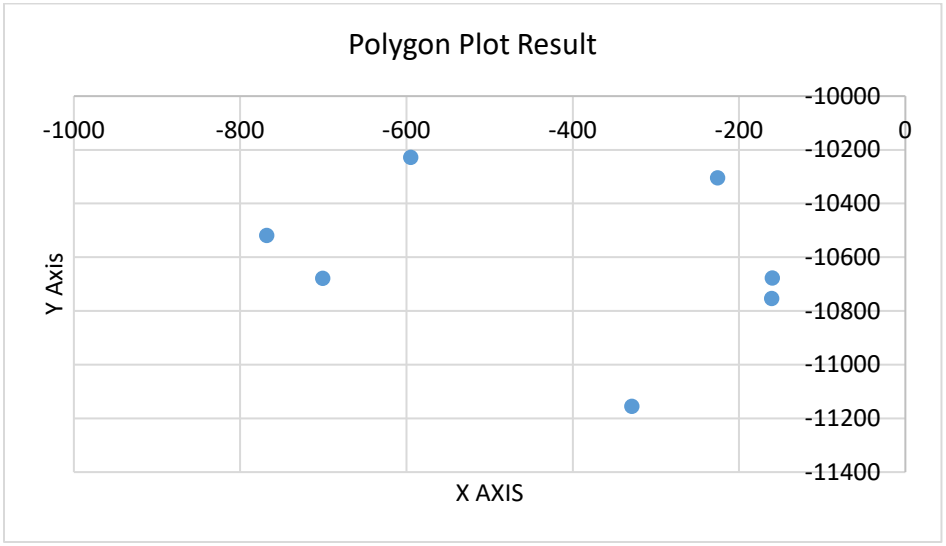

Figure 5-12 Plotted returned data points to form the convex hull

5.2. *Timing Analysis*

The code execution times were recorded executing 200 UDP packets from the Velodyne HDL 32E (~two and a half 360-degree rotations). The timing executions were completed in three methods. Method one is the time taken to complete the UDP packet parsing/organization. Method two is the time taken to complete the UDP packet parsing/organization and clustering of the 200 packets. Method three is the total time taken to complete the UDP packet parsing/organization, clustering, and shape extraction of 200 packets.

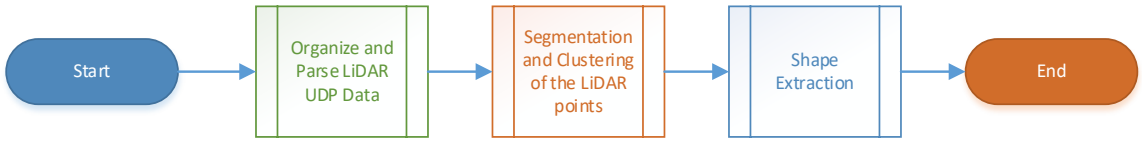

Figure 5-13 Process of the software stages that every Velodyne HDL 32E UDP data packet must complete

5.2.1. *Preparing the Windows Machine for Timing Evaluation*

Timing analysis is very difficult on a Windows computer as the microprocessor is not dedicated to a single task such as a microcontroller, application specific integrated circuit (ASIC), or FPGA chip is. In order to set priority timing to the execution time of this code the priority was set to "Realtime" in the task manager before execution. By setting the Windows personal computer (PC) to Realtime, approximately seven milliseconds were shaved off the total execution time. In the case of the Intel I7 processor this priority setting to Realtime accounted for approximately a nine percent timing gain.

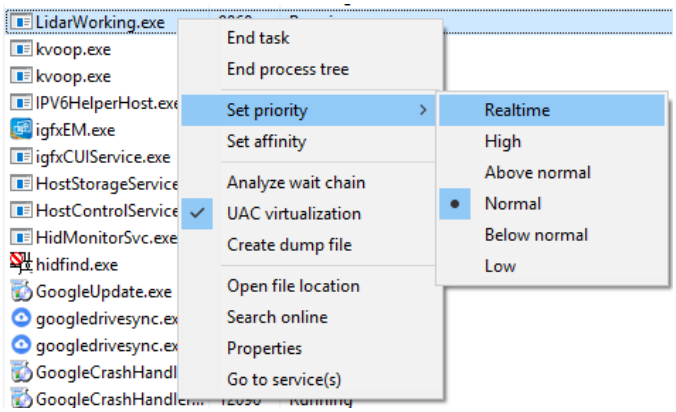

Figure 5-14 Setting processor priority on Windows operating system

5.3. *Performance of the Code on Different Machines*

Code execution was performed on two different Windows computers to provide an example of algorithm runtime on different machines. Both computers executed the same code and examine the same 200 Velodyne HDL 32E UDP data packets. The code was executed on Code blocks 17.12 on both machines.

Manufacturer	**Dell Latitude E5570**	**Dell Latitude E5570**
Processor	Intel I7- 6820HQ CPU at 2.7 GHz Processor, 4 cores	Intel I5 – 6440HQ CPU at 2.6 GHz Processor, 4 cores
RAM	16 GB	8 GB
Hard Drive	SSD hard drive model SK Hynix SC311 SATA 512 GB	Toshiba MQ01ACF050 500 GB SATA-600, SATA 6GB/s 7278 RPM
Operating System	Windows 10 Enterprise Edition 64 bit	Windows 10 Enterprise Edition 64 bit
Differentiator	Dell I7	Dell I5

Table 5-1 Personal computers used for testing

5.4. *Dell I7 Timing Analysis*

5.4.1. *Dell I7 Execution Times*

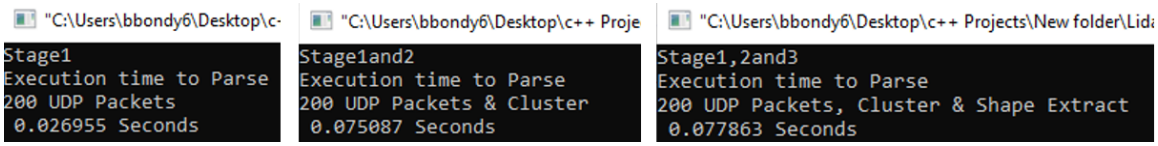

Figure 5-15 Dell I7, timing snippets from code execution

5.4.2. *Dell I7 UDP Packets Parsing Data*

Total time to Parse 200 UDP Packets is 26.955 milliseconds

Mean Values

$$UDP\ Packet\ Parsing\ Mean = 134.957\ \mu\ Seconds$$

5.4.3. Dell I7 Clustering and Segmentation

Total time to cluster and segment 200 UDP Packets is 48.132 milliseconds

Mean Values

$$Clustering\ \&\ Segmentation\ Mean\ per\ UDP\ packet = 240.66\ \mu\ Seconds$$

5.4.4. Dell I7 Shape Extraction

Total time to perform shape extraction on 200 UDP Packets is 2.776 milliseconds

Mean Values

$$Shape\ Extraction\ Mean\ per\ UDP\ packet = 13.88\ \mu\ Seconds$$

Percentage of time spent by category,

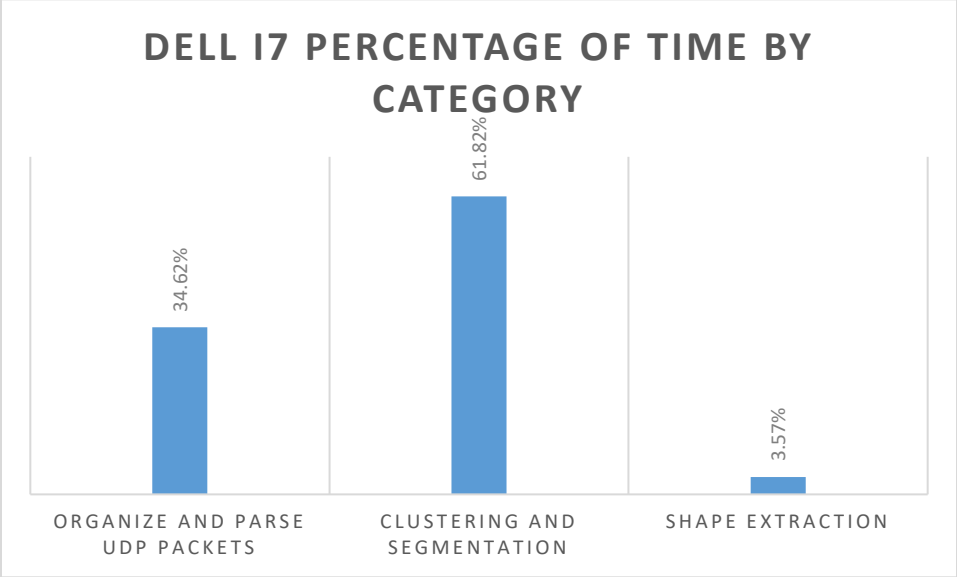

Figure 5-16 Dell I7, bar graph percentage of time by category

5.4.5. Dell I7 Average Execution Time per UDP Packet

Total time to perform all three stages (Parse UDP packets, Cluster, and Shape extraction) on a single UDP packet produced from the Velodyne HDL 32E.

$$UDP\ Packet\ Mean = 389.497\ \mu\ Seconds$$

5.5. Dell I5 Timing Analysis

5.5.1. Dell I5 Execution Times

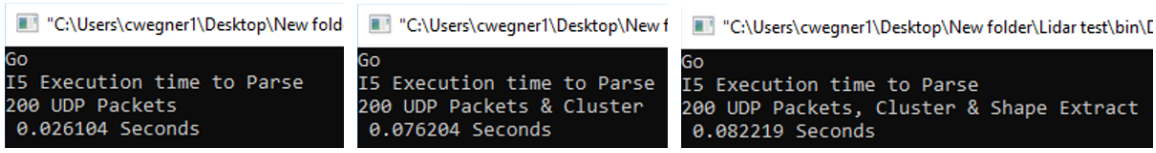

Figure 5-17 Dell I5, timing snippets from code execution

5.5.2. Dell I5 UDP Packets Parsing Data

Total time to Parse 200 UDP Packets is 26.104 milliseconds

Mean Values

$$UDP\ Packet\ Parsing\ Mean = 130.52\ \mu\ Seconds$$

5.5.3. Dell I5 Clustering and Segmentation

Total time to cluster and segment 200 UDP Packets is 50.1 milliseconds

Mean Values

$$Clustering\ \&\ Segmentation\ Mean\ per\ UDP\ packet = 250.5\ \mu\ Seconds$$

5.5.4. Dell I5 Shape Extraction

Total time to perform shape extraction on 200 UDP Packets is 6.015 milliseconds

Mean Values

$$Shape\ Extraction\ Mean\ per\ UDP\ packet = 30.075\ \mu\ Seconds$$

Percentage of time spent by category,

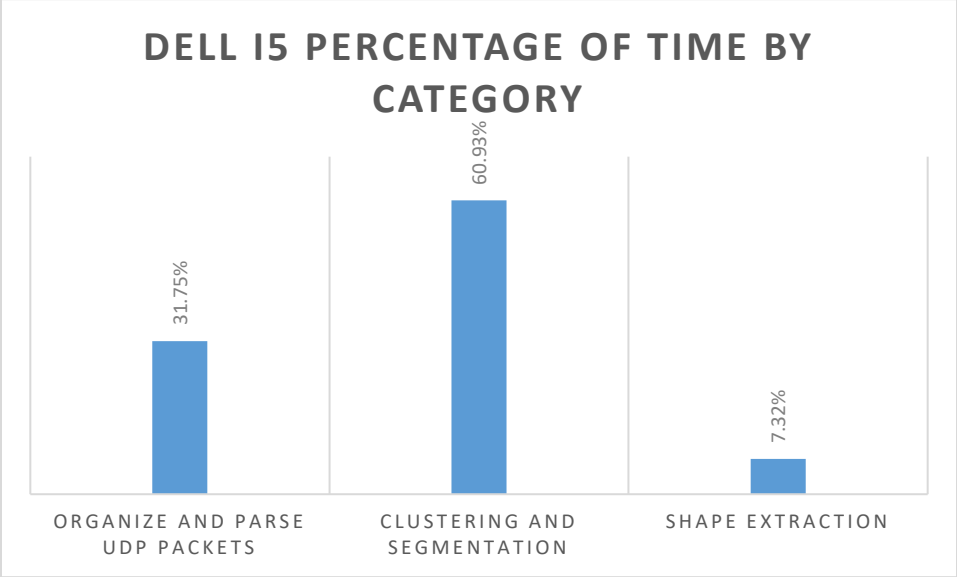

Figure 5-18 Dell I5, bar graph percentage of time by category

5.5.5. Dell I5 Average Execution Time per UDP Packet

Total time to perform all three stages (parse UDP packets, cluster, and shape extraction) on a single UDP packet produced from the Velodyne HDL 32E.

$$UDP\ Packet\ Mean = 411.095\ \mu\ Seconds$$

5.6. Comparison of Both Windows PC Machines

The chart below illustrates the timing required for every stage on the two different personal computers used in this research.

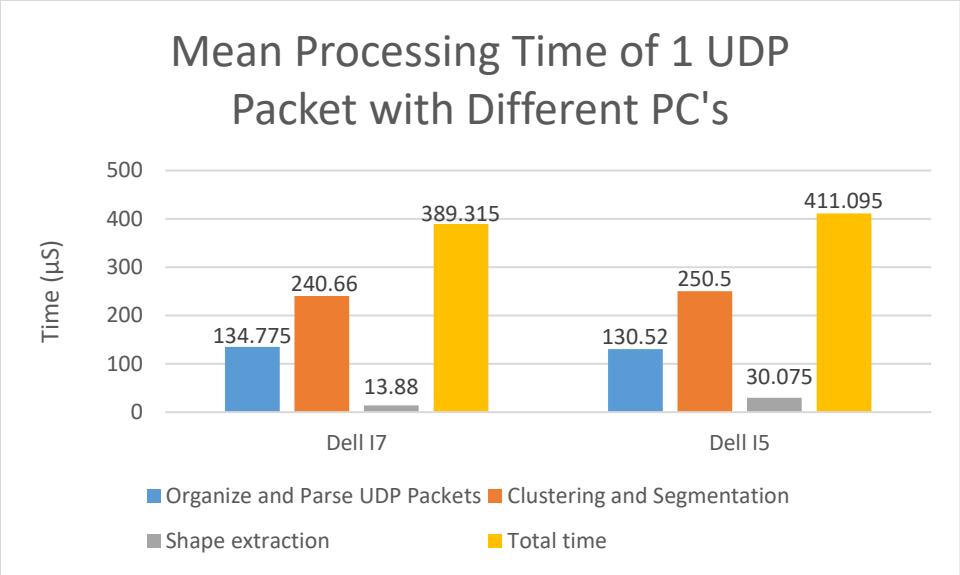

Figure 5-19 Mean processing time of 1 UDP data packet with two different PC's (Dell I7, Dell I5)

5.7. Timing Comparison to Related Work

[6] tested 2d+ Lidar data utilizing MATLAB. The research from [6] did not state the sensor type used in their research, only that the sensor produces 100,000 LiDAR data points per second, whereas the LiDAR sensor used in this research produces 700,000 points of LiDAR data per second (Velodyne HDL 32E). [6] Delivered an average packet completion time of 287 milliseconds. The completion time included a pre clustering row by row stage, pre shape extraction row by row, a row and shape merging stage, and a final shape extraction stage.

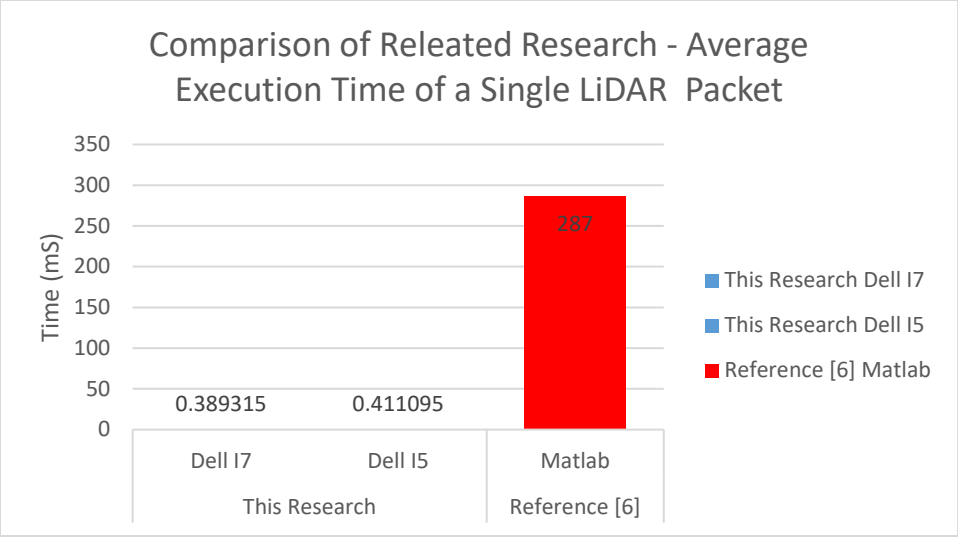

Figure 5-20 Comparison of related research showing the difference of the average execution time of a single LiDAR UDP data packet

The future comparisons will only consider the time of the I7 computer as the fastest execution time was provided from this Windows machine.

Number of data packets that the algorithm completed in this research compared to a single packet in [6].

$$\frac{Reference[6]\ Single\ Pkt\ Execution\ time}{This\ Research\ Single\ Pkt\ Execution\ time} = \frac{287\ mS}{0.389315\ \mu S} = 737.19$$

This research proved to execute the complete algorithm on 737.19 UDP data packets for every single data packet executed by [6] who utilized MATLAB for the execution of their algorithm.

If one considered that [6] algorithm processed 100,000 LASER returns per second and that this research processed 700,000 LASER returns per second, one could conclude that this research performed:

$$737.19 \times 7 = 5160.33$$

Concluding that the algorithm in this research performed 5160.33 times faster than the related work of [6].

5.8. *Timing Comparison to Benchmark*

The processing time benchmark discussed in Chapter 3 shows that LiDAR UDP data packets arrive every 552.96 µs on the Velodyne HDL 32E. Comparing the results obtained from the proposed algorithms acceleration on the PC's.

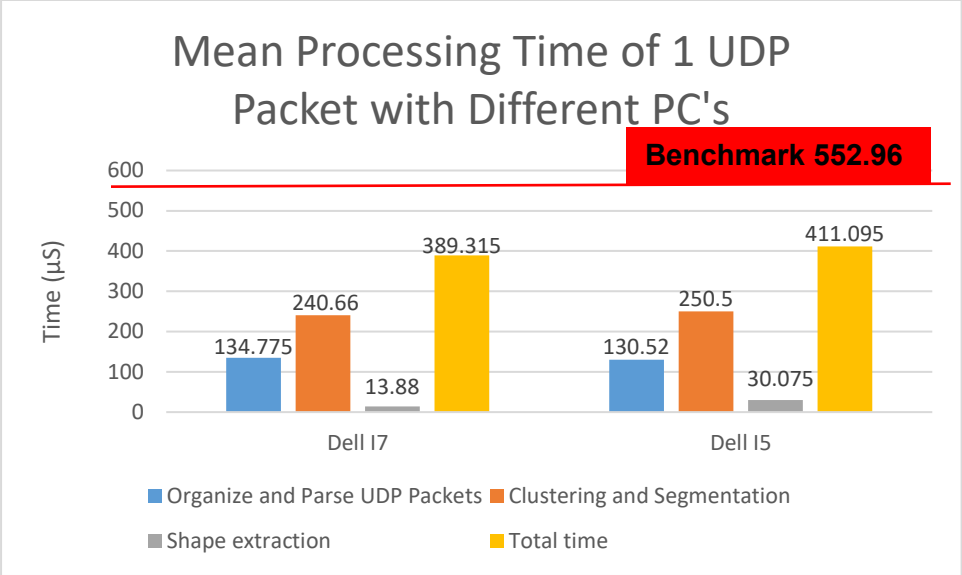

Figure 5-21 Comparison of the PC's tested, average UDP data packet execution time compared to the benchmark time of the UDP data packet arrival time

Time to next packet is 552.96

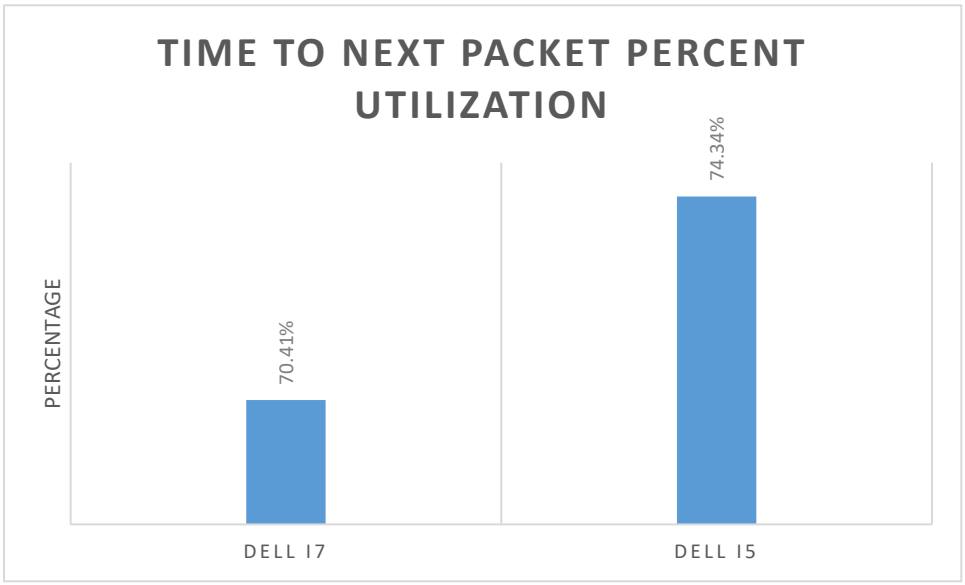

Figure 5-22 Percentage of the time utilized between UDP data packet

CHAPTER 6
Discussion

6.1. Tuning Parameters

The algorithm in this research has several tuning parameters. The tuning of the parameters could greatly improve segmentation of shapes if done correctly.

A list of tuning parameters:

Mounting height of the LiDAR Sensor - The height of the LiDAR sensor was provided by Velodyne for this research. However, when performing the ground point removal stage, the mounting height of the physical sensor must be considered.

Adaptive Breakpoint Detector (ABD) - λ or the maximum incidence angle must be chosen carefully. λ must always be greater than the greatest angle between points ($\Delta\theta$). Choosing the proper λ value must go hand in hand with the greatest distance between points that will ever be searched. The selection of λ will affect the greatest angle between points while trying to avoid occlusions.

Search Points - The number of LASERs layers and rows to look above, below, and left of for occlusion avoidance. The value of "Search Points" can aid greatly in reducing under segmentation of an object when an occlusion is present. The further the distance between points, the greater the threshold circle. Selection of the "Search Points" parameter should be consciously adjusted to avoid sloppy data that will result in over segmentation. As mentioned above, the adjustment of λ must be considered when setting the "Search Points" parameter.

Variance Thresholds – Variance threshold are used to determine line shapes that run parallel to the X or Y axis during shape extraction. The variance threshold must be considered to avoid confusing an L-Shape or Polygon with a line.

Eigenvalue Thresholds – Eigenvalue threshold ($\lambda_1 \; and \; \lambda_2$) are used to determine point shapes that have more than a single LASER return as well as distinguishing horizontal line shapes. The Eigenvalue threshold must be considered to avoid confusing an L-Shape or Polygon with a line.

6.2. Future Work

Tuning of parameters and validation in different automotive environments is necessary with hardware LiDAR sensors. Tuning should be validated in a known environment so that performance could be evaluated for best results.

Implement the simultaneous localization and mapping (SLAM) algorithm. The SLAM algorithm will continuously remap data points. The algorithms in this research that exploit the organization of the LiDAR return data needs to be proved to operate with SLAM running effectively together.

Improvements to the ground point removal stage can be implemented. Traffic lines and street signs are generally made from reflective materials. The intensity of the LiDAR return could be used in turn on the GPR stage to extract road lines that could also aid in autonomous vehicle features.

Clustering. Expanding the candidate set by more than one dimension could improve on azimuth occlusions. If data was not received for a few columns, the data in the row may become too far away for a cluster match. The validation of a cluster match could still be accomplished by looking above and below the current row under test. [8] expanded the candidate set and was able to improve on ghost elimination.

Barycentric Coordinates. The current test forms a triangle that may not include data points that are very close to the triangle or even partially touching the edge of the triangle. Test making the triangle larger to account for points that may be closest to the triangle. [37] proposes an algorithm that considers the thickness of the lines and vertexes in order to reduce error of points close to the boundary of the triangle.

Processing time. This research proved that the data from a LiDAR sensor (Velodyne HDL 32E) could be run in Realtime utilizing a Windows 10 operating system with an I7 or I5 processor. The I7 produced a mean resting time of ~30%. When the SLAM algorithm is added to this research, there is the potential that this algorithm may not be able to keep up with the requirements of operating in Realtime. The algorithms in this research should be accelerated on a dedicated hardware processor perhaps FPGA or GPU.

CHAPTER 7
Conclusion

This thesis provided an algorithm that processes data from the LiDAR sensors ethernet output stream UDP packet parsing, clustering/segmentation, and shape extraction of LiDAR data. The proposed algorithm exploits an algorithm during the first stage that orders the LiDAR LASER data in vertical rows and horizontal columns for subsequent stages to utilize less computationally expensive processing. This algorithm is designed to be able to run in Realtime with a vehicle rooftop mounted rotary three-dimensional LiDAR sensor, such as the Velodyne HDL 32E, which was used in this research. This algorithm is designed with the intent to pass shape data onto an object tracking algorithm as part of an ADAS or autonomous vehicle sensor suite.

The manual assessments proved that the algorithms were functioning as designed and provided proper data as a result. Segmentation results and tuning parameters were assumed based on previous research with positive results. Access to actual LiDAR hardware was unobtainable during this research, therefore tuning parameters could not be finely adjusted.

The execution timing analysis proved that a Windows I7 or even I5 machine running the Windows 10 operating system could process the LiDAR data at a faster rate than the Velodyne HDL 32E produced the UDP data packets. The Velodyne HDL 32E produces UDP data packets at a rate of 552.96 μS per packet, whereas the I7 Windows machine was able to complete the parsing, clustering and shape extraction of UDP data packets at a mean rate of 389.47 μs when a 200 UDP data set was tested. The I5 Windows machine was able to provide a data packet mean at 411.09 μs when tested with the 200 UDP data packet set.

REFERENCES

[1] Grundstein, Andrew & Meentemeyer, Vernon & Dowd, John. (2009). Maximum Vehicle Car Temperatures under Different Meteorological Conditions. International journal of biometeorology. 53. 255-61. 10.1007/s00484-009-0211-x.

[2] Guinness Book of World Records, "Lowest temperature -inhabited", -67.6 67.7°C (-90°F), recorded Oymyakon Republic of Russia March 6th, 1933 2019 [Online]. Available: https://www.guinnessworldrecords.com/world-records/lowest-temperature-inhabited?fb_comment_id=957394674288768_2259891097372446 (accessed November 05, 2019).

[3] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Sharlf, Samuel Webb Williams and Katherine A. Yelick, "The landscape of parallel computing research: A view from Berkeley," ACM, pp. 56-57, 2009.

[4] Beth Braverman, "Americans spend 300 hours a tear driving in their cars", Business insider, the Fiscal Times, Sept 8th, 2016 [Online]. Available: https://www.businessinsider.com/americans-driving-300-hours-a-year-2016-9 (accessed November 05, 2019).

[5] Li-Juan Zheng and Yu-Cheng Fan, "Data packet decoder design for LiDAR system," 2017 IEEE International Conference on Consumer Electronics - Taiwan (ICCE-TW), Taipei, 2017, pp. 35-36.

[6] Johansson, Tobias and Oscar Wellenstam. "LiDAR clustering and shape extraction for automotive applications." (2017).

[7] Su-Yong An, Jeong-Gwan Kang, Lae-Kyoung Lee, and Se- Young Oh, "Line segment-Based Indoor Mapping with Salient Line Feature Extraction", Advanced Robotics, 26:5-6,437-460, DOI: 10.1163/156855311x617452 Proposes a Dual Breakpoint detector for clustering points

[8] Kim, Beomseong & Choi, Baehoon & Yoo, Minkyun & Kim, Hyunju & Kim, Euntai. (2014). Robust Object Segmentation Using a Multi-Layer Laser Scanner. Sensors (Basel, Switzerland). 14. 20400-18. 10.3390/s141120400.

[9] Clean Technica, “Autonomous Driving Levels 0-5+ Implications” [Online]. Available: https://cleantechnica.com/2017/12/02/autonomous-driving-levels-0-5-implications/ (accessed November 05, 2019).

[10] Woodside Capital Partners, “ADAS/Autonomous Sensing Industry: Beyond the Headlights Report”, Carolyn Fortuna, September 27th, 2016 [Online]. Available: http://www.woodsidecap.com/wcp-publishes-adasautonomous-sensing-industry-beyond-headlights-report/ (accessed November 05, 2019).

[11] Parrish, Christopher. “Lidar and Height Mod Workshop”, August 18th, 2011 [Online]. Available: https://www.ngs.noaa.gov/corbin/class_description/Parrish_Lidar_and_Height_Mod_Presentation.pdf (accessed November 05, 2019).

[12] RENISHAW “Optical encoders and LiDAR scanning”, [Online]. Available: https://www.renishaw.com/en/optical-encoders-and-lidar-scanning--39244 (accessed November 05, 2019).

[13] Wikipedia, “Laser Scanning” [Online]. Available: https://en.wikipedia.org/wiki/Laser_scanning (accessed November 05, 2019).

[14] First Sensor AG. "The Importance of Avalanche Photodiode (APD) for LIDAR Applications". AZoSensors. [Online]. Available: https://www.azosensors.com/article.aspx?ArticleID=864. (accessed November 05, 2019).

[15] Chevrier, Matt. (2017). How to build a LIDAR system with a time-to-digital converter. [Online]. Available: http://www.ti.com/lit/an/slyt706/slyt706.pdf (accessed November 05, 2019).

[16] Gigahertz-Optik “The wavelength range of Optical Radiation”. [Online]. Available: https://light-measurement.com/wavelength-range/ (accessed November 05, 2019).

[17] Wojtanowski, Jacek & Zygmunt, Marek & Kaszczuk, Mirosława & Mierczyk, Z. & Muzal, Michal. (2014). Comparison of 905 nm and 1550 nm semiconductor laser rangefinders’ performance deterioration due to adverse environmental conditions. Opto-Electronics Review. 22. 10.2478/s11772-014-0190-2.

[18] Jarvis, Albie. “Guide to LiDAR Wavelengths”, November 6th, 2018 [Online]. Available: https://velodynelidar.com/newsroom/guide-to-lidar-wavelengths/ (accessed November 05, 2019).

[19] GISGeography. “A Complete Guide to LiDAR: Light Detection and Ranging”, February 17th, 2018 [Online]. Available: https://gisgeography.com/lidar-light-detection-and-ranging/ (accessed November 05, 2019).

[20] InfoQ. “How Apple Uses Neuaral Networks for Object Detection in Point clouds”, Novemeber 28th, 2017 [Online]. Available: https://www.infoq.com/news/2017/11/apple-voxelnet/ (accessed November 05, 2019).

[21] Novatel. “An introduction to GNSS, Chapter 6”, [Online]. Available: https://www.novatel.com/an-introduction-to-gnss/chapter-6-gnss-ins/ (accessed November 05, 2019).

[22] Wikipedia, “Simultaneous Localization and Mapping” [Online]. Available: https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping (accessed November 05, 2019).

[23] Velodyne, “HDL-32E User Manual” 63-9113 Rev. M [Online]. Available: https://velodynelidar.com/docs/manuals/63-9113%20REV%20M%20MANUAL,USERS,HDL32E.pdf (accessed November 05, 2019).

[24] Law.Resource.org, “Three-Dimensional Reference System” [Online]. Available: https://law.resource.org/pub/us/cfr/ibr/005/sae.j1100.2001.html (accessed November 05, 2019).

[25] Kitware Data, “Velodyne LiDAR PCAP files” [Online]. Available: https://data.kitware.com/#folder/5b7f47568d777f06857cb208 (accessed November 05, 2019).

[26] Public Domain Vectors, “Images are constructed with vectors which are public domain” [Online]. Available: https://publicdomainvectors.org/ (accessed November 05, 2019).

[27] Danilo Caceres Hernandez, Alexander Filonenko, Dongwook Seo, and Kang-Hyun Jo. Lane marking recognition based on laser scanning. In Industrial Electronics (ISIE), 2015 IEEE 24th International Symposium on, pages 962–965. IEEE, 2015.

[28] Vincent Spruyt. “A geometric interpretation of the covariance matrix”, [Online]. Available: https://www.visiondummy.com/2014/04/geometric-interpretation-covariance-matrix/ (accessed November 05, 2019).

[29] Arnout Tilgenkamp. “Theil-Sen Estimator”, Math Works December 2011 [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/34308-theil-sen-estimator (accessed November 05, 2019).

[30] Pranab Kumar Sen. Estimates of the regression coefficient based on Kendall’s tau. Journal of the American Statistical Association, 63(324):1379–1389, 1968.

[31] Granato, G.E., 2006, Kendall-Theil Robust Line (KTRLine—version 1.0)—A visual basic program for calculating and graphing robust nonparametric estimates of linear-regression coefficients between two continuous variables: Techniques and Methods of the U.S. Geological Survey, book 4, chap. A7, 31 p.

[32] S. Qu *et al*., "An Efficient L-Shape Fitting Method for Vehicle Pose Detection with 2D LiDAR," *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Kuala Lumpur, Malaysia, 2018, pp. 1159-1164. doi: 10.1109/ROBIO.2018.8665265

[33] Forsythe, George E. and Peter Henrici. “THE CYCLIC JACOBI METHOD FOR COMPUTING THE PRINCIPAL VALUES OF A COMPLEX MATRIX.” (1960). [Online]. Available: https://www.semanticscholar.org/paper/THE-CYCLIC-JACOBI-METHOD-FOR COMPUTING-THE-VALUES-A-Forsythe-Henrici/0af1fab0b346ea83bae75e0fd9958ed0e62a1520 (accessed November 06, 2019).

[34] Fu, Zhongliang and Yuefeng Lu. “AN EFFICIENT ALGORITHM FOR THE CONVEX HULL OF PLANAR SCATTERED POINT SET.”, 2012 [Online]. Available: https://www.semanticscholar.org/paper/AN-EFFICIENT-ALGORITHM-FOR-THE-CONVEX-HULL-OF-POINT-Fu-Lu/c1fdc3a6ee9fb0c70bd000fa1d47f48c11c0d608 (accessed November 12, 2019).

[35] Ayush Dewan, Tim Caselitz, Gian Diego Tipaldi and Wolfram Burgard, "Motion-based detection and tracking in 3D LiDAR scans," 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, 2016, pp. 4508-4513.

[36] Bryant Walker Smith. “Human Error As A Cause Of Vehicle Crashes.”, The Center for Internet and Society at Stanford Law School, December 18th, 2013 [Online]. Available: http://cyberlaw.stanford.edu/blog/2013/12/human-error-cause-vehicle-crashes (accessed November 12, 2019).

[37] Cédric Jules. “Accurate Point in Triangle Test”, January 25th, 2014 [Online]. Available: http://totologic.blogspot.com/2014/01/accurate-point-in-triangle-test.html (accessed November 12, 2019).

APPENDIX A.

HDL 32E Firing Order Chart

The chart below deciphers the byte order access needed to arrange the Velodyne HDL 32E UDP data packet from the most negative LASER firing angle to the most Positive LASER firing angle.

Access Pattern	Byte Order Access	Length	Firing order	DSR #	Vertical Angle (Degrees)	Radian Angle (Rads)
Laser Block ID	42	2 Bytes	***	***	***	***
Rotational	44	2 Bytes	***	***	***	***
Distance Information	46	2 Bytes	1	0	-30.67	-0.535292482
Intensity	48	1 Byte				
Distance Information	52	2 Bytes	3	2	-29.33	-0.51190507
Intensity	54	1 Byte				
Distance Information	58	2 Bytes	5	4	-28	-0.488692191
Intensity	60	1 Byte				
Distance Information	64	2 Bytes	7	6	-26.66	-0.465304779
Intensity	66	1 Byte				
Distance Information	70	2 Bytes	9	8	-25.33	-0.4420919
Intensity	72	1 Byte				
Distance Information	76	2 Bytes	11	10	-24	-0.41887902
Intensity	78	1 Byte				
Distance Information	82	2 Bytes	13	12	-22.67	-0.395666141
Intensity	84	1 Byte				
Distance Information	88	2 Bytes	15	14	-21.33	-0.372278729
Intensity	90	1 Byte				
Distance Information	94	2 Bytes	17	16	-20	-0.34906585
Intensity	96	1 Byte				
Distance Information	100	2 Bytes	19	18	-18.67	-0.325852971
Intensity	102	1 Byte				
Distance Information	106	2 Bytes	21	20	-17.33	-0.302465559
Intensity	108	1 Byte				
Distance Information	112	2 Bytes	23	22	-16	-0.27925268
Intensity	114	1 Byte				
Distance Information	118	2 Bytes	25	24	-14.67	-0.256039801

Intensity	120	1 Byte				
Distance Information	124	2 Bytes	27	26	-13.33	-0.232652389
Intensity	126	1 Byte				
Distance Information	130	2 Bytes	29	28	-12	-0.20943951
Intensity	132	1 Byte				
Distance Information	136	2 Bytes	31	30	-10.67	-0.186226631
Intensity	138	1 Byte				
Distance Information	49	2 Bytes	2	1	-9.33	-0.162839219
Intensity	51	1 Byte				
Distance Information	55	2 Bytes	4	3	-8	-0.13962634
Intensity	57	1 Byte				
Distance Information	61	2 Bytes	6	5	-6.66	-0.116238928
Intensity	63	1 Byte				
Distance Information	67	2 Bytes	8	7	-5.33	-0.093026049
Intensity	69	1 Byte				
Distance Information	73	2 Bytes	10	9	-4	-0.06981317
Intensity	75	1 Byte				
Distance Information	79	2 Bytes	12	11	-2.67	-0.046600291
Intensity	81	1 Byte				
Distance Information	85	2 Bytes	14	13	-1.33	-0.023212879
Intensity	87	1 Byte				
Distance Information	91	2 Bytes	16	15	0	0
Intensity	93	1 Byte				
Distance Information	97	2 Bytes	18	17	1.33	0.023212879
Intensity	99	1 Byte				
Distance Information	103	2 Bytes	20	19	2.67	0.046600291
Intensity	105	1 Byte				
Distance Information	109	2 Bytes	22	21	4	0.06981317
Intensity	111	1 Byte				
Distance Information	115	2 Bytes	24	23	5.33	0.093026049
Intensity	117	1 Byte				
Distance Information	121	2 Bytes	26	25	6.67	0.116413461
Intensity	123	1 Byte				
Distance Information	127	2 Bytes	28	27	8	0.13962634
Intensity	129	1 Byte				

Distance Information	133	2 Bytes	30	29	9.33	0.162839219
Intensity	135	1 Byte				
Distance Information	139	2 Bytes	32	31	10.67	0.186226631
Intensity	141	1 Byte				

VITA AUCTORIS

NAME:	Bradford Scott Bondy
Place of Birth:	Windsor, Ontario, Canada
Year of Birth:	September 2nd, 1980
Education:	Sandwich Secondary School, LaSalle, Ontario, Canada 1994-1998
	St. Clair College, Windsor, Ontario, Canada 2008-2012
	Lakehead University, Thunder Bay, Ontario, Canada 2012-2014 B.Eng
	University of Windsor, Windsor, Ontario, Canada 2016-2020 MASc