

SYSC4907: Group 11 Project Report

Data Visualization and Curb Detection for Autonomous Vehicles

Group members:

Danilo Vucetic 100999548

Michael Pruss 101008219

Michael Skalecki 100969837

Philip Naida 101012663

Keyan Cassis 101011524

Supervised by Professor Gohary

Cosupervised by Professor Lambadaris

Abstract

Curb detection is a basic task in the field of autonomous driving where 3D LiDAR is commonly used both independently and in corroboration with other sensors or cameras. This paper presents a curb detection method for use on LiDAR point clouds which follows 5 main steps: (1) LiDAR data is passed through a filter to remove points which are part of the road or objects such as trees. (2) Similar points are clustered together so that each cluster represents a different entity in the point cloud. (3) Point clusters are denoised using principal component analysis to further reduce noise which could cause false positives. (4) The point cloud is split in half to separate the left and right curb and each side is segmented into small linear segments which allows both linear and curved curbs to be represented. These segments are grouped with other similar segments which likely belong to the same entity. (5) Confidence values are assigned to each segment cluster based on their curb-like characteristics. In an effort to reduce processing time of real-time data, successive LiDAR snapshots are filtered around the likely location of the curb which is determined from the previous snapshot's detection. A road detection method is also presented which determines the curb location based on the edges of the detected drivable road. The difference in radii of adjacent LiDAR lasers are examined to find inflection points which represent the road's edges, and thus contain the drivable road between them. A corroboration method is proposed to combine the results of the curb detection method and road detection method in an effort to increase the confidence of the detection. Results show that curbs are detected consistently with little occurrence of false positives.

Acknowledgements

We would like to acknowledge all the people who helped us throughout the year. Firstly, we would like to thank Caleb Turcotte who familiarized the group with a QNX platform which was used to extract data and test our methods. We would like to acknowledge the Systems and Computer Engineering department at Carleton University for providing the resources and venue to facilitate the research and development of our project. We would like to acknowledge Dr. Gohary and Dr. Lambadaris for supervising our project. We would like to thank Dr. Gohary for meeting with us almost every week for the entire school year to discuss ideas and our progress. Without Dr. Gohary's guidance and expertise, this project would never have accomplished the goals we set. Lastly we would like to thank QNX for providing a problem, the data, and some expertise for the project.

Table of Contents

Chapter 1: Introduction	1
1.1 Problem Motivation	1
1.2 Problem Background	2
1.3 Problem Statement	3
Chapter 2: The Engineering Project	4
2.1 Health and Safety	4
2.2 Engineering Professionalism	4
2.3 Project Management	5
2.4 Justification of Suitability for Degree Program	5
2.5 Individual Contributions	6
Chapter 3: Background and Related Work	10
3.1 Structure and Characteristics of LiDAR Data	12
3.2 Properties of Curbs and the Road in LiDAR Point Clouds	14
3.3 Data Structures	15
Chapter 4: Transforms and Filters	16
4.1 Hough Transforms	16
4.2 Fourier Transforms for Point Density-Based Feature Extraction	19
4.3 Road Surface Filter	21
Chapter 5: Sliding Window Approaches	23
5.1 Height Variation	23
5.2 Mean Squared Error-Based Curb Detection	25
5.3 Principal Component Analysis-Based Normal Vector Extraction	27
Chapter 6: Curb Detection Method	30
6.1 Preprocessing	31
6.2 Filtering	31
6.2.1 Similarity Filter	32
6.2.2 Augmented Similarity Filter	34
6.3 Clustering	35
6.4 Denoise	39
6.5 Segment and Linear Fit	41
6.6 Segment Grouping	43
6.7 Assigning Confidence to Detected Curbs	46
6.7.1 Intra-Cluster Confidence	46
6.7.2 Inter-Cluster Confidence	47

6.8 Real-Time Acceleration Methods	48
6.8.1 Segment-Based Point Cloud Reduction	48
6.8.2 Rear-Environment Recall Point Cloud Reduction	51
6.8.3 Speedup Results of Real-Time Acceleration	54
Chapter 7: Road Detection and Corroboration of Curbs	56
7.1 Road Detection	56
7.1.1 Surface Variation Approach	57
7.1.2 Averaged Difference of Radii Approach	58
7.2 Results on Road Detection	59
Chapter 8: Combined Methods and Results	61
8.1 Assigning Confidence to Corroborated Detection	61
8.2 Results with HDL64E	62
8.3 Results on Sparse Point Clouds	66
Chapter 9: Future Work and Research Possibilities	69
9.1 Real-Time Speedup	69
9.1.1 Multithreading Speedup	70
9.1.2 Hardware Acceleration	71
9.2 Improvements to Filtering and Clustering Steps	72
9.3 Machine Learning for Sliding-Window Filters	72
9.4 Sensor Fusion	73
Conclusion	75
References	77

List of Figures

Chapter 3: Background and Related Work	10
Figure 3.1: LiDAR Coordinate System	13
Figure 3.2: Unfiltered LiDAR Snapshot	13
Figure 3.3: Example Curb Model	14
Chapter 4: Transforms and Filters	16
Figure 4.1: Top view of 3D LiDAR Point Cloud	17
Figure 4.2: Cartesian Space to Hough Parameter Space	17
Figure 4.3: Top view with potential and classified curb lines	18
Figure 4.4: Rising/Falling Step Functions and their Fourier Transforms	20
Figure 4.4: LiDAR Point Cloud before and after Road Surface Filter	22
Chapter 5: Sliding Window Approaches	23
Figure 5.1: Points within a bounding box with y-axis collapsed	24
Figure 5.2: Ideal representation of curbs	25
Figure 5.3: Ideal data results	26
Figure 5.4: MSE method results on real point cloud data	27
Figure 5.5: Principle Component Vectors	28
Figure 5.6: Results from PCA on point cloud	29
Chapter 6: Curb Detection Method	30
Figure 6.1: Cosine Similarity Visualization	33
Figure 6.2: LiDAR point cloud before and after the Similarity Filter	34
Figure 6.3: LiDAR point cloud before and after the Augmented Similarity Filter	35
Figure 6.4: Distance-based K-Means ($k=2$) vs density-based DBSCAN	36
Figure 6.5: DBSCAN Clustering	37
Figure 6.6: DBSCAN core, border, and noise point identification	38
Figure 6.7: LiDAR point cloud before (left) and after (right) DBSCAN clustering	38
Figure 6.8: Point cloud before de-noising and after	41
Figure 6.9: Curb detected in point cloud	42
Figure 6.10: Effects of changing intercept (left) and slope (right) for similarity metric	44
Figure 6.11: Effects of changing intercept and slope for similarity metric in 3D plot	45
Figure 6.12: Curb Segment Grouping	45
Figure 6.13: Exponential averaging example with bus lane over multiple snapshots	49
Figure 6.14: Graphs of the exponentially averaged boundary and the real curb location	51
Figure 6.15: Frame to frame vehicle translation	52
Figure 6.16: Before and after rotation of the vehicle	53

Chapter 7: Road Detection and Corroboration of Curbs	56
Figure 7.1: Adjacent Laser Radii and their difference	56
Figure 7.2: Road surface detection using inflection points	60
Chapter 8: Combined Methods and Results	61
Figure 8.1: Raw point cloud (left) vs. detected curbs (right)	62
Figure 8.2: Cluster size versus removal percentage for sparsity testing	67
Figure 8.3: Similarity of largest cluster in detection to known curb versus removal percentage for sparsity testing	67
Figure 8.4: Average Similarity of all detected clusters in detection to known curb versus removal percentage for sparsity testing	68

List of Tables

Chapter 2: The Engineering Project	4
Table 2.1: Project and Final Report Individual Contributions	6
Chapter 6: Curb Detection Method	30
Table 6.1: Real-Time Detection Acceleration Results	54
Chapter 8: Combined Methods and Results	61
Table 8.1: Curb detection method results using HDL64E LiDAR sensor	63

Part 1

Introduction and Project Management

Chapter 1: Introduction

Autonomous Vehicles are increasingly becoming a hot topic among tech-enthusiasts and the public alike. Companies such as Tesla, Uber, and Google are involved in shaping a new market of cars that drive themselves. These companies often herald the benefits of autonomous driving and the tools that have enabled their achievements so far. Some of the most important advances to date involve driver assistance systems and the suite of sensors used to gain an understanding of the car's environment. These sensors allow a computer to analyze the road, plan paths, and identify objects such as cars, bikes, or pedestrians. There are multiple approaches to the analysis of the road; Tesla and Google, for example, use machine learning extensively to analyze the driving environment, detect objects, and plan paths [1]. Other companies, like QNX, prefer mathematical and algorithmic approaches. Nonetheless, the topic of autonomous driving often brings up interesting questions around vehicle safety, accountability for road accidents, and energy efficiency¹. That said, this project is a collaboration with QNX to develop novel methods of reliable, real-time curb detection for autonomous vehicles. The remainder of this report proceeds as follows: First, the problem is identified in the Problem Motivation and Background sections. Then, the problem being investigated is explicitly stated and a solution is proposed. Next, the management of the project is discussed, with sections describing the safety precautions taken, the individual contributions, and engineering professionalism. Then, the Design section elaborates the methods used to accomplish the proposed solution. The Design section includes a detailed background which elaborates related work and terminology, discussion of alternative design solutions, and finally, a discussion of the achieved solution.

1.1 Problem Motivation

Driving a vehicle is an inherently dangerous task. This is evidenced by collision statistics showing that 114,158 injury-causing collisions occurred in Canada in 2017, resulting in 154,886 injuries [2]. Consequently, driving is regulated by governments to ensure a minimum standard of knowledge and competence among drivers (i.e. getting a license). The Ontario Ministry of Transportation, for example, regulates driving and defines guidelines for safe driving, stating that drivers must “know the traffic laws and driving practices that help traffic move safely” and that “breaking these ‘rules of the road’ is the major cause of collisions” [3]. Furthermore, drivers are imbued with responsibilities such as defensive driving, and being predictable and courteous to other drivers and road users [3]. These guidelines are often followed by drivers but accidents still occur and they may cause injury and death. It is worth noting that as many as 80% of collisions are at least partly caused by driver distraction or inattention [4]. It is unsurprising that the Canadian Automobile Association estimates the “economic and social consequences of road crashes in Canada ... to be \$25 billion per year” [4].

¹ The trolley problem for example.

Distracted driving is just one factor in a multitude that contributes to road accidents. The effects of distracted driving can however be severe. For example, the RCMP states that distracted driving can lead to “reduced reaction time, impaired judgement, falling asleep behind the wheel, injuring or killing yourself and/or your passengers and others” [5]. There are a few obvious remedies to the problem of driver distraction and inattention, namely, increasing fines as was done in Ontario in 2019, or passing new laws [6]. However, engineers have a duty to the public to design safe systems, thus, an effective engineering solution would include systems to reduce incidences of collisions and injuries thereafter related.

The motivation of this project is to produce a system with improved awareness of its environment in order to assist the driver and any automated systems in completing the driving task safely. The outcome of this project allows for increased safety of all road users by driver assistance systems, allowing the on-board computers to be paying attention even when the driver may not be.

1.2 Problem Background

Advanced Driver Assistance Systems (ADAS) are features in vehicles that aim to avoid and/or lessen the impact of collisions [7]. ADAS technologies are meant to be supplementary systems to the driver, i.e., the driver must still be fully alert while driving since these systems can not prevent accidents, only mitigate their risk. ADAS systems are back ups; the last resort for when the driver is unable to make the correct decision to maintain safety. They are meant to prevent or reduce the likelihood of injury and death. ADAS systems are already being used for blind spot detection, collision avoidance, adaptive cruise control, lane departure warnings, and lane keeping to name a few. These systems are implemented to aid drivers in the driving task by supporting their requisite actions such as checking blind spots, keeping a safe distance between themselves and the car ahead, staying in their lane, etc. Essentially, the point of ADAS systems as they currently stand is to help human drivers make better decisions, and increase safety through hazard detection and mitigation [7].

ADAS systems require sensors to accomplish their tasks. These sensors differ for each application, but in essence, the sensors are used to gain a better understanding of the environment around the vehicle. For instance, automatic emergency braking, adaptive cruise control, and blind spot detection use an assortment of RADAR, cameras, LiDAR, and ultrasonic sensors to accomplish their detection tasks [8-10]. The sensor of interest for this project is LiDAR (Light Detection and Ranging). LiDAR uses lasers to detect the distance between the sensor and the surrounding environment by calculating the time-of-flight of a laser pulse [11]. This project uses the HDL64E LiDAR manufactured by Velodyne. The HDL64 is a 3D LiDAR sensor, meaning it generates a view around the entire vehicle (360° horizontal field-of-view) for multiple vertically-stacked lasers (27° vertical field-of-view) [12].

1.3 Problem Statement

The goal of this project is to build a framework whose functions are to: 1) visualize three dimensional LiDAR data; and 2) detect and identify road curbs in said data. The framework visualizes the detected curbs in a three dimensional map to form a view of the vehicle's environment. The framework reports the detected curbs to an ADAS program where data fusion and route planning algorithms (e.g. those designed by QNX) use the detected curbs to gain a better understanding of the car's environment. The framework is designed to operate within a real-time deadline to allow for reliable, safety-critical, decision making. A completed system (i.e. the framework integrated with an ADAS platform) should aid the driver and car computer systems in understanding the road environment and it should allow for more complete ADAS functionality such as lane-keeping and path planning, with the goal of decreasing the number of driving accidents.

Chapter 2: The Engineering Project

The fourth year project is a culmination of all of the work that a student has performed throughout their degree. This chapter will discuss the project's adherence to health and safety, usage of engineering professionalism, how the project was managed, relation to each member's degree program, and each member's contributions.

2.1 Health and Safety

Automotive systems such as ADAS directly impact the safety of the occupants of a vehicle. Given the aforementioned ADAS functions and the role of an assistance system in the prevention and mitigation of road accidents, it is necessary that these systems are designed with safety and reliability in mind. In the case of curb detection, an ADAS system will continuously monitor the vehicle's environment, tracking the curbs, and identifying the limits of the road. Redundancy and corroboration of detection is a necessity in this, and many other safety-critical tasks. The curb detection algorithm presented in this project has been designed to integrate two methods of curb detection for redundancy and corroboration. In this way, an environmental obstacle can be validated as a curb. This makes certain that a curb is not missed, nor incorrectly detected.

The project is entirely software based so the majority of health and safety concerns were not applicable to this project. The project was developed in a safe environment which respected each team member's health and safety. A room was assigned to the group where members could work together while seated in ergonomic chairs and developing software on personal laptops or the provided desktops. There were no hazards present in the room that could negatively affect a member's health. One issue that concerned the room was the lack of windows and openings for fresh air. This issue was overcome by leaving the door ajar in order to allow fresh air. The team members were accommodated with a fridge as well as ample space to ensure that the room was easy to exit in case of emergency. Many breaks were taken during development and members had the option of working at home on their personal devices. Breaks were recommended in order to avoid significant eyestrain and muscle fatigue. The amount of time spent on development daily was restricted in order to ensure that fatigue is limited and consistent performance be maintained. Each member had a choice of chair they wished to use and an appropriate height to work at for ergonomic purposes.

2.2 Engineering Professionalism

This project, through the development of an ADAS-accompanying system, concerns public safety. Professionalism is a necessity to ensure a safe and reliable system for use by the public. Engineering practitioners have a legal obligation to ensure the welfare of the public. This project was designed with the intention of detecting curbs along the road. During development of this

system, focus was placed on reliability and safety through the practice of testing. The system was tested with several sets of data on different roadways to ensure effective detection across multiple scenarios. The team worked diligently to avoid mistakes and to develop a reliable system.

The system consists of original software which involves the use of state-of-the-art methods for curb and obstacle detection. These methods were researched and analyzed to determine the best application for curb detection with LiDAR. For ethical and professional purposes, all persons who deserve credit have been referenced and any work that has been used in this project is included as reference. This project displays the notion of public welfare and safety being paramount since it strives to reduce road accidents, and increase overall road safety through environmental perception. Professional ethics are required due to the safety concerns present in an ADAS system since driver and pedestrian safety may be impacted.

2.3 Project Management

The project was managed through scheduled weekly meetings where progress, complications, and future plans were discussed. These meetings were required for all members of the project and Dr. Gohary attended each of them. Additional meetings between the entire group were also arranged when necessary. Work was divided amongst the group in order to ensure that each individual had tasks to work on. The project group was divided into subgroups in order to focus on parts of the project which would later be integrated into the final product. The primary focus of each subgroup was to develop their own implementation of the curb detection algorithm. Both of these methods were later integrated in order to corroborate one another. Subgroups held meetings during the week when necessary and all members of the project communicated through Slack. Dates and deadlines were established for tasks.

The implementation of each method followed an incremental development process. The implementation began with unreliable detection which functioned under limited circumstances and assumptions. Incrementally, the method was improved upon and limitations and assumptions were progressively removed.

This process of forming subgroups and dividing work along with holding weekly meetings provided valuable experience. Projects in industry will involve working in teams and will necessitate coordination amongst team members to ensure an efficient development process. The action of working together in order to build the final system will prove to be invaluable experience.

2.4 Justification of Suitability for Degree Program

Each member of this project is enrolled in Computer Systems Engineering. The project provided valuable experience in the development of software in the context of a real-time system. The project performed data analysis on point clouds from the Velodyne HDL64E 3D LiDAR. Data

analysis is an important aspect of engineering and the development of curb detection models provided valuable experience in data science.

Initially, the point cloud data was extracted from a QNX visualization environment in order to develop the curb detection algorithm externally in Python. Using Python, each member contributed to the development of a curb detection algorithm. The process of developing this algorithm exposed each member to the software development cycle of requirements elicitation, design, implementation, and validation/testing. Carleton has provided the group with classes that enabled each student to develop robust and logical software needed to complete this project. SYSC 3303 and SYSC 3010 taught students how to work together in groups in order to develop a software system. These classes provided valuable experience, and taught how to divide work and integrate individual contributions. SYSC 3020 and SYSC 4106 provided experience with systematically developing software and the proper lifecycle of software. After completion in Python, the software was ported to C where the curb detection algorithm was run on a simulation of the vehicle's real-time computer system. From experience with embedded systems and real-time courses, the team strived to optimize the data analysis algorithm to execute within the vehicle's hard real-time deadline. This project exposed each member to a variety of software concepts and greatly helped members develop coherent software. Members acquired invaluable literature skills through the research of contemporary methods of curb detection.

2.5 Individual Contributions

Each member performed their own research and application of said research. The report was written and reviewed equally and each member's specific contributions are listed below in **Table 2.1**.

Table 2.1: Project and Final Report Individual Contributions

Member	Project Contribution	Report Contribution
Danilo Vucetic	<ul style="list-style-type: none"> Contributed to the curb detection algorithm through research of state-of-the-art methods, development of these and new methods, testing, and modelling. Contributed to the similarity filter. Developed the similarity metric. Contributed to segment grouping technique. Developed the segment-based point reduction technique and modelled its behaviour. Developed early detection techniques including sliding windows, linear curb detection, and idealized curbs. Labelled point clouds for testing. 	1, 1.1, 1.2, 1.3, 3, 3.3, 5, 5.1, 6, 6.2.1, 6.8.1, 8.3, 9, 9.1, 9.3, 9.4, Conclusion

Michael Pruss	<ul style="list-style-type: none"> Contributed to the curb detection algorithm through research of state-of-the-art methods, development of these and new methods, testing, and modelling. Contributed to the similarity filter. Developed PCA and normal vector extraction through research and applied it to various steps in the curb detection method. Developed the segment grouping technique. Developed confidence metrics. Developed early detection techniques including sliding windows, linear curb detection, and idealized curbs. Tested numerous clustering techniques for use in the final method. 	5.2, 5.3, 6.2.2, 6.4, 6.5, 6.6, 6.7, 6.7.1, 6.7.2, 8, 8.1, 8.2, 9.2
Michael Skalecki	<ul style="list-style-type: none"> Developed the Hough transform method through research of several articles for detection of linear curbs. <ul style="list-style-type: none"> Application to curb detection is viable but was not used in the final method. Aided in road surface detection and developed a second method for detecting relevant inflection points that lie on the road's curbs. Labelled point cloud data for various scenarios to be used to validate detection. Created a video to demonstrate the project. 	2.1, 2.2, 2.3, 2.4, 4, 4.1, 7.1.2, 7.2
Philip Naida	<ul style="list-style-type: none"> Developed the road detection method through research of state-of-the-art methods. <ul style="list-style-type: none"> Developed the surface variation method of inflection point detection. Developed and Researched methods for curb detection and noise filtering using Fourier transforms. Developed the road surface filter through research of point cloud edge detection using the surface variation metric and tested its use in curb detection. Developed novel edge detection methods for point clouds using the surface variation metric. Contributed to the combined detection method. 	Abstract, 3.1, 3.2, 4.2, 4.3, 7.1, 7.1.1
Keyan Cassis	<ul style="list-style-type: none"> Contributed to point cloud preprocessing. Contributed to the similarity filter. Implemented most of the final method in C in the QNX environment for real-time testing and demo. 	Table of Contents, List of Figures, List of Tables, 6.1, 6.2, 6.2.1, 6.3, 6.8, 6.8.2, 6.8.3

	<ul style="list-style-type: none"> • Investigated clustering techniques as a method of denoising and curb detection. • Developed the rear-environment recall method of real-time speedup. • Investigated curb line equation extraction using RANSAC 	
--	--	--

Part 2

Account of Engineering Solutions

Chapter 3: Background and Related Work

Autonomous vehicles and ADAS require some level of environmental perception if they are to aid in, or fully complete, the driving task. Depending on the level of automation, different levels of environmental perception are required. The accepted scale of automation levels for autonomous vehicles is presented by the Society of Automotive Engineers (SAE) as a six-level scale from 0 to 5, where 0 indicates no automation, and 5 indicates fully autonomous driving. The ADAS solutions presented in **Section 1.2** are never surpassing an SAE Level 1 automation, meaning that they never aid the driver past assistance with acceleration or steering [13]. From the point of view of an ADAS, this should be self-evident since the system necessarily complements the driving task. It was previously stated that these Level 1 systems generally use RADAR, LiDAR, cameras, and ultrasonic sensors to perceive their environments. These sensors are usually used in the context of detecting other vehicles and, for example, matching their speed, or using cameras to detect and stay within a driving lane [7-10]. Higher levels of automation require a more complete understanding of the environment around the vehicle. If, for example, a car were to reach Level 3 automation, where the car itself handles the dynamic driving task but still requires a human driver, the car would need to perceive the driveable road, the edges of the drivable space, obstacles, and so on [13]. The detection of curbs and the road form a useful analog to the detection of the edge of drivable space, and thus provides a starting point to the advancement of autonomous driving. Various approaches have been studied to this end and are introduced below.

Camera-based methods of curb and road detection have been studied extensively. In [14], an elevation map is constructed from stereo camera data and edge detection is applied to extract height variations in the image. The Hough transform is used to extract curbs from a set of persistent curb points based on height variations in the elevation map. Curved curbs are detected using a chain of straight segments. Other camera-based methods look at road detection and the use of machine learning algorithms like Convolutional Neural Networks (CNN). A popular dataset for the development and comparison of road detection methods is KITTI [15, 16]. This dataset provides data for video cameras, LiDAR, and localization equipment with thousands of labelled data points. Road detection benchmarks on KITTI aim to detect the road based on either the use of cameras, LiDAR, or a combination of both. [17] used the camera data from the KITTI dataset and applied a CNN with gated recurrent units for road segmentation². An embedding of the image was created using convolutional layers, which was then fed to gated recurrent units and decoded to produce the segmentation. This method's effectiveness suffered from the sole use of cameras as the detection medium. Cameras are subject to issues around lighting conditions (shadows, confusing colours) and blur which cause false positives and false negatives. Road and curb detection should therefore be completed with, or supplemented with, other sensors which do not suffer from the same sensitivities.

² Segmentation refers to the process of assigning a label (curb, pedestrian, road, camel, etc.) to each element of an input, be it a pixel in an image or point in a point cloud.

Many of the highest-accuracy benchmarks on KITTI for road detection make use of both cameras and LiDAR through various sensor fusion techniques [18-21]. In [18], a Progressive LiDAR Adaptation for Road Detection (PLARD) approach is introduced to map LiDAR data to images with the goal of better detecting the road. This is accomplished through the adaptation of LiDAR data to the visual data space by aligning the perspectives of each sensor and by adapting visual features to features in the LiDAR data. PLARD uses a cascaded fusion architecture with Deep CNNs (DCNN) to adapt and fuse the data. The final stage of the neural network is a parsing layer which classifies road features. PLARD is exceptionally effective at road detection and ranks among the top benchmarks on KITTI. The authors state that “LiDAR is helpful for boosting robustness” and that “LiDAR is robust to visual noises and can complement monocular image data” [18]. Thus, fusion-based road detection techniques seem to be quite promising. That said, other LiDAR-based techniques perform remarkably well in road detection without the use of sensor fusion. In [22], a method for road detection using LiDAR is presented which examines adjacent LiDAR lasers to locate the edges of the road and classify all areas in between as the road surface. A major advantage of this approach is that it allows the edges of the road to be detected in both structured roads (with curbs) as well as unstructured roads (without curbs). This approach, however, fails to delineate objects, such as cars, from the road edge. Thus, for curb detection specifically, other approaches must be examined.

Curb detection remains a difficult task regardless of the detection medium. Camera-based methods are sensitive to environmental conditions (light, weather, etc.), noise, and blur, whereas LiDAR can detect distance without interference from light, but is sensitive to weather conditions (e.g., heavy snow and rain) and may become distorted in turns or at high speed [23, 24]. A common trend among most LiDAR-based curb detection methods is the reliance on filtering³. For example, [23, 25, 26] use some form of ground segmentation to differentiate on-ground points (road, curbs, cars, etc.) from off-ground points (trees, buildings, etc.). [23] and [25] use plane-based segmentation methods to delineate ground points and apply Random Sample Consensus (RANSAC) to better fit the plane to the on-ground points. [26] on the other hand uses a voxel grid to identify on-ground points. While the three methods fundamentally differ in how they achieve ground segmentation, they produce the same result: reducing the number of points in the point cloud while maintaining resolution on the curb. This idea was used in the development of the curb detection method presented in this report, and is essential to understanding how each of the three cited methods work.

In [23], distortion is removed from LiDAR data and ground points are segmented in a preprocessing step. Ground points are passed to an extraction step which uses multiple features of a curb (height, smoothness, tangent between neighbouring points, horizontal distance) to detect and extract candidate curb points. Then, the candidate points are clustered using a variant of DPCA and DBSCAN to classify left and right candidate points under different

³ Filtering is considered here not in the traditional sense of applying a transfer function to remove certain frequencies or elements of a signal as such; but instead as the application of an algorithm to remove unwanted points from a point cloud.

road curvature conditions. Next, a filtering step is applied to the curb points which removes obstacles inside and outside the road. Finally, a curb line is fit to the left and right candidate points. [25] uses a similar candidate curb extraction technique, but with an added sliding-beam segmentation method prior to the curb extraction. [25] also omits a clustering step and a regression step which decreases the experimental precision and recall compared to [23]. This suggests that, for a robust curb detection technique, a method of clustering and regression to remove non-curb points and obstacles is necessary.

[23] and [25] presented some interesting methods, particularly, the extraction of a curb based on the tangent vector between neighbouring points. This method calculates an average direction vector for a set of points on either side of a candidate point⁴. If the angle between the vectors is low enough, as set by a threshold, the point can be considered a curb point. [26] similarly extracts curbs based on curb features but uses voxels to its advantage. Voxels maintain locality in a 3D environment, this is useful in curb detection as changes in height, gradient, and surface normal within each voxel can be used to detect a curb. Each voxel is analyzed such that the height and gradient don't exceed expected thresholds for curbs. The calculation of a surface normal is done through Principal Component Analysis (PCA) based on the eigenvalue decomposition of the covariance matrix of the voxel. Any points fitting all three of the characteristics are considered curb points. The methods cited above for curb detection mostly report curb points or least-squares regression models of the curb lines. While this is effective for continuous curbs, a regression model cannot adequately detect instantaneous differences in curb position such as those seen with bus stops, intersections, and parking lots. Reporting curb points on the other hand may misrepresent the tendency of the curb as no indication on the rate of change of its position is given. Thus, a new reporting mechanism shall be introduced.

3.1 Structure and Characteristics of LiDAR Data

Light Detection and Ranging (LiDAR) is being used by many companies such as General Motors and Baidu for environment perception [23]. LiDAR is a surveying method which uses lasers projected from a sensor to map the surrounding environment. Referring to **Figure 3.1**, the distance, p , that each laser travels before it hits an object can be calculated based on its time of flight or phase difference, giving position data of the object it hit. This time of flight value is simply the difference in time between the moment the laser was projected from the sensor and the moment the reflection is received again at the sensor [27]. This distance combined with the azimuth of the laser, θ , from when it captured the sample as well as the vertical angle of the laser, ϕ , allows mapping of each sample point to a 3D cartesian coordinate (x , y , z). The LiDAR sensor does this with reference to itself as the point of origin. LiDAR data is structured so that with reference to the LiDAR sensor, the x -axis represents the lateral distance to the left and right, the y -axis represents distance forward and backwards and the z -axis represents the vertical distance. It should be noted that the values for the y -axis coming from the LiDAR sensor

⁴ As in, the x and y components of all component points on that side are averaged to form a new direction vector with respect to the candidate point.

were multiplied by -1 prior to processing due to the ADAS platform having an inverted y-axis. Representations for p , θ and ϕ are shown in **Figure 3.1**.

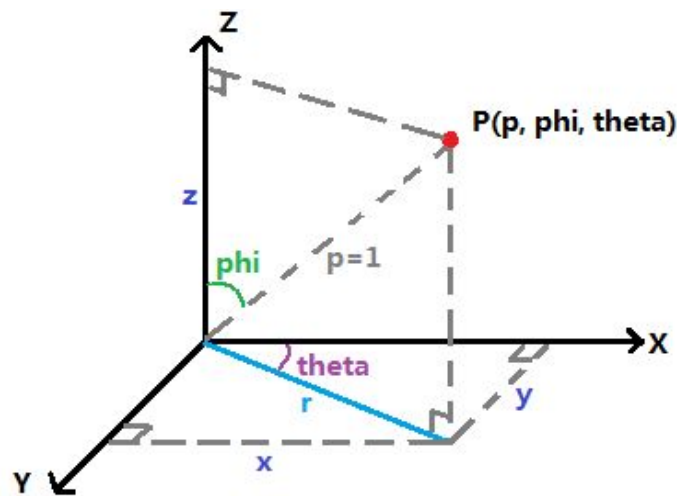


Figure 3.1: LiDAR Coordinate System [28]

The specific LiDAR sensor used for this project was the Velodyne HDL64E LiDAR sensor. The HDL64E has a fan of 64 lasers arranged vertically, each which are mounted at different incrementing angles with reference to the ground. This results in a vertical field of view of 27° [12]. This fan of lasers is then rotated about the vertical axis of the LiDAR sensor at a rate of 10Hz while each of the 64 lasers is sampled at a high frequency to produce 1821 sample points per rotation per laser. This produces 116,544 total data points per rotation which map the surrounding environment. Due to the fact that the 64 lasers are mounted to the LiDAR sensor at incrementing angles, the farther you move away from the sensor, the point density of LiDAR samples decreases. This has the effect of decreasing the resolution of the LiDAR point cloud far away from the vehicle causing detection algorithms using only LiDAR to suffer more as distance from the vehicle increases. This is shown in **Figure 3.2**.

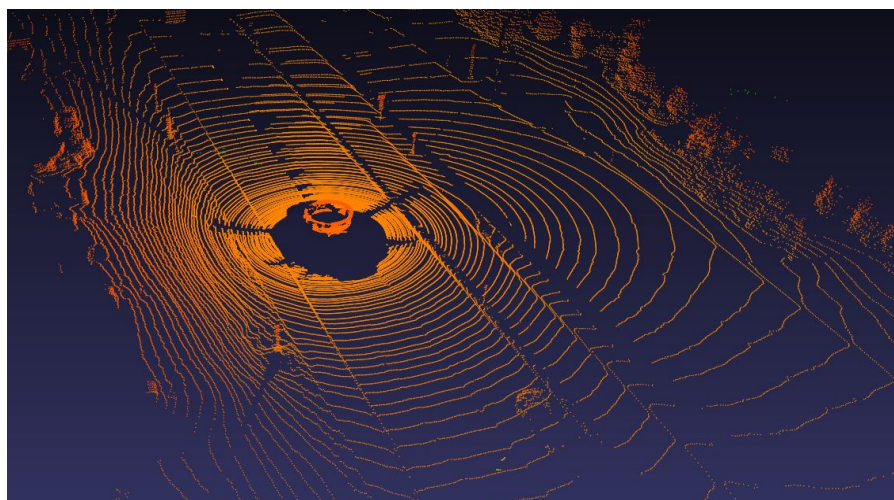


Figure 3.2: Unfiltered LiDAR Snapshot

Figure 3.2 depicts a full unfiltered LiDAR snapshot (i.e. data from one full rotation of the LiDAR sensor). This figure shows how the LiDAR point density decreases as you move further away from the vehicle. The result of this is that further away objects will have less data points on them than objects which are close to the sensor. Generally, the more data points on an object, the easier it will be to identify. This means that identifying objects which are far away will be harder than identifying objects which are close due to a lack of data.

3.2 Properties of Curbs and the Road in LiDAR Point Clouds

LiDAR point clouds have useful properties for detecting objects: each point is represented by a 3D cartesian coordinate (x , y , z) meaning the distance to an object or any sample point in the point cloud with reference to the sensor is known. The height and width of an object is also represented by a set of discrete points which lie upon it within the point cloud. In a scenario such as curb detection, the important properties for LiDAR include the number of points on the curb itself, the height of the curb, and the continuity of the curb. The number of points on the curb determines the amount of detail which can be extracted from the data. This number completely depends on where the curb is located with reference to the LiDAR sensor and the number and positioning of the lasers in the LiDAR sensor. As explained in **Section 3.1**, the LiDAR point density decreases the further away from the sensor you go meaning there will be less points on a curb that is far away than one which is close to the LiDAR sensor. This results in it being more difficult to identify objects such as curbs at far distances from the sensor.

The City of Toronto's official road engineering guidelines state that a road curb has a height of 150mm but can be as low as 100mm depending on drainage factors [29]. These variations are important to know for filtering steps that are presented in later sections as the tops of the curb could be removed if the data were to be truncated, for example, at some height level. Curbs can also be linear or curved. They can be continuous on long stretches of unpopulated road or have many discontinuities such as on a road with depressed sections of curb for driveways. It is important that the LiDAR sensor is able to capture the differences between these situations as they may present different actions to be taken on behalf of the driver.

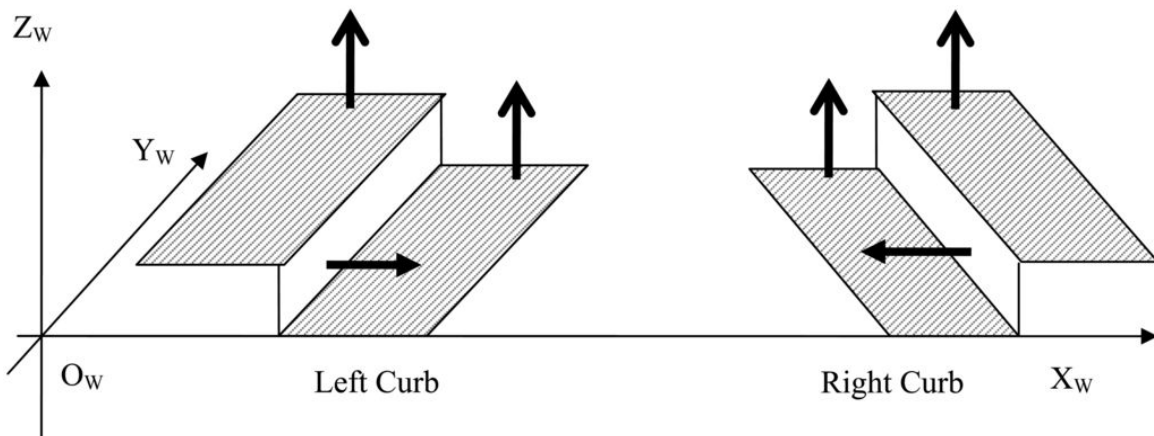


Figure 3.3: Example Curb Model [30]

Figure 3.3 shows an example of a linear continuous curb. The curb is present on both the left and right side of the road. Ideally, the LiDAR sensor will sample an adequate number of data points on this curb so that the full structure is recreated in the point cloud. Edges should remain sharp and flat surfaces should remain planar. The sampling frequency of the velodyne HDL64E sensor used for this research satisfies these requirements.

3.3 Data Structures

Point clouds generated by the Velodyne HDL64E contain around 116k points per rotation. Each point individually denotes the cartesian coordinate of the location where a laser reflected off of a surface. Henceforth, the vector $x = [x^{(1)}, x^{(2)}, x^{(3)}]$ shall represent a point in the point cloud, where each component of the vector represents the cartesian coordinates x , y , and z . Given the large size of these point clouds, a data structure was created to quickly store and access the points in an ordered manner. Due to the size of the point cloud and the data per point (laser ID, azimuth, elevation angle, distance), it was decided to use a matrix whose indices are: the laser ID for the rows, and the point number for the columns. The matrix form is a useful representation of the data insofar as it maintains the locality of the points in the point cloud. This allows for effective iteration through the points since adjacent points in a laser line are next to one another and points in adjacent laser lines have similar azimuths. The matrix is defined as such:

- Given a laser ID l , bounded by the number of lasers L , $l \in [1, L]$
- Given a point number p , bounded by the number of points P , $p \in [1, P]$
- Where each element of the matrix X is indexed as the vector x_{lp}

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1P} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2P} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{L1} & x_{L2} & x_{L3} & \dots & x_{LP} \end{bmatrix}$$

In later processing steps, locality between points in the same laser line is no longer required, so the matrix form of the point cloud becomes obsolete. These steps use a vector form of the pointcloud where the point cloud is represented as a vector of points, Y . The vector is defined below:

- Given a point number n , bounded by the number of remaining points N , $n \in [1, N]$
- Where each element of the vector Y is indexed by the vector x_n

$$Y = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

Chapter 4: Transforms and Filters

Transforms are useful in signal processing to analyze individual components of a signal. The individual components may provide insight into particular parts of the whole signal. Transforms, such as the Fourier Transform, and the Hough Transform, were experimented with to find patterns in the 3D point cloud data. Members have had exposure to various Fourier Transforms, which can decompose a signal into its constituent frequencies. The Hough Transform was introduced by Dr. Gohary and its properties of line detection within point clouds were explored. The Hough Transform is useful in detecting shapes within 2D and 3D point clouds, such as lines and circles [31]. These transforms were experimented with to see if their use can be applicable to curb detection.

4.1 Hough Transforms

The Hough Transforms are primarily used to detect lines within a 2D plane of points. This property can be applied to the detection of curbs in point cloud data. Hough Transforms are also capable of detecting circles and various shapes however this functionality was not used for this method [31].

An initial assumption of the project was to detect curbs of linear nature, such as curbs along a highway which do not curve significantly and have minimal discontinuities. This limitation allowed the detection to be easier by limiting the variables present in the data. The LiDAR point cloud data is originally provided in 3D but this is not necessary for detection of curb lines using Hough Transforms. The first step was to convert the 3D point cloud into a 2D plane which can be visualized from a bird's eye view as in **Figure 4.1**. This will remove the vertical z component and will cause an apparent clustering of points along the linear curbs where the laser tends to hit the side of the curbs. It is visually apparent that the curbs have a distinct linear shape on both sides of the road. The more points that lie along this line, the better the detection of the potential line will be. The Hough Transform is used to detect the lines within the grid.

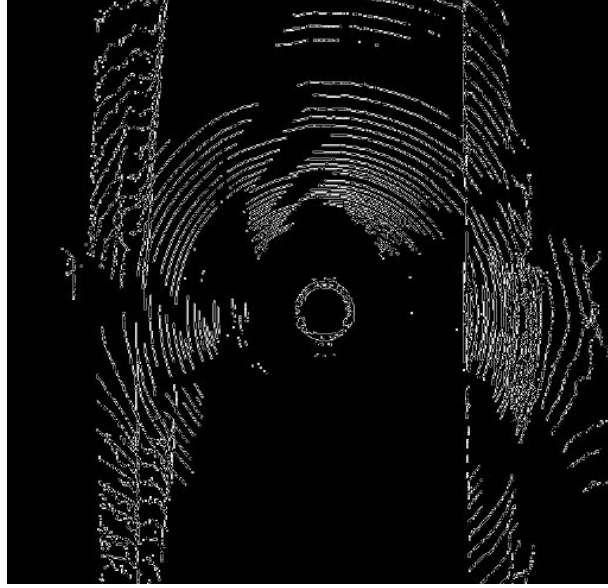


Figure 4.1: Top view of 3D LiDAR Point Cloud

The Hough Transform functions by detecting points within the plane that lie along a line. The Hough Transform operates similarly to a Fourier transform, where it transforms the cartesian space into a frequency space. The cartesian space consists of the familiar (x, y) points and the frequency space (known as the Hough parametric space) consists of (m, b) points corresponding to a linear equation as seen below.

$$y = mx + b \quad (1)$$

Each point in the cartesian space, as viewed in **Figure 4.1**, is joined to another point in the cartesian space. An (m, b) point corresponding to the line between two (x, y) points is created in the parametric space, shown in **Figure 4.2** below. If multiple (x, y) points lie along the same (m, b) line, then there will be a greater density of the same (m, b) point in the parametric space. A curb should be a straight line, so the parametric space will have a large density of particular (m, b) points. Referring to **Figure 4.2**, any point that would lie on the line equation is converted to a single (m, b) point in the parametric space. These (m, b) points will correspond to the linear segments found within the 2D plane [32]. If the number of (m, b) values is greater than a threshold, then this defines a pronounced line within the 2D plane.

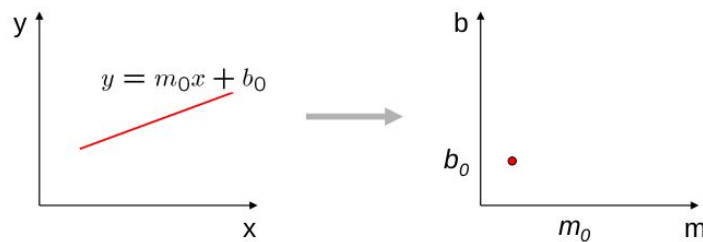


Figure 4.2: Cartesian Space to Hough Parameter Space [32]

Once a list of potential (m, b) points is determined, it is important to determine which points correspond to the curbs, rather than environmental obstacles or noise. Threshold values are defined for setting the minimal density of an (m, b) point. If this minimal density is reached, then it can be assumed that the (m, b) values form a well defined line. This line will then be further validated to determine how probable it is to be a curb. For example, a vehicle obstacle will yield a short line, so the probability of a longer line being a curb is greater. There may also be discontinuities in the detected line, due to LiDAR error and deviations in the intensity of the curb. The curb may be shorter in particular segments, or the LiDAR may have noise. Therefore a minimum distance between series of cartesian points lying on the same (m, b) line must be defined. This will ensure that the clusters of points in the cartesian space correspond to a curb, rather than several vehicles travelling in the same direction or something else with a similar orientation. If several vehicles are travelling in the same direction, then the edges of the vehicles may be detected and there would be a disjointed series of cartesian points which do not correspond to a curb. Therefore the minimum distance will prevent this (m, b) line from being classified as a curb [31].

Once potential curb lines are detected, verification techniques can then be used to increase the confidence of the line being a curb. On a straight road, curb lines are parallel, so if two (m, b) values are determined to be parallel (having the same m value) then there is a greater probability of these lines being curbs. As seen in **Figure 4.3** below, several potential curb lines were detected in red and the actual curbs were detected in green. The detected curbs were determined to be parallel and contained a greater number of points than the other potential curb lines. Therefore, there is a greater confidence that these lines correspond to the curbs. Once the curbs are detected, the (m, b) values will be used to define the location of the curbs

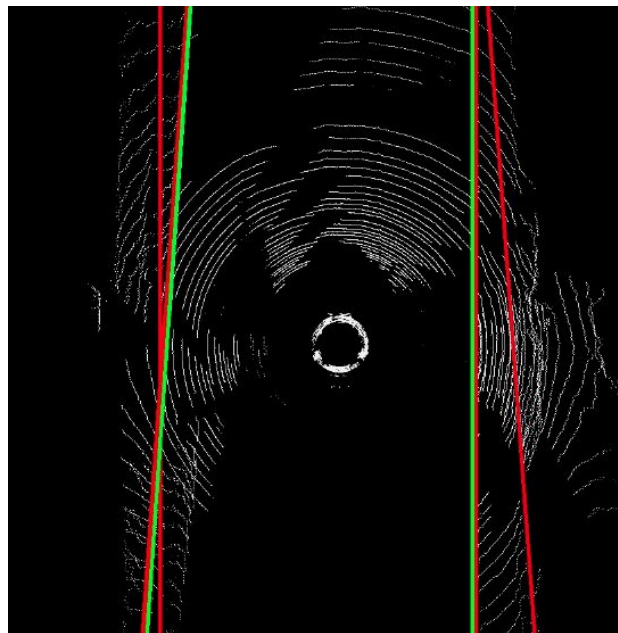


Figure 4.3: LiDAR top view with potential and classified curb lines

The Hough Transform approach was not used in the final method due to its restriction of being primarily effective when linear curbs are present. The Hough Transform can be extended to detect curved curbs but this poses complications as the LiDAR data itself is circular in nature. As is apparent from **Figure 4.3**, the LiDAR data consists of a large number of concentric circles. It is therefore extremely challenging to be able to distinguish a curved curb from the LiDAR points themselves. Therefore, the application to curved curbs may not be feasible using LiDAR. The Hough Transform also has the caveat of detecting potential lines that are not curbs as many linear shapes exist in the environment. It is susceptible to noise as points must lie along the (m, b) line with minimal noise to be deemed as being part of the line. Further research into the application of Hough Transforms should be done, and their application in curb detection is promising.

4.2 Fourier Transforms for Point Density-Based Feature Extraction

The Fourier transform is a tool used to convert a signal from the time or space domain into the frequency domain. In doing so, it decomposes the signal into all of the respective frequencies which comprise it. The structure of the signal in time/space domain is directly related to its structure in the frequency domain. If a signal is transformed into the frequency domain, an inverse Fourier transform can be used to transform the signal back to its original time/space domain structure. A signal transformed into the frequency domain allows identification of qualities such as its dominant frequencies and specific features found in the time/space domain [33].

3D LiDAR data from an automobile contains a lot of useless information when being used specifically for the purpose of an ADAS system. In the case of curb detection, the curbs themselves make up only a very small portion of each LiDAR snapshot. When on an open road, the majority of the data points are part of the road, trees/greenery or objects such as vehicles. This information is not necessarily required for the task of curb detection.

This method is an attempt to use the Fourier transform to decompose the LiDAR data into its frequency components and identify specific time/space domain characteristics (such as continuous curbs) from them. The Fourier transform is most used in 2 dimensions, and since the LiDAR data is 3D it will require dimensionality reduction to be processed effectively.

To effectively explain how the dimensionality reduction was done, the coordinate system for 3D LiDAR data must first be understood. As explained in **Section 3.2**, the direction looking out the front of the vehicle is the y-axis. Perpendicular to that on the horizontal plane is the x-axis. Finally the z-axis represents the vertical change. The particular 2D plane we are interested in reducing our 3D data to is the x-z plane. Ideally, the x-z plane of a road would show the curb on the left side of the road, then a step down representing the start of the road, and finally a step

back up to represent the curb on the other side of the road. These two steps are represented in the space domain within this data and will also have a distinct representation when transformed into the Frequency domain. The goal was to transform this spatial data into the frequency domain and confirm the presence of curbs through their frequency domain representation. An example of ascending and descending unit step functions and their Fourier transforms are shown in **Figure 4.4**.

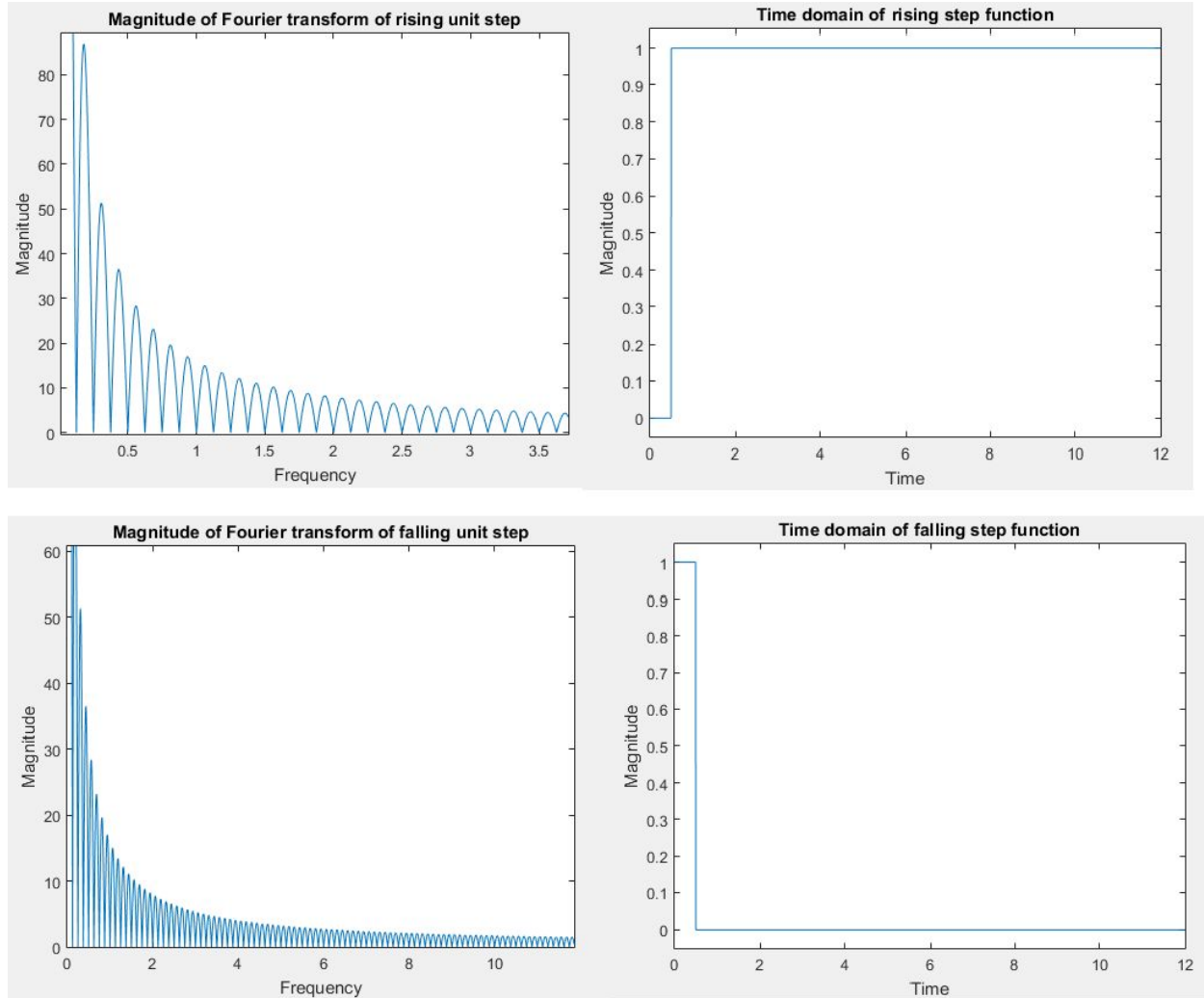


Figure 4.4: Rising/Falling Step Function and their Fourier Transforms

Dimensionality reduction on the 3D LiDAR data was done in 2 different ways. The first, was done by averaging all points across the y-axis to create an x-z plane. The second was done by using a sliding window average to examine a single slice of the y-axis at a time in effort to reduce noise created when averaging the entire y-axis into one x-z plane. With the first method, an entire 3D LiDAR snapshot is averaged across the y-axis, effectively removing the y dimension. This leaves an x-z plane which represents the average LiDAR structure over the y-axis.

By using the Fourier transform to represent this data in the frequency domain, it was expected to see a specific pattern that confirmed the presence of curbs. The result showed very little of what was expected. Because of the presence of noise and spatial variation in the data, neither the first or second methods of dimensionality reduction and transformation presented any patterns that confirmed the presence of curbs. This resulted in Fourier transforms not being used for our methods.

4.3 Road Surface Filter

LiDAR data from a road going vehicle is inherently going to contain large quantities of flat surfaces (i.e. road and ground). The curbs of a road can be thought of as the intersection of two planes (flat surfaces), that of the road and that of the ground beside the road. The ground beside the road is typically at a slightly higher elevation than the road which creates the curb when the two intersect. By using a 3D point cloud edge detection method, these intersections can be extracted from the rest of the data. Using the road surface filter, the objective is to remove the flat planes within the LiDAR data, leaving only the curb edges and objects. This result is shown in **Figure 4.5**.

A measurement called surface variation, allows determining if a group of cartesian points represent a plane or an edge (intersection of planes). This measurement can be used on groups of points within the LiDAR data to determine whether they are part of a flat surface or an edge and remove them if necessary. The surface variation equation divides the smallest eigenvalue of the 3D covariance matrix with the sum of all 3 eigenvalues. If the result is close to zero, then the group of points is representative of a flat plane. Otherwise, they represent an edge [34]. The 3 eigenvalues obey the following: $\lambda_0 < \lambda_1 < \lambda_2$.

$$\sigma_k(p) = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2} \quad (2)$$

To examine the entire point cloud for edges, the K-Nearest-Neighbours (kNN) algorithm was used. For each point in the point cloud, its K-Nearest-Neighbours are examined and the 3D covariance matrix is extracted [35]. From this covariance matrix, the 3 eigenvalues are computed and the surface variation of this group can be found using equation 2 above. If the surface variation is close to zero, it is classified as a flat surface and only the initial query point is removed, not its neighbours. If the surface variation is not close to zero, the point is not removed. This process results in a LiDAR point cloud which consists entirely of edges and no flat surfaces. By removing flat surfaces from the LiDAR data, other curb detection methods will have less points to process which will make them more efficient.

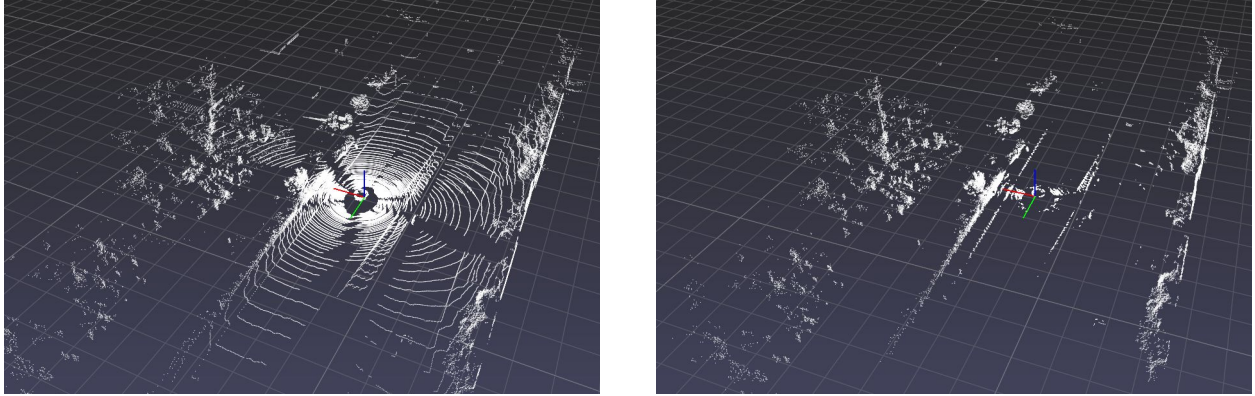


Figure 4.5: LiDAR Point Cloud before and after Road Surface Filter

The kNN algorithm is extremely useful for this method as it can be used to help assess the immediate neighbours of a point to determine whether it is part of a flat surface or an edge. The disadvantage of this method is the high complexity of the kNN algorithm. The kNN algorithm has a high complexity due to examining every point in the point cloud along with their k nearest neighbours. This is a detriment to the real time task of curb detection and therefore the Road Surface Filter was not used in the final detection method. If a more efficient algorithm for examining points and their spatial locality could be used, this method would be much more suited for curb detection.

Chapter 5: Sliding Window Approaches

Convolutional Neural Networks employ learned filters with the convolution operator to transform an input image to an output set of values, usually consisting in a ‘volume’ of outputs corresponding to the size of the input and the number of nodes in a layer [36]⁵. This is the root of the idea of a sliding window: an operation occurring in a sliding or moving fashion across a data space, encompassing multiple data points at once. Convolutional filters can be taught to detect edges in an image, to detect shapes, and more [36]. This was the motivation towards sliding windows as a curb detection technique. If a filter could be created to detect curbs, this would allow for a one-pass curb detection algorithm. This project opted not to use machine learning and CNNs, but to instead research methods of curb detection using the general idea of bounded inputs and sliding windows.

In [26], voxels were introduced as a bounding method. The bounding criteria for a voxel is the physical space taken by the bounding box. A voxel-based approach maintains the same volume per voxel, meaning that the number of points per voxel differ depending on the distance from the vehicle⁶. Voxels are also generally rectangular prisms, meaning that they do not effectively capture the spread of LiDAR points around a vehicle. The benefit of voxels, however, is that they provide a set space over which the LiDAR data can be analyzed. [26] used this to calculate the normal vector of the plane of points in the voxel and to generate other criteria such as height difference and gradient. To apply a sliding window, voxels wouldn't work. The sliding window approach requires the same number of points in the filter as in the selected data. It would be impractical to create a new filter for each voxel and the number of points contained within. So a new bounding box was conceived which would retain the same number of points while moving over different point densities. This was dubbed the polar bounding box. The methods presented in this chapter make use of polar bounding boxes in various ways. Each method iterates over the point cloud with a fixed-size bounding box, completing various calculations with the aim of detecting the curb directly.

5.1 Height Variation

Height variation uses a basic approach to detect curbs: in any given bounding box, iterate from the points nearest to the car, to the points furthest away, and calculate the height difference between points in adjacent laser lines. If any set of points exhibit a height difference within those levels considered for curbs (between 5 and 20 cm), then these points are considered curb points⁷. **Figure 5.1** graphs the points in a bounding box to the right of the vehicle, extending 12 meters to the right of the vehicle, a meter in the direction of travel centered to the vehicle, and

⁵ A filter is moved across the pixels in an image to create this output.

⁶ See **Section 3.1**.

⁷ This check was completed between multiple laser lines, not just directly adjacent lines.

with no vertical bound (i.e., $x \in [0, 12] \text{ m}$; $y \in [-0.5, 0.5] \text{ m}$). The points are projected to the x-z axis for ease of viewing.

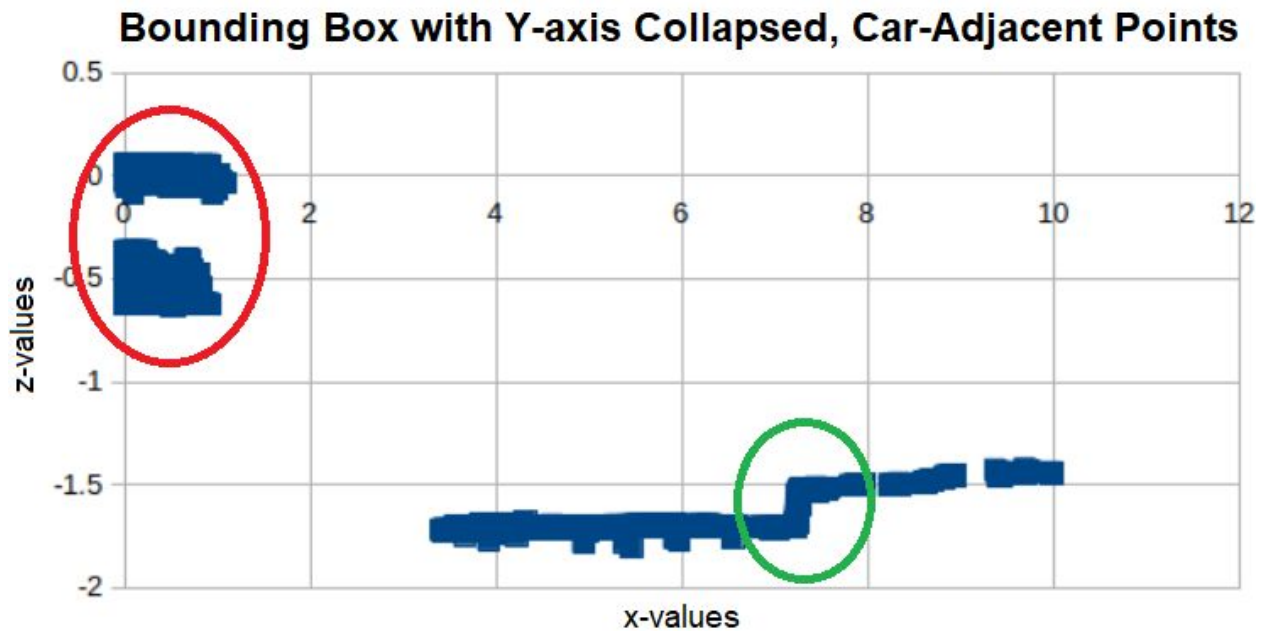


Figure 5.1: Points within a bounding box with y-axis collapsed

The red circle in **Figure 5.1** is noise from points hitting the top of the vehicle, these points can be ignored. The green circle, however, clearly illustrates the curb rising from a ground level of about -1.7 meters (relative to the LiDAR sensor height), to about -1.5 meters. The height variation method was developed with this example in mind. The idea being that every curb would share similar characteristics and would therefore be detectable by the height variation relative to the road. In practice, most curbs and road environments are not as perfect as the figure would suggest. The height variation method would, for example, falsely detect grass between the curb and the sidewalk as a curb, without so much as a single point detected on the true curb. Other situations like a sloping road or a sloping ground section outside of the road area would be detected as a curb entirely, or in addition to the true curb. Attempts were made to mitigate the errors, such as ensuring that a small distance on the ground plane actually corresponded with a curb-like height increase. This attempt was better able to detect true curbs rather than slopes, but failed to distinguish curbs from trees and grass. No other attempts were made to adjust the method.

This method showed promise in initial tests as it was capable of detecting nicely defined curbs in low-noise environments. In noisy environments, with non-ideal curb and road conditions, the method failed to adequately detect curbs. Some important lessons were learned from this method, namely, that the road environment is not perfect and that simple methods that just iterate over the bounding box, completing basic mathematics, would be ineffective. Thus, more complicated methods were examined, using ideal curbs, and a true sliding-window approach.

5.2 Mean Squared Error-Based Curb Detection

An initial approach to curb detection was the comparison of an idealized curb model to real point cloud data. The intuition was that curbs have a distinct shape that differentiate themselves from other objects in the point cloud. Comparing sections of the point cloud to an idealized curb model was a promising approach to detect curbs and remove other objects in the point cloud. To do this, an idealized curb model was created based on the structure of the point cloud data. The idealized model was converted into a matrix format based on the point cloud matrix representation presented in **Section 3.3**. The idealized model contains road space, sharp elevation change of 0.2m to represent the curb, and space behind the curb for 0.2m. **Figure 5.2** showcases a small segment from the idealized curb model. **Figure 5.2** shows the road in blue, the curb as a sharp elevation with varying change of colours, and the medium following the curb in red. This can be thought of as an intersection between three planes, first is the road in blue, then the curb, and finally the medium following the curb in red.

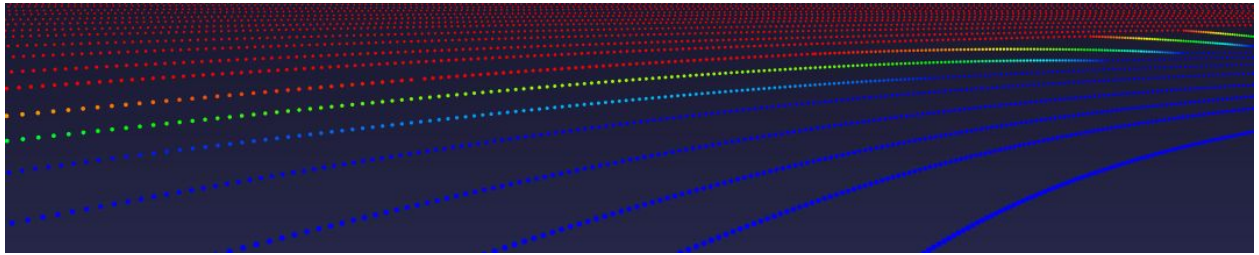


Figure 5.2: Ideal representation of curbs

For curb detection on real point cloud data, a small section that represents the idealized curb was extracted from the idealized model. This section is a sub-matrix of the idealized curb model matrix. The sub-matrix is defined as follows:

- Given a laser ID n , bounded by the number of lasers N , $n \in [1, N]$
- Given a point number m , bounded by the number of points M , $m \in [1, M]$
- Where each element of the matrix T is indexed as the vector t_{nm}

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} & \dots & t_{1m} \\ t_{21} & t_{22} & t_{23} & \dots & t_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_{n1} & t_{n2} & t_{n3} & \dots & t_{nm} \end{bmatrix}$$

The above matrix contains N laser lines with M points per laser. Matrix T was compared against sections of matrix X which was described in **Section 3.3** using a sliding window approach.

In order to detect curbs in real point cloud data using the idealized subsection, a metric that compared both the idealized subsection and the real point cloud data was required. The Mean Squared Error (MSE) takes the average sum of difference squared between each entry of the sub matrix and the real point cloud data matrix. The following formula is used to compute the MSE:

$$err = \frac{1}{nm} \sqrt{\sum_n \sum_m (t_{nm}^{(3)} - x_{nm}^{(3)})^2} \quad (3)$$

Where x_{nm} is a point in the X matrix that represents the full point cloud and t_{nm} is a point in the matrix T that represents the idealized curb section. Based on experiments, it was found that comparing only the z values of points resulted in better detection. The MSE metric gives a value that represents the similarity between the two matrices. A section of the real data that contains a curb is expected to have a high similarity to the idealized curb section. Therefore the lower the MSE value is, the more similar the matrices are and the more probable the current section is a curb. **Figure 5.3** below showcases the results of the MSE method on the ideal data.

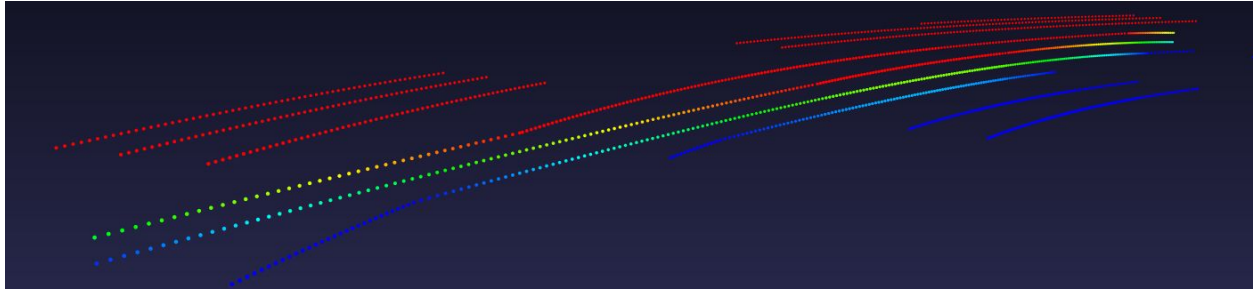


Figure 5.3: Ideal data results

As shown in **Figure 5.3**, this method shows promising results on idealized data, however there are critical limitations that reduce the effectiveness of this method on real data. First off, only a single side of the curb can be detected using one ideal curb model. The other curb side would need a separate curb model and would require two passes through the point cloud to detect both sides of the curb. Second, the density of points in the point cloud varies and decreases the further the points are from the vehicle. The idealized curb section does not take this into account and assumes constant point densities in the point cloud. Third, the point cloud contains noise that might distort the shape of the curb. Fourth, there are several different types of curbs that vary in height, therefore a single curb model will not capture the similarities of all types of curbs. **Figure 5.4** below shows the detection output using the MSE method for the snapshot.

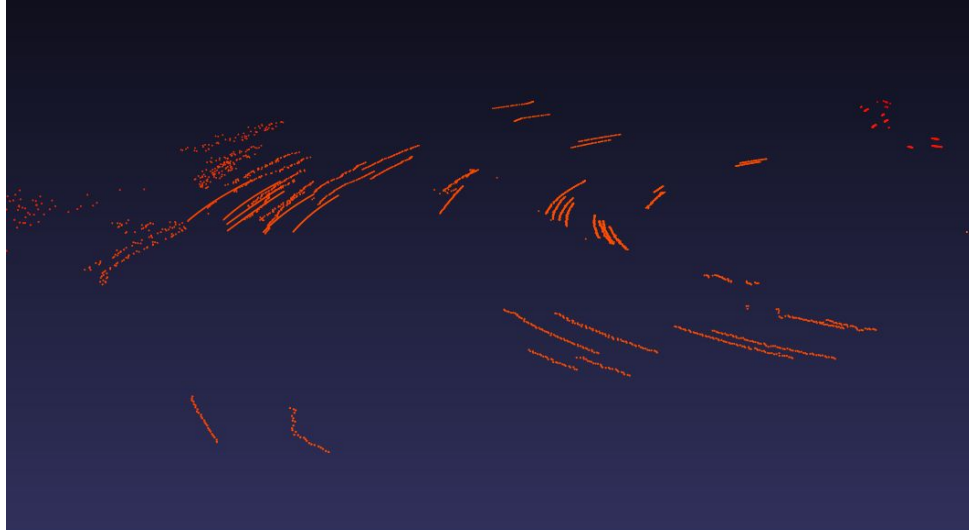


Figure 5.4: MSE method results on real point cloud data

As shown in **Figure 5.4**, it was found that this approach is not effective at detecting curbs and is prone to many false positives on real point cloud data. Based on the limitations discussed previously, it is understandable that this method has poor detection of curbs. This experiment also showcased that many other objects in the point cloud have relatively similar characteristics which makes a similarity method between ideal data and real data difficult. Due to this, the method produced many false positives in the real data with very few correct detection of curbs. For the above reasons, the MSE approach was not included in the final method.

5.3 Principal Component Analysis-Based Normal Vector Extraction

Principal Component Analysis-Based Normal Vector Extraction is an approach for curb detection that uses the normal vector to determine the likelihood a set of points represent a curb [26]. A normal vector is a vector that is orthogonal to the plane of points. Curbs are expected to have a strong normal vector component in the X direction (orthogonal to the direction of travel) and a weak component in the Z direction (height). Based on these criterias, normal vectors can be computed on the real point cloud data using a sliding window approach. Based on previous experiments, it was found that a bounding box of 2 laser lines and 4 points per laser gave the best results.

PCA computes vectors that correspond to the variance of the dataset. **Figure 5.5** below showcases two vectors in red that correspond to the variance of the blue dataset. Each vector in red is a principal component vector. The normal vector is the principal component vector with the smallest eigenvalue pair.

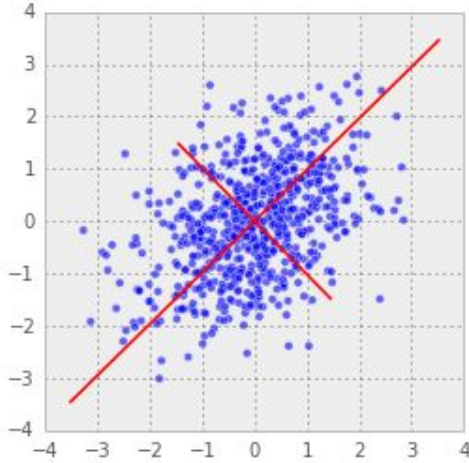


Figure 5.5: Principle Component Vectors

To compute the normal vector of a set of points, eigenvalue decomposition must be done on the covariance matrix of the points. Therefore, the first step in the process is to compute the covariance matrix. The covariance matrix is a square matrix that gives the covariance between each random variable. In this case the covariance matrix is 3x3 where the random variables are the coordinate vector of each point which is $x = [x^{(1)}, x^{(2)}, x^{(3)}]$. First, the mean vector of the set of points is computed by averaging each component of the coordinate vector. Second, the covariance matrix is computed using the formula below. Where N is the total amount of points in a bounding box, x_i is a coordinate vector for a point, and \bar{x} is the mean vector.

$$Cov = \frac{1}{N} \sum_{n=1}^N (x_i - \bar{x}) \cdot (x_i - \bar{x})^T \quad (4)$$

Third, eigenvalue decomposition must be computed to retrieve the smallest eigenvalue and vector pair. The eigenvector with the smallest magnitude eigenvalue will be the normal vector. A numerical method called Power Method was used to calculate the smallest eigenvalue and vector pair [37]. The Power Method is used to find the dominant eigenvalue and eigenvector pair, so the Power Method needed to be modified. To find the smallest eigenvalue and vector pair, the dominant one is found first and then is subtracted from the diagonal of the covariance matrix. The covariance matrix is negated and the Power Method is applied again. The results from the Power Method now provide the smallest eigenvalue and vector pair.

The normal vector is the eigenvector that corresponds to the smallest eigenvalue. Using the normal vector, subsections of the real point cloud matrix can be ignored. Subsections with a strong normal vector component in the Z (height) directions are most likely road points or other noisy points and can be removed. Subsections with a strong normal vector component in the X direction are most likely curbs and remain. A threshold value was experimentally set for the Z and X normal vector components.

This method worked better than the MSE method described in **Section 5.2**. This method was able to remove a significant amount of road and noise points, leaving mostly points that resemble the curb. However, some limitations of the method were noticeable and limited the total curb detection possibilities. First off, the locality of points for two adjacent laser lines varies based on the surface the LiDAR detects. For instance, a sharp elevation will introduce a large variation in the distance between adjacent points of a laser line. This means that the sliding window approach will not always retrieve points that are adjacent to each other. This is a very large limitation as the method expects tight locality of points. Second, noise on the curbs themselves may affect the normal vector calculations and result in a strong normal vector component in the Z direction. It was found that this method is not as resilient to noise compared to other methods. Third, other methods were found to have better road and noise point removal than this method.

The Principal Component Analysis-Based Normal Vector Extraction method was not used as the main road and noise removal method. However this method is very promising in categorizing segments and clusters of the point cloud based on the normal vector. It is known that curb segments will have a strong normal vector component in the X direction and a weak component in the Z direction. Through better clustering and grouping of points, this method becomes very promising and is used in the final curb detection method.

Figure 5.6 below showcases the results of the PCA method onto the point cloud data. Curbs are shown to the left and right of the middle halo. The curb resolution is significantly better than the method described in **Section 5.2**. It is also shown that road points are removed from the point cloud. While curb detection is good, **Section 6** details a stronger curb detection algorithm with better detection and less false positives.

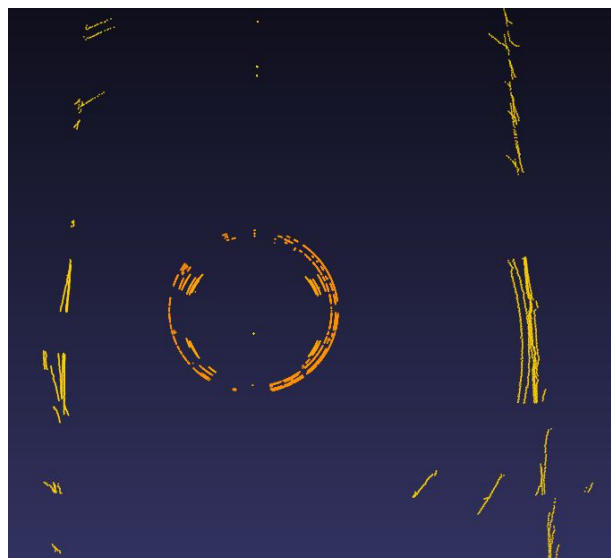


Figure 5.6: Results from PCA on point cloud

Chapter 6: Curb Detection Method

The methods examined thus far have failed as effective curb detection algorithms. It was shown in **Sections 4.1** and **4.2** that using transforms as a method of curb detection failed due to the over-reliance on the algorithm without employing filtering steps to remove non-curb points. **Section 4.3** showed that an edge detection algorithm was only partially effective in removing non-curb points. These results together suggest that no single algorithm should be relied on to perform curb detection. A method like the Hough transform might certainly have worked given sufficient constraints (straight, continuous curb), under a scenario where many non-curb points had already been removed. The Hough transform has already been proven capable in the camera-based methods described in **Chapter 3**, and there is no reason to doubt its capabilities in LiDAR point clouds. **Section 5** examined sliding window approaches and how, when considering multiple laser lines, they failed to detect curbs or to adequately remove points from the point cloud. Sliding windows were adapted from the ideas of CNNs, where a learned filter is convolved over an image or a data-space. This idea is still useful for curb detection, but filters for single curb lines must be developed to avoid the issues presented above. Overall, the experiments conducted for the methods in **Chapter 4** and **5** were useful and they informed the development of the curb detection method. For instance, sliding windows and PCA are still used, be it in different ways. These experiments were, for the most part, inspired by the papers presented in **Chapter 3**. The papers introduced many useful concepts that were, along with novel methods, instrumental in the development of the curb detection and road detection methods.

This chapter shall henceforth describe the method of curb detection developed in the course of this project. As described in **Chapter 3**, LiDAR is an effective sensor for environmental perception due to its resilience to noise. Many of the best-performing road detection algorithms employed multiple sensors to detect the road, using sensor fusion techniques to mutually detect and corroborate the road. This idea was explored, but ultimately it was decided that the project would explore detection through LiDAR solely⁸. As such, the curb detection method was designed to employ the filtering methodology: successively remove points until a curb can be confidently extracted. The extraction methodology is also important to highlight. As was made clear in the analysis of [25], a clustering and regression step prior to curb extraction is required for higher resilience against false positives. Furthermore, it was discovered that a new detection mechanism was required to more effectively report the curb location. With these incites, the curb detection method could be designed. The rest of this chapter proceeds as follows: First, a preprocessing step removes duplicate points and performs basic road segmentation. Then, the filtering steps are described. Next, clustering is used to group similar points. Then, the point cloud is de-noised, segmented, and the segments are grouped to produce the final detection result. Each step of the algorithm is analyzed to ensure computational complexity remains

⁸ There are a number of reasons why the scope was restricted, namely, the difficulty of fusing sensors coherently to produce a more effective detection, and a restriction from QNX, who sponsored our project.

reasonably low; after all, point clouds are large data structures so it is desirable to avoid large complexities. Once a set of segments is grouped, a confidence metric is used to assign a probability of detection. Finally, real-time speedup methods are described.

6.1 Preprocessing

In the first step of the curb detection algorithm, the raw LiDAR data received from the HDL64E sensor is processed and converted to a format that is easier to work with. All of the vehicle's sensory data for a given time frame is received in the form of a snapshot which includes the LiDAR point cloud. The LiDAR point cloud is then extracted from the snapshot. The HDL64E LiDAR sensor used for this project was configured with the vehicle's computer system, every point in the cloud is duplicated and therefore these duplicates must be removed as they are not necessary and increase computation time. This is done by iterating over the point cloud and ignoring every second grouping of 64 points per their laser ID. In this step, the size of the data is reduced from 233,088 to 116,544 points. The data points are then converted from spherical coordinates to cartesian coordinates using the following three equations to obtain their x , y , and z values:

$$x = d * \cos(Ei) * \sin(a) \quad (5)$$

$$y = d * \sin(Ei) + h \quad (6)$$

$$z = d * \cos(Ei) * \cos(a) * -1 \quad (7)$$

where d is the distance of the point from the vehicle, Ei is the elevation height of the respective laser ID i , a is the azimuth angle from the vehicle, and h is the height offset from the LiDAR sensor to the ground. Next, a height filter is applied to the point cloud to remove all points above a height of 0.3m above ground level. These points are not necessary as they are far above the 0.2m region where curbs are expected to be found as described in **Section 3.2**. A leeway of 0.1m is thus allocated for robustness. This step removes several thousand points from the point cloud. Finally, the data structure described in **Section 3.3** is initialized and each of its rows are populated with the cartesian points of the corresponding laser ID.

6.2 Filtering

The task of extracting road curbs from a point cloud is rendered difficult by the size of the point cloud and noise. The process of filtering is used to remove unwanted points in the data on the assumption that they share different properties from the curb points. The filtering methods detailed in this section consider noise to be any LiDAR points that are distant, random, or otherwise highly variant from the expected curb and road-area characteristics. The filtering method, unlike the transforms described in **Chapter 4**, operates on the basis of conditional statements. In other words, after performing arithmetic operations, points are removed from the point cloud based on whether the result of the operations exceeds predetermined thresholds. The first step of the filtering method, described in 6.2.1, is the Similarity Filter. In this step, the

similarity between adjacent points on the same laser line is evaluated based on the angle between points and height differences. This step is effective for reducing most of the noise in the point cloud; however, it fails to detect curb points that are directly adjacent to the vehicle. The second step, described in 6.2.2, is the Augmented Similarity Filter which is used to compliment the Similarity Filter in areas where it fails. This step uses PCA to maintain resolution of the curb near the vehicle. Together, these steps reduce the point cloud to the candidate curb points on which further analysis is conducted for extraction. The filtering step yields a significant reduction in the size of the point cloud down to approximately 4000-6000 points (96% reduction).

6.2.1 Similarity Filter

The similarity filter is the first filtering step of the curb detection algorithm. According to the filtering methodology, this step should remove non-curb points such as noise, greenery, the road, other cars, etc; while maintaining as many on-curb points as possible. The papers presented in **Chapter 3** termed this process “candidate point extraction” with the specific goal of identifying curb points by searching for curb-like features in the point cloud. These features include: the height difference, the gradient, the normal vector, and the angle between adjacent sets of points [23, 25, 26]. The similarity filter uses two features to distinguish curb points: 1) the angle between adjacent points, and 2) the height difference between adjacent points. The former is directly inspired by [23] and [25], but uses a different approach to calculate the angle. The latter is a novel technique which, when used with the former, allows for effective differentiation of curb points. In unison, these features allow for the identification of ground points and noise points versus what is expected from curb points.

The similarity filter was designed based on real LiDAR data, and the features that were observed of curbs, the ground, and noise. Upon analyzing points on the road and curb, it became clear that adjacent road points on the same laser line had small height variations not exceeding 1 cm, while curb points had height variations between 3 cm and 5 cm. This observation could be used to differentiate points on the road versus points on the curb. However, many noise points (e.g. shrubs, grass, cars), would often exhibit the same height variation as a curb, so this feature, the height variation, was effective, but not sufficient, for curb extraction. Further analysis revealed the angle between adjacent curb points on the same laser line was around 180°; the same held true for adjacent road points on the same laser line. Noise, however, would often exhibit large differences in angle between adjacent points. Using these qualities, the curb points could be differentiated from the road and noise points.

The first step in computing similarity of adjacent points is evaluating the cosine of the angle, denoted by α , between two vectors that separate five adjacent points on the same laser line. This will yield a value between -1 and 1, however, the absolute value is taken as the orientation of the angle is irrelevant. The angle is calculated based on all three components (i.e., x, y, and z) of the points. This is different from [23] and [25], where the angle is calculated based only on the x- and y-components, on a variable number of points depending on the vertical angle of the

laser. **Figure 6.1** illustrates the calculation of the angle. Let a , b , c , d , and e , be the five three-dimensional adjacent points on the same laser line in the point cloud. The process of computing the similarity in angle, is broken down into three steps below:

- 1) Compute a new middle point between ab and de , yielding points f and g respectively,
- 2) Compute the direction vectors $v1$ from point g to point c and $v2$ from point g to point c ,
- 3) Compute the cosine of the angle between vectors $v1$ and $v2$ using the following formula:

$$\cos(\alpha) = \frac{v1 \cdot v2}{||v1|| ||v2||} \quad (8)$$

where $v1 \cdot v2$ is the dot product between both vectors and $||v||$ is the euclidean distance of the vector.

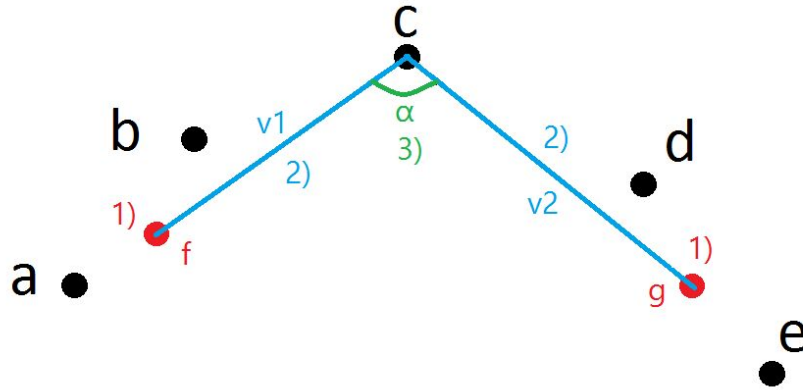


Figure 6.1: Cosine Similarity Visualization

To determine similarity, the cosine of the angle is used in conjunction with the height difference of the adjacent points. The groupings of five points that yield a cosine greater than 0.95 and a height difference between $v1$ and $v2$ that is greater than 1 cm but less than 5 cm are considered to be similar and are kept in the point cloud. All other groupings of five points that do not adhere to these ratios are discarded. Once this operation is completed, the process is repeated for the next group of five adjacent points on the laser line in a sliding-window fashion to avoid overlaps in processing the points. It is then again repeated for each of the 64 laser lines in the point cloud. The defined numerical ratios were picked through observation of the trends in the LiDAR data as well as through experimentation. They have provided desired results in filtering noise in the data. It was however noticed that points on the curb near the vehicle were involuntarily removed by the similarity filter due to their almost negligible height difference. **Figure 6.2** below shows the LiDAR point cloud before and after applying the Similarity Filter:

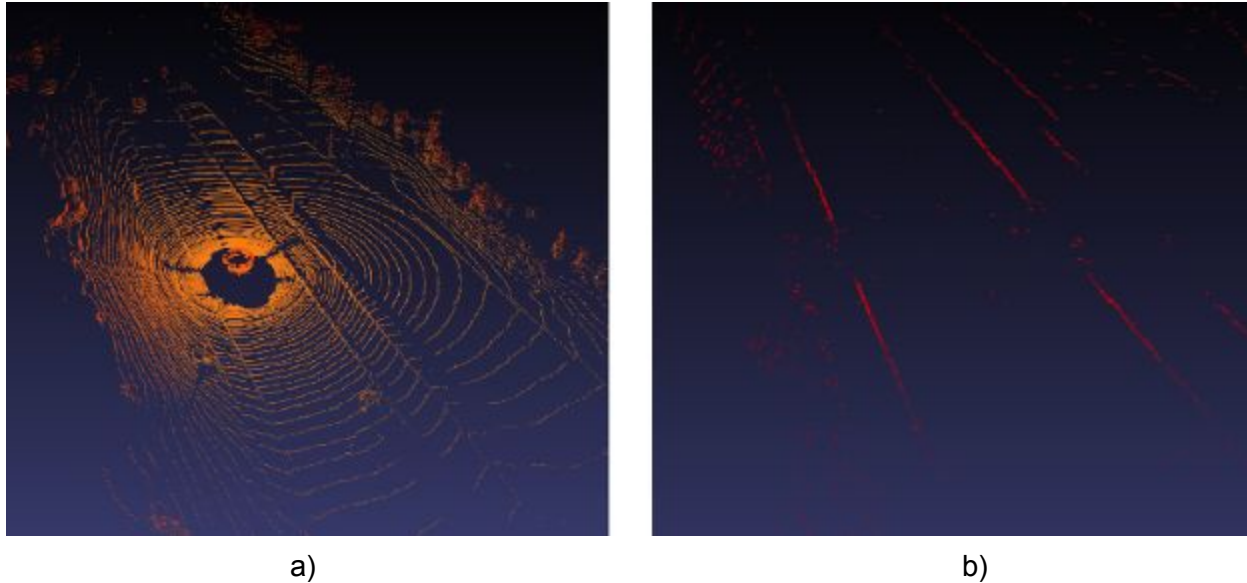


Figure 6.2: LiDAR point cloud before and after the Similarity Filter: a) The untouched LiDAR point cloud. b) the LiDAR point cloud after the similarity filter.

It is worth noting that in **Figure 6.2 b)**, most of the road, sidewalk, LiDAR halo, and trees have been eliminated from the data. On the other hand, parts of the curb directly adjacent to the vehicles have also been removed. For this reason, the similarity filter was combined with a second filter to retain the curb points near the vehicle.

6.2.2 Augmented Similarity Filter

The similarity filter is unable to detect curb points directly to the sides of the vehicle (left and right sides extending about 5m centered on the y-axis). This is due to the fact there is no height difference between adjacent points on curbs directly to the side of the vehicle. To fix this issue, Principal Component Analysis (PCA) was used to detect curbs beside the vehicle. As described in **Section 5.3**, PCA is used to extract the normal vector from a set of points. A curb is expected to have a strong normal vector component in the X direction, and a weak normal vector component in the Z direction. Although the PCA method described in **Section 5.3** was not used directly in the curb detection method, **Figure 5.5** shows strong curb detection for curbs beside the vehicle. This is due to the high density of points near the vehicle. A sliding window approach is then sufficient for detecting curbs directly to the side of the vehicle.

An approach similar to **Section 5.3** was then created. Using a bounding box of 2 lasers and 8 points per laser, PCA was computed using a sliding window approach on the point cloud. The main detection of the augmented similarity filter are the curbs to the side of the vehicle, therefore PCA does not need to be computed on the entire point cloud. It was experimentally found that curbs directly beside the vehicle were captured between 0 and 30 laser line ids. Therefore the augmented similarity filter is computed for half the point cloud.

The augmented similarity filter is run independently from the similarity filter and results are combined once both filters finish. This allows for parallelism and further speedup. For each window in the point cloud, the normal vector is computed and compared to a threshold. If the x normal vector component is above T_x and the z normal vector component is below T_z , then the points in the window are saved. **Figure 6.3** showcases the combined results of the similarity filter and the augmented similarity filter.

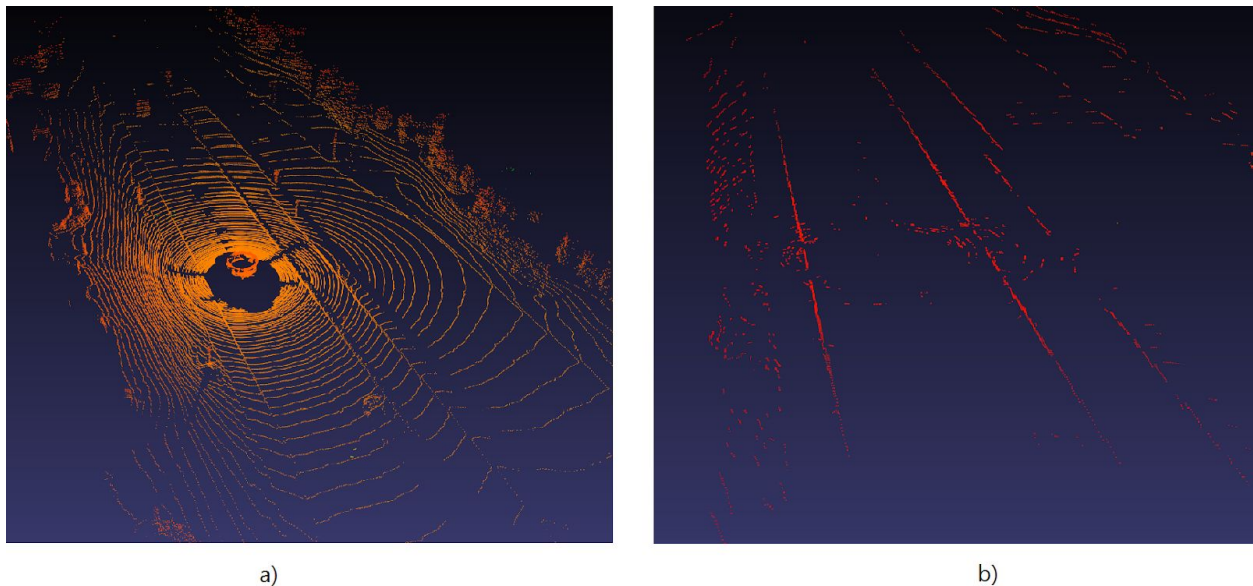


Figure 6.3: LiDAR point cloud before and after the Augmented Similarity Filter: a) The untouched LiDAR point cloud. b) the LiDAR point cloud after the similarity and augmented similarity filters.

It is worth noting that in **Figure 6.3 b)** the curb points directly adjacent to the vehicle remain, however extra noise is introduced on the road in the vicinity of the car. The extra curb points lead to better detection results for later steps, but the extra noise has a negative effect, and has the potential to increase miss-detection.

6.3 Clustering

Clustering is the process of divvying data points from a large data set into subsets based on similarity of shared properties. Ideally, all data points within one subset are more similar to one another than they are to other points in the point cloud. In regards to a LiDAR point cloud, similarity of points can be evaluated on the basis of point locality and point density. Clustering can thus be used to group neighboring points in 3D space based on each point's x, y, and z coordinates. By using point locality as a clustering property, the point cloud becomes structured into groups of points representing different entities. Clustering is therefore essential for grouping and classifying the curb points as a separate entity from the rest of the point cloud. This then allows for extraction and further analysis of these curb points.

The research papers presented in **Chapter 3** proposed using a variation of Density Based Spatial Clustering of Application with Noise (DBSCAN) for clustering LiDAR points [23]. DBSCAN is a density-based clustering algorithm that typically uses the euclidean distance between neighboring points and the density of nearby points to group them into clusters. It is also capable of identifying outliers in the data that do not fit into any cluster. DBSCAN differs from other clustering methods such as distance-based algorithms in multiple ways. For one, like distance-based clustering algorithms, DBSCAN evaluates the distance between neighbouring points, but it also evaluates the density of nearby points as well [38]. Distance-based algorithms such as K-Means attempt to divide the data into a pre-defined number of clusters while DBSCAN does not require the number of clusters to be known ahead of time [39]. Additionally, DBSCAN is more effective at identifying shapes and linearly-separable clusters. **Figure 6.4** and **6.5** illustrate these differences.

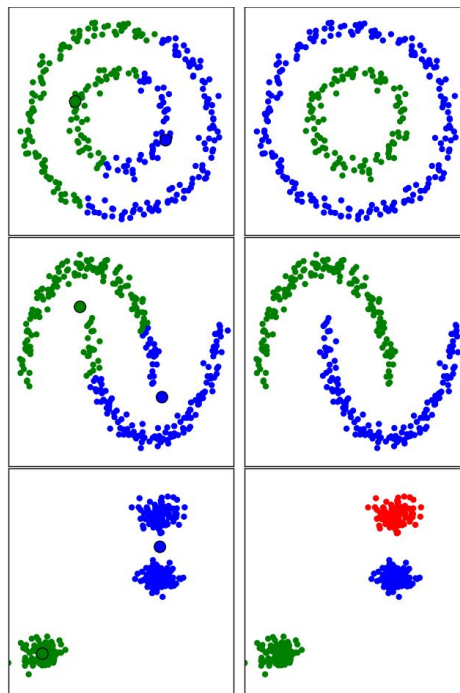


Figure 6.4: Distance-based K-Means ($k=2$) (left) vs density-based DBSCAN (right) [40]

In **Figure 6.4**, it is evident that DBSCAN is better at identifying shapes and non-linearly separable clusters even when the number of clusters is unknown ahead of time. **Figure 6.5** shows how DBSCAN is able to identify clusters of varying point density.

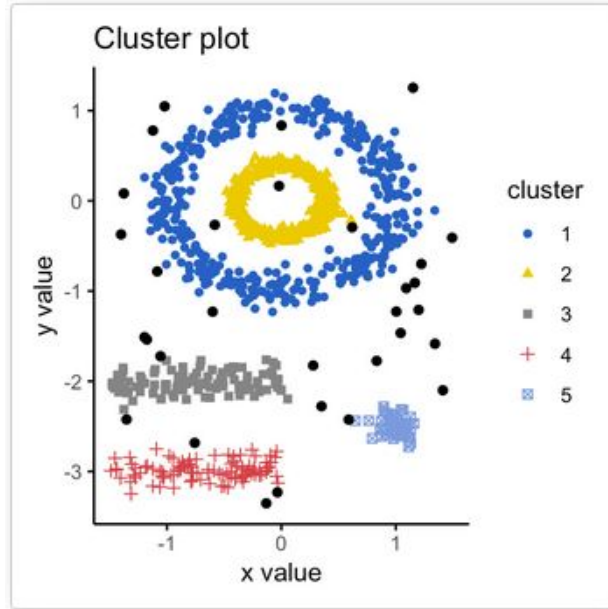


Figure 6.5: DBSCAN Clustering [41]

In **Figure 6.5**, the DBSCAN algorithm computed five clusters identified by the colours: dark blue, yellow, grey, red, and light blue. It is visible that each cluster has a different point density, and shape. The outliers are illustrated as black points that do not fit into any of the clusters.

Given that the LiDAR point clouds captured from the vehicle are subject to large variability due to the vehicle's changing environment, it is impossible to predefine the number of clusters in the data. Each captured point cloud in itself has a large variation in point densities. For example, points on the road or the curb near the vehicle are more dense than points further from the vehicle. The point clouds also contain a large amount of distant outlying data points that would ideally be removed. For these reasons, the density-based DBSCAN algorithm was selected over other algorithms to cluster the LiDAR point cloud.

The DBSCAN algorithm has two parameters. The first parameter, denoted *epsilon*, is the minimum distance from one point to another for them to be considered in the same cluster. The second parameter, denoted *minPts*, is the minimum number of points required to form a cluster. The algorithm begins by determining all core points in the data set. These are points that are surrounded by *minPts* or more points that fall within the *epsilon* range. Next, for each core point, border points are identified. These are points that have less than *minPts* points within *epsilon* but fall within *epsilon* of a core point. All core points that fall within the *epsilon* of another core point are grouped together into a cluster. All border points are added to the cluster of their closest core point. All other points that do not fall within the *epsilon* range of a core point are considered to be noise. This is illustrated in **Figure 6.6**.

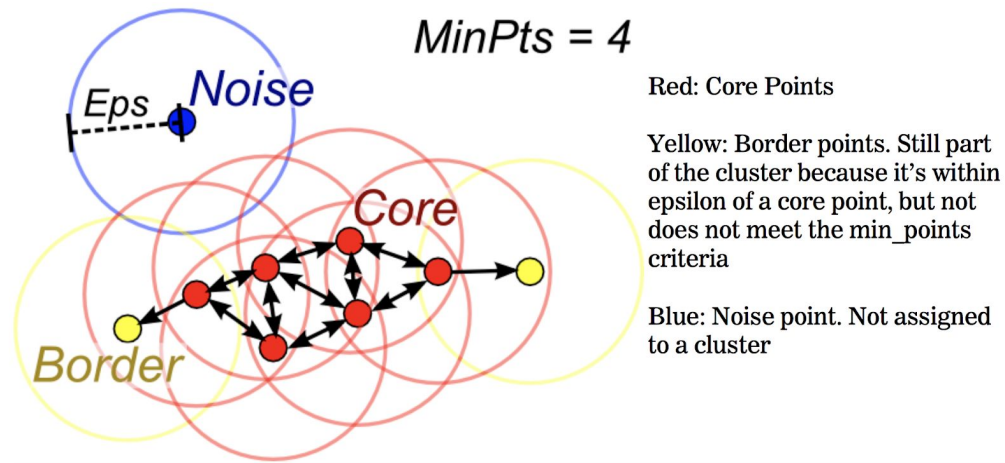


Figure 6.6: DBSCAN core, border, and noise point identification [38]

After experimentation, the *epsilon* value was chosen to be 0.5 and *minPts* was chosen to be 5. These values yielded desirable results in clustering the point cloud as shown in **Figure 6.7** below:

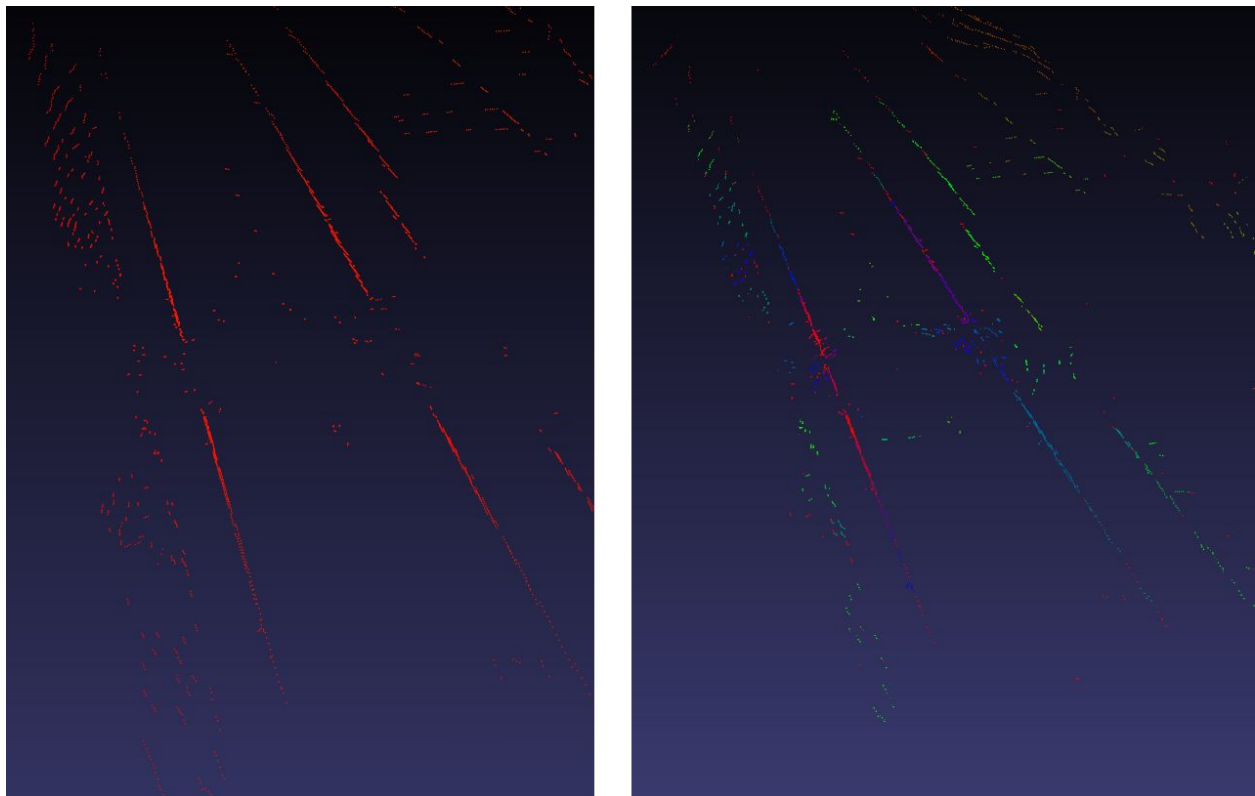


Figure 6.7: LiDAR point cloud before (left) and after (right) DBSCAN clustering

In **Figure 6.7**, the DBSCAN clusters are shown by color. It is visible that points in linear road curb segments have been clustered together. Now that the point cloud and the curb points have been effectively clustered, further analysis is possible on individual groupings of points.

While DBSCAN proved effective for clustering the point cloud, it was noticed that at further distances from the vehicle, the points that formed the curb were not properly clustered together. Rather than one linear cluster for the curb line, the algorithm split the curb into multiple short line clusters. This was due to the density of points in the point cloud decreasing as their distance from the vehicle increases. In other words, at further distances from the vehicle, curb lines had a varying point density. DBSCAN thus fell short in this aspect as DBSCAN assumes clusters to be of uniform density within themselves. For this reason, the Hierarchical DBSCAN (HDBSCAN) was investigated. HDBSCAN, like DBSCAN, is a density based clustering algorithm however, it differs in that HDBSCAN supports varying in-cluster density. HDBSCAN also does not require a set *eps* value as it varies for each cluster.

6.4 Denoise

Once the point cloud is clustered, further noise reduction is required to achieve effective curb detection with few false positives. The PCA-based normal vector extraction of **Section 5.3** is reproduced here to denoise the clusters. The normal vector is used to classify a cluster of points as curb candidates based on the magnitude of the X and Z components. For a cluster to be considered as a curb candidate cluster, it is expected that it will have a strong normal vector component in the X direction, and a weak normal vector component in the Z direction. Several limitations for PCA with the sliding window approach were noted in **Section 5.3**, however, the preceding clustering step removes these limitations. First off, point locality is maintained in each cluster which is crucial for PCA. It is less likely for normal vectors to be calculated between far away points, thus, normal vectors are able to convey more accurate results. Additionally, the clustering of the point cloud separates potential curb clusters from noise, which improves the accuracy of the normal vector. These improvements to point grouping are advantageous for PCA as more accurate normal vectors can be calculated.

The denoising step iterates through every cluster and calculates the normal vector using PCA. Two threshold values were experimentally chosen for the X normal vector component (T_x) and Z normal vector component (T_z). To reduce the possibility of removing curb clusters, the threshold values were made lenient. Later steps in the curb detection method also remove noise, so it is central for this step to maintain as much curb resolution as possible. The X and Z normal vector components of each cluster are compared to T_x and T_z . If the X component is above T_x and the Z component is below T_z , then that cluster is considered a curb candidate cluster and is maintained. Clusters that fail the comparison are treated as noise and removed.

Due to the large amount of points in a point cloud, it is crucial to use efficient methods in processing the point cloud. The time complexity of the PCA method used above is $O(n)$ since

each step in the PCA method is iterative. This means that a single pass through each cluster is required.

After further experiments, it was found that the PCA method was reducing the resolution on the curb. The threshold values were modified to be even more lenient, however doing so introduced more noise into the point cloud. Clearly a modified approach was required. It was noticed that clusters that represented curbs contained some form of noise. This noise perplexed the PCA method which calculated a normal vector that did not fall within the thresholds. The noise was usually contained within a section of the curb cluster. Therefore a modified method was developed that calculated the normal vector on 50 adjacent points within a cluster.

The modified method required that the points within clusters be sorted for easy iteration. A sorting algorithm called Quick Sort was used to sort the points in each cluster. Points were sorted based on their Y coordinate value and were sorted in ascending order. The time complexity of Quick sort is $O(n\log(n))$. Once the cluster is sorted, PCA is computed on 50 adjacent point increments, with leftover points combined at the end. If the normal vector computed falls within the thresholds, the 50 point group is maintained. This will be repeated until all the normal vectors are calculated for all the points within the cluster. The advantage of the modified method is that it is less resilient to noise since usually only a section of curb clusters will contain noise. The modified method is then able to keep more curb resolution, while also removing more noise. The disadvantage is that the time complexity has increased, and that PCA must be computed on 50 point iterations in a cluster instead of for the whole cluster. This performance hit is manageable since the similarity filter step of the curb detection method described in **Section 6.2** removes a significant amount of points from the point cloud. Due to the large point removal from previous steps, more time consuming and accurate methods are able to be chosen in this step.

Figure 6.8 below showcases the results of the denoise step when using PCA. It is shown that many noise points are removed and candidate curb clusters mostly remain.

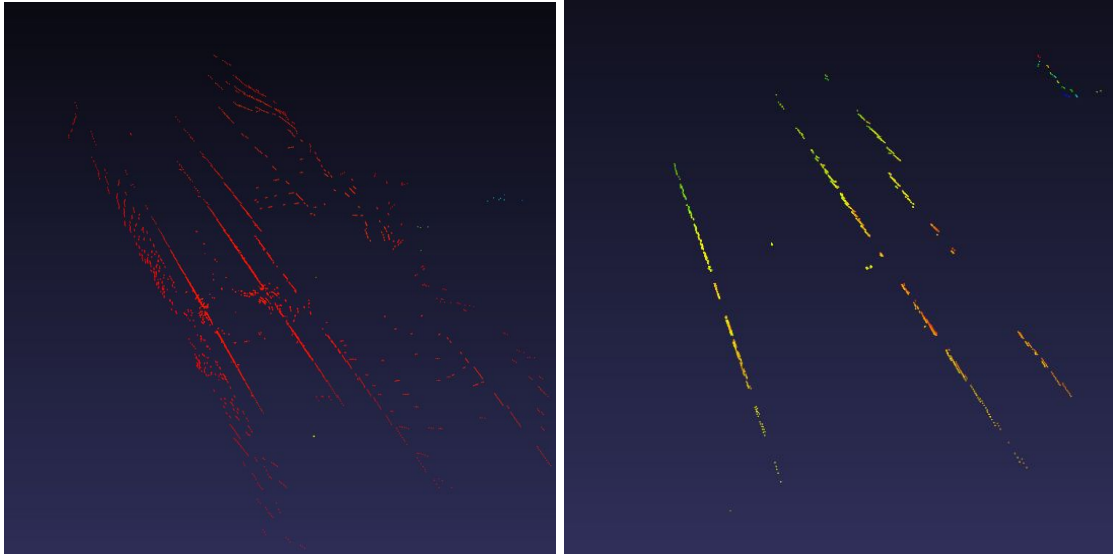


Figure 6.8: Point cloud before de-noising (Left) and after (Right)

Figure 6.8 showcases the strength of PCA to extract curbs and other objects that have similar properties. It is clearly seen that the right image contains mostly curbs without any points on the road. The papers presented in **Chapter 3** described the use of PCA through a voxelization approach. In contrast, the PCA implementation focuses on clustered point cloud data which allows for varying cluster sizes and densities. A clustered approach allows for clusters to represent distinct objects in a point cloud such as curbs. This allows PCA to be more effective in extracting normal vector components from clusters that already represented objects within a point cloud.

6.5 Segment and Linear Fit

The preceding steps of the curb detection method significantly reduce the number of points in the point cloud, leaving primarily curbs and some noise. To further extract and model curbs, the point cloud is split into two separate point clouds based on the X value of points. All points with a negative X value are segmented off into a separate point cloud. This is done so curbs on either side of the vehicle can be detected and extracted independently. The assumption of curbs being on the left and right side of the vehicle, as described in **Section 3.2**, is used to split the point cloud. The main focus of the curb detection method is to detect curbs to the side of the vehicle and not directly in front. These curbs will not be affected by the point cloud segmentation and will be captured fully within their respective point cloud segment.

The papers in **Chapter 3**, specifically [23, 26], use linear regression with quadratic equations to model curbs. Curbs follow a circular relationship beside curved roads, and are mostly linear beside straight roads. Several complications arise when using quadratic models for curbs. First, curved curbs would need to be sectioned off from linear curbs. The clustering algorithm described in **Section 6.3** does not differentiate between curved and linear surfaces, and therefore would most likely cluster them together. Second, multiple models would need to be

created to accurately model complicated roads and intersections. Third, noise will interfere with more complex modeling compared to simpler approaches such as a linear model. Modeling curbs using quadratic equations was not chosen for the curb detection method, instead curbs are modeled as small linear segments. This allows for variation in curvature of curbs and allows modeling of complex roads and intersections. In addition, reporting small changes in the curb allows for the reporting of instantaneous rate of change of the curb. This was not used in the curb detection method, however it is useful when further analyzing the curb. With the basis that the curb will be modelled as small linear approximations, the 50 point sections passed down from **Section 6.4** will need to be further cut down. It was found that 25 point segments resulted in a good linear approximation.

Each section passed down from the Denoise step is split into 25-point segments while retaining point locality since sections are sorted based on y-value of points. Leftover points are combined into a segment smaller than 25 points. Linear regression is run on each segment. Linear regression is run using the y-value of points as the independent variable and the x-value of points as the dependent variable. Segments that represent curbs are expected to have high linearity since small curb segments are approximately linear. **Figure 6.9** shows a detected curb section. It is clearly shown that the curb is linear and therefore can be modeled using linear models. Furthermore, small segments of curved curbs will still be mostly linear and will also be represented as linear models.

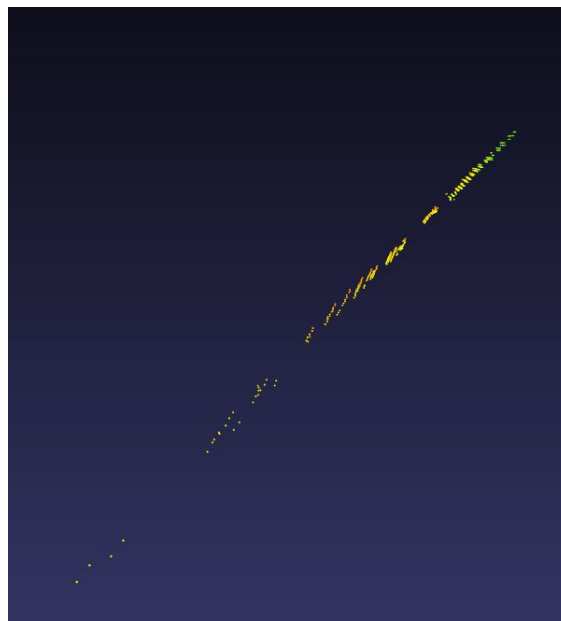


Figure 6.9: Curb detected in point cloud

The slope, intercept, and linear correlation coefficient (r^2) are computed for each segment. Segments that showcase a high linear correlation coefficient are kept and passed to later stages in the curb detection method. A threshold value for linear correlation coefficient was experimentally found to be 0.8. Segments that have a linear coefficient less than the threshold

value are removed. In most cases those segments represent other objects in the point cloud including noise, and removing them is not detrimental to curb detection. Segmentation and linear fit is an important step in the curb detection method since it allows for further noise removal, and more effective curb modelling compared to the large scale linear and quadratic models. This allows later steps in the curb detection method to take advantage of the linear representation of curb segments. The slope, intercept, and linear correlation coefficient are stored for each segment and used later on in the method.

6.6 Segment Grouping

The preceding steps deal with point removal from the point cloud. The goal of these steps is to remove points that do not represent curbs. Segment grouping, on the other hand, assumes that sufficient point removal has been done. From the previous step, each cluster has been broken down into 25-point segments, and linear regression was calculated for each segment. Each segment is then modeled as a line. However, the total number of segments in the point cloud is large, and each segment models a small part of the point cloud. To effectively output the location of curbs, these segments must be adequately grouped together.

To reduce the grouping complexity, segments within a cluster will be grouped. Clusters are not grouped amongst themselves during this step. The motivation behind the groupings of segments is to convey a simple representation of each cluster, and to allow further steps in the method to give confidence of detection for each cluster. Segment grouping is done through three criteria: 1) relative x location of segments, 2) relative y location of segments, and 3) through a novel similarity metric. First off, segments within a cluster need to be sorted in ascending order based on the average y value of each segment. This is done so that adjacent segments can be compared. Next, each adjacent segment is compared, and a list is formed for adjacent segments that meet the three criteria. These lists contain the segment groupings. More segments are appended to the list that meet the criterias. When a segment does not meet all the criterias, a new list is created and a new grouping of segments is formed. A cluster may contain multiple groupings of segments.

The first grouping criteria is the average x value of adjacent segments. It is paramount that close segments are grouped together, so looking at the distance between them is useful. The average x value of adjacent segments is compared, and the absolute difference is taken. If the difference lies below a threshold T_x , then the next criteria are computed. Otherwise a new segment grouping is created. The second grouping criteria is the average y value of adjacent segments. Similar to the previous criteria, segments are grouped when they are close together. The average y value is computed for adjacent segments, and the absolute difference is taken. If the difference lies below a threshold T_y , then the final criteria is computed. Otherwise a new segment grouping is created.

The final criteria is the similarity metric. The similarity metric compares the slope and intercept of adjacent segments. The similarity metric is high when slope and intercept of adjacent segments

match. The linear equations of adjacent segments are represented as vectors D and L , which include the slope and intercept values of each segment.

$$D = \begin{bmatrix} m_D \\ b_D \end{bmatrix} \quad L = \begin{bmatrix} m_L \\ b_L \end{bmatrix} \quad (9)$$

The similarity metric is then computed between vectors D and L using the following equation, where ρ is the similarity value.

$$\rho = 1 - \frac{\|D - L\|}{\|D\| + \|L\|} \quad (10)$$

The similarity metric is bounded between 0 and 1, and the closer it is to 1, the higher similarity D and L have. Using this metric, similarity of adjacent segments is computed and compared to a threshold T_s , which was experimentally found to be 0.6.

The similarity metric compares the slopes and intercepts of two line segments. The similarity metric returns 1 when both slopes and intercepts match up, and trails off to 0 when they are dissimilar. **Figure 6.10** showcases the effects of changing the slope and intercept for a line equation $y = x + 1$. The graph on the right illustrates the effect of a changing slope to the similarity metric. As the slope deviates from 1, the similarity metric drastically falls to a value near 0. The results of a changing intercept are similar, and are illustrated by the graph on the left.

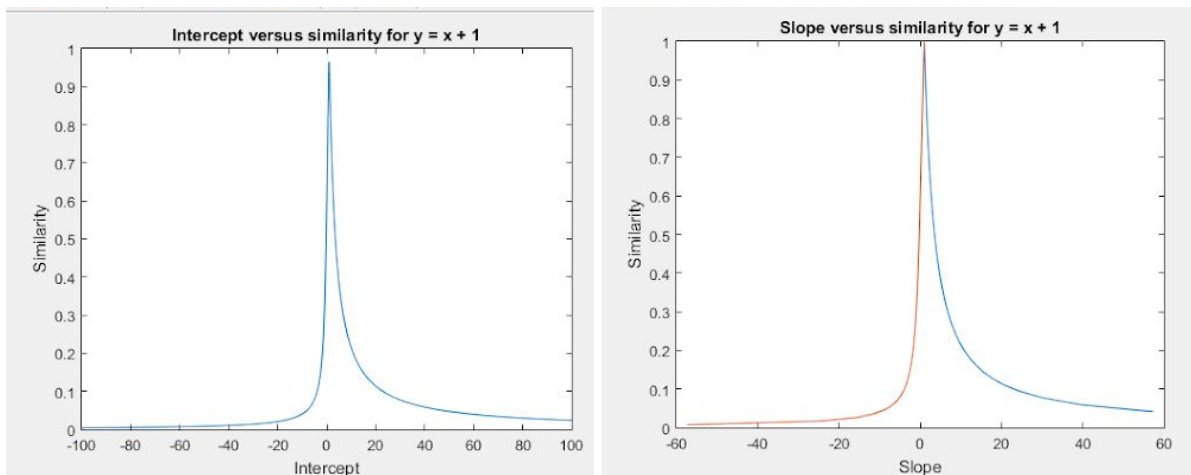


Figure 6.10: Effects of changing intercept (left) and slope (right) for similarity metric

Figure 6.11 below combines the effects of changing intercept and slopes into a single graph. It is shown that when two line segments have similar slope and intercept values, the similarity

metric will output a value close to 1. As the slope and intercept get dissimilar, the similarity metric drops.

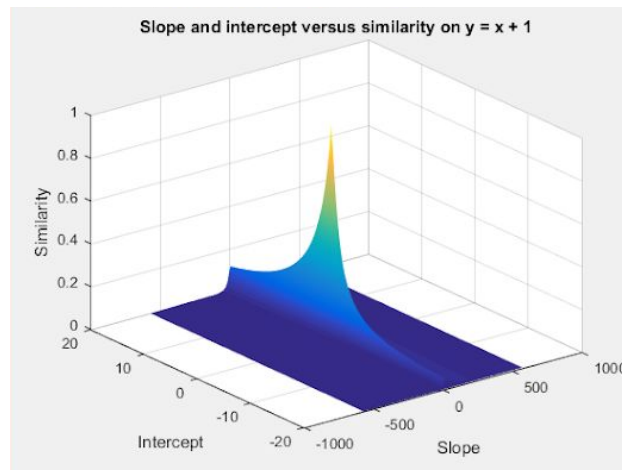


Figure 6.11: Effects of changing intercept and slope for similarity metric in 3D plot

The similarity metric is powerful in quantifying the likelihood that two adjacent curb segments belong to the same curb. As stated in **Section 6.5**, curb segments are approximately linear, and are adjacent to each other.

Figure 6.12 below showcases the results of the segment grouping.



Figure 6.12: Curb Segment Grouping

It is clearly shown in **Figure 6.12** that multiple segments have been grouped together. Each segment grouping is denoted by a different colour. For each grouping of segments, minimum

and maximum values are computed for x and y coordinates. The average x and y values are computed, as well as the standard deviation. These groupings of segments are considered curb points and are used to convey curb detection results. All of these metrics help convey the location and structure of the curb that the segment grouping represents.

6.7 Assigning Confidence to Detected Curbs

Confidence values give a level of assurance to detected objects in the point cloud. The curb detection method is not perfect, and the method will make mistakes and detect false positives. These false positives may be detrimental to later steps in the ADAS stack. Furthermore, confidence is required when reporting detection to higher level sensor fusion software. Not every segment detected will represent a curb, so a method to give confidence to segments is crucial. **Section 3.2** describes the expected structure of curbs, as well as the assumptions made prior to detection. Curbs are expected to be between 0.2m tall, with a width of 0.2m, and curb segments are expected to be near-linear. Based on these characteristics, segments can be tested for compliance and given a confidence score.

Confidence metrics give the curb detection method resilience to noise and false positives by flagging false positive segments with a low confidence score. This improves the accuracy and reliability of the curb detection method since the detection is tested based on curb characteristics. The confidence metrics developed for this project allow for configurable time and reliability performances. These confidence metrics are bounded between 0 and 1, where the closer the confidence is to 1, the more confident the method is that the detection is a curb. The methods can be configured to tradeoff speed versus accuracy when assigning confidence. These configurations are made on the fly and are dependent on the speed of the vehicle. When the vehicle is driving slowly, the real-time deadlines are more lenient, and therefore more accurate confidence methods can be employed. On the other hand, when the vehicle is driving fast, the real-time deadlines are more strict, and therefore more efficient and quicker confidence methods are required.

Two separate confidence values are calculated, one for segments within clusters, and another between clusters on each side of the vehicle. The sections below describe the two confidence metrics and their usage. **Section 6.7.1** and **Section 6.7.2** describe the confidence metrics intra- and inter-cluster respectively. Furthermore, **Chapter 7** describes a different curb detection method which is corroborated with this method. A method of corroboration was required which used the LiDAR data in a different way to generate curb locations. This new method can be used to further enhance the intra-cluster confidence.

6.7.1 Intra-Cluster Confidence

Curbs are composed of one or more clusters, so confidence for each cluster needs to be calculated. Several curb characteristics are used for calculating cluster confidence. These characteristics are then combined into a single confidence value that is assigned to each

cluster. Each characteristic is denoted by a different confidence method, these methods are independent and can be configured separately to run based on the speed of the vehicle. The first method iterates through adjacent segments in each cluster and calculates the spatial difference between the segments. Since each cluster is sorted based on y-value, adjacent segments will be near each other spatially. For each adjacent segment, the difference between the lower y-value and larger y-value segments is taken. The difference is taken between the minimum x and minimum y of the larger y-value segment, subtracted by the maximum x and maximum y of the lower y-value segment. The difference is then compared to a threshold T_y and T_x , if the difference is below the thresholds, the confidence increases for that cluster. The second method looks at the number of segments in a cluster. Usually clusters that represent curbs will contain many segments. Therefore clusters that contain many segments obtain a higher confidence value. The third method uses the similarity metric described in **Section 6.6**. Adjacent segments within clusters are expected to have similar slope and intercept values. Small sections of curbs do not deviate greatly, and therefore a metric that compares the similarity between segments can be used. This method iterates over all the segments in each cluster and computes the similarity between adjacent segments. If the similarity is above a threshold T_s , then the confidence for that cluster increases.

The three methods described above are independent and measure the likelihood that the cluster represents a curb. Each method requires varying time complexity to complete, and therefore tradeoffs between quick execution time, and more accurate confidence can be made. To calculate the aggregate confidence for each cluster, the above confidence values are averaged and represented for each cluster.

6.7.2 Inter-Cluster Confidence

Each half of the point cloud (i.e. left or right side of the vehicle) is expected to detect one or more curbs. A confidence value must be given to indicate the likelihood that two or more clusters form a single curb. This is important since it allows for software later on in the ADAS stack to quickly determine the confidence of a curb detected in the point cloud. This confidence is an aggregate of the cluster confidence described in **Section 6.7.1**. Confidence is computed between close clusters since multiple clusters may represent a curb. Two methods for inter-cluster confidence were designed. The first method is an aggregate average of the confidence values for each cluster. This gives a quick snapshot for the point cloud to see whether a curb was detected with high confidence on either side of the vehicle. This method is quick to compute and gives good performance for confidence accuracy. The second method looks at adjacent clusters and computes the likelihood that the clusters form a continuous curb. The likelihood is computed threefold: First, the confidence values of each cluster are checked and averaged. Second, the difference between the cluster average y and x coordinates are taken, if the difference is below a threshold T , then the confidence of the point cloud increases. Third, the similarity metric is taken between the adjacent segments of both clusters. The confidence values are then averaged and displayed for each point cloud.

6.8 Real-Time Acceleration Methods

A Real-Time System is a computer system which is required to periodically process data within a critical time constraint. In the case of the HDL64E LiDAR sensor, new data is produced at a rate of 10Hz. In other words, every 100ms new LiDAR data is captured and must be processed before the next set of data is received. For safety purposes, it is pivotal that the curb detection algorithm execute from start to finish within this time constraint. Failing to do so could have serious repercussions, including the possibility that the vehicle makes a wrong decision and endangers the passengers.

Considering the sheer size of the LiDAR data, the curb detection algorithm must be optimized to process all of the points and generate a confident output within the time constraint. For this reason, real-time acceleration methods were investigated to reduce the computational time of each frame of LiDAR data. The methods outlined in this section operate on the basis of using results from previous snapshots to reduce the number of points that require processing. Given the time difference of 100ms between frames, these methods make use of the fact that, at regular velocities, changes in the vehicle's surrounding environment from frame to frame are quite small. For instance, if a curb is detected to the right of the vehicle in one frame, it is likely that in the next frame, the curb will still be in the same location. This use of prior knowledge significantly reduces the search area for curb points in the point cloud. Thus is the principle on which the Segment-Based Point Cloud Reduction method, described in **Section 6.8.1**, relies on to reduce the computation time for each snapshot. **Section 6.8.2** describes the Rear-Environment Recall Point Cloud Reduction method. In this method, only the half of the new LiDAR data that is in front of the vehicle is processed while the rear half is reused from the previous frame's point cloud.

6.8.1 Segment-Based Point Cloud Reduction

Using the segments reported in previous snapshots by the curb detection method, it is possible to speed up the next snapshot's execution time through point cloud reduction. Removing points from the point cloud is obviously a risky task since the car would presumably rely on the correct detection of a curb for either navigation, path planning, or environmental perception in general. This is why any method of point removal must be relatively conservative in that it should strive to only remove points that can confidently be presumed not to be the curb in the next snapshot. In the case of segment-based point removal, the detected curbs are used as a boundary within which no points may be removed. This should retain many of the curb points as well as the road. Additionally, curbs often exhibit changes in their positioning relative to the vehicle over time and space. Thus, the segment-based approach also checks for variance in the curb position, and adds an extra buffer zone to ensure no curb points are accidentally removed. This process is run on both the right and left sides of the vehicle, and the point removal on each side is independent.

Each segment reports numerous statistics about its position including: the linear equation, minimum and maximum x value, minimum and maximum y value. In the case of segment-based reduction, the minimum and maximum x values are of interest. These values allow for the maximum curb position on either side of the vehicle to be defined and tracked over time⁹. The maximums are illustrated in **Figure 6.13 a)**, where parallel, straight curbs, are on either side of the vehicle. The Δx_{left} and Δx_{right} describe the maximum boundaries of the curb. This is evident in **b)** and **c)** of the same figure. Using these maxima, the next snapshot can set a boundary of where the curb and road are likely to lie, and retain these points.

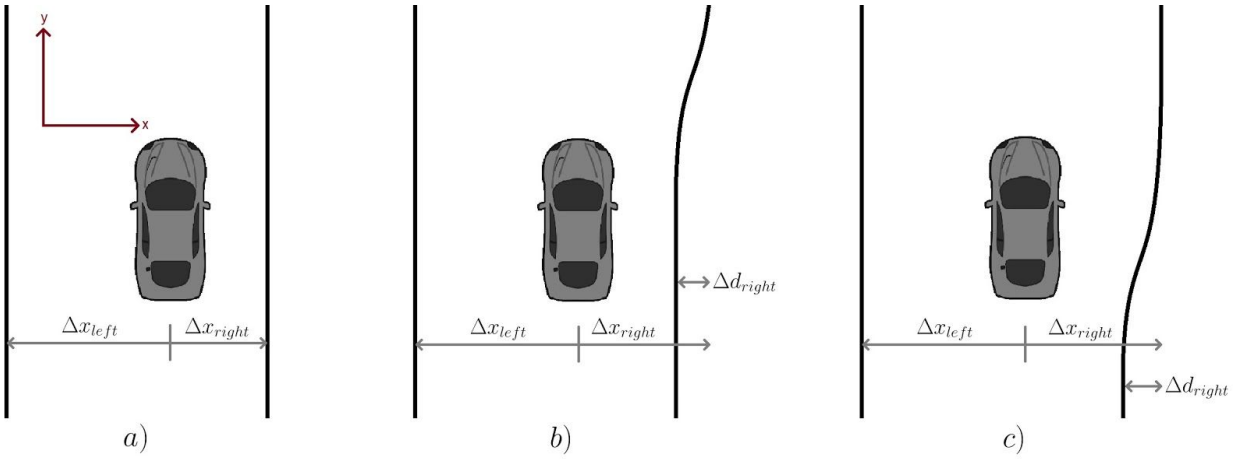


Figure 6.13: Exponential averaging example with bus lane over multiple snapshots

Figure 6.13 b) and **c)** illustrate a condition where, on the right side of the vehicle across multiple snapshots, a bus lane emerges. It is, however, not clear in **b)** that this divergence in the curb maximum is continuous, or ending. In case of the former, if the road boundary was set according to just the maximum x location in **a)**, then this would be entirely insufficient, and the curb may be completely lost in future snapshots. Thus, the rate of change of the maxima must also be taken into account when considering the reduction boundary. This rate of change was previously referred to as the variance of the curb maxima. In **b)** and **c)**, the variance is reflected by Δd_{right} , which is the difference between the maximum and minimum x positions in the detected segments on the right side. However, when setting a boundary, taking only the maximum and the variance and applying these to the next snapshot is insufficient and potentially dangerous. Thus, when setting boundaries, extra buffer space must be allowed to account for sharp changes in the curb, or other unexpected variations. Additionally, the difference in the maximum and minimum curb position is highly variant, so any changes in this difference should be averaged over multiple snapshots to avoid an overreaction or oscillations¹⁰.

⁹ Maximum curb position on the left is the smallest x value, maximum on the right is the largest x value.

¹⁰ Initial testing on a method using just the difference of maxima and minima for variance revealed an underdamped response, where the variance would oscillate about a central value.

In order to calculate the boundaries, three values are required: The maxima, the averaged deltas, and the buffer. Since these boundaries will affect future snapshots, and since curbs are often continuous (excluding intersections, and other gaps), it follows that the boundary calculations should be temporally continuous. In other words, the boundary should be set and informed by the current, and previous detections. One method that allows for such a calculation schema is exponential averaging. This allows for each of the variables to be averaged over time with a weighting so that old results minutely affect newer ones, and newer results more strongly affect the boundary location. The variables to be averaged are the maxima on both sides of the vehicle, and the variance on both sides of the vehicle. The general exponential averaging equation is shown below, where $g(t)$ is the current averaged value, $g(t-1)$ is the previously averaged value, $h(t)$ is the currently detected value, and α is the smoothing factor.

$$g(t) = \alpha h(t) + (1 - \alpha)g(t-1) \quad (11)$$

The optimum value for the smoothing factor was experimentally found to be 0.4 for both the variance and maxima. When calculating the boundary, the exponential average for the maxima, $x_{max}(t)$, and the exponential average for the variance, $v(t)$, are summed. A buffer is also added to the variance to ensure that any sudden changes of the curb are accounted for. The buffer is exponentially averaged along with the curb variance. The buffer's value was experimentally found to be 2 meters¹¹. The boundary, $b(t)$, is calculated according to the equations below.

$$x_{max}(t) = \alpha \Delta x_{right} + (1 - \alpha)x_{max}(t-1) \quad (12)$$

$$v(t) = \alpha(x_{max}(t) - x_{max}(t-1)) + (1 - \alpha)v(t-1) + 2 \quad (13)$$

$$b(t) = x_{max}(t) + v(t) \quad (14)$$

When segment-based point cloud reduction is used, it is necessary to provide at least one snapshot prior to the calculation of a boundary. This first snapshot is used to calculate the initial variance and maxima which will then be used to set the boundary in the next snapshot. In **Figure 6.14 a)**, the boundary is calculated for each snapshot for a simulated bus lane according to the example of **Figure 6.32**. After an initial adjustment, the boundary adapts to the curb location, and is able to take into account the bus lane with no issue. Similarly, **Figure 6.14 b)** illustrates the boundary conforming to the initial curb; however, an issue arises when a large change in the curb location occurs. The red circle in **b)** indicates an error in the boundary. Since the boundary is set to around 6m, the new curb location of 9m is not captured. In this case, since no curb was detected, the boundary calculation restarts, and the entire snapshot is analyzed in the next time frame. This allows for errors from rapidly changing curbs to be accounted for so that the car can recover quickly. After the 10th iteration, another large change in the curb location is introduced, however, since this change is gradual, occurring over multiple

¹¹ Since $\alpha = 0.4$, and the buffer is 2m, given that the variance and maxima are added over subsequent iterations, the buffer will be realized as a 2m buffer plus the geometric expansion of original 2m buffer multiplied by the smoothing factor. This leads to a total buffer over multiple snapshots with an unchanging curb of up to 5m. This is given by the following: $(2m) + (2m) * (0.6 / (1 - 0.6))$.

snapshots, the boundary is able to keep up. It is worth noting that these huge changes in curb location are unexpected in real-world driving environments, however, these error conditions must still be addressed to ensure safety and reliability.

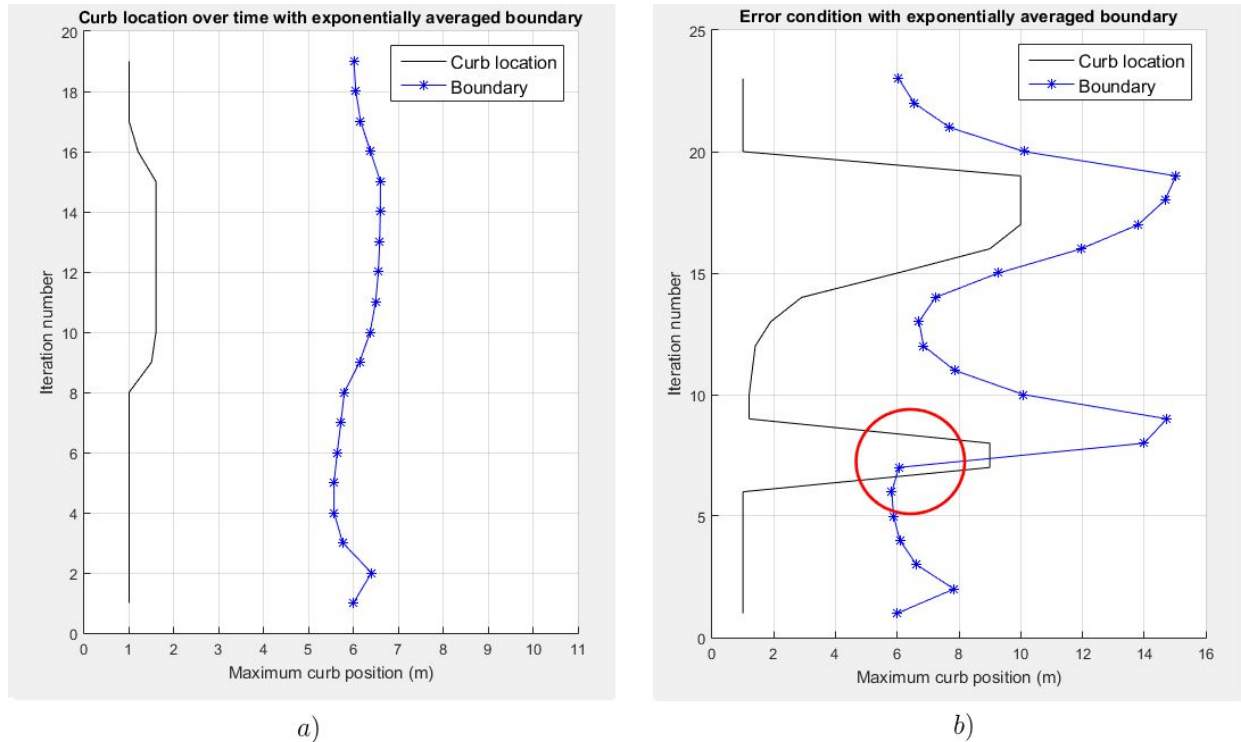


Figure 6.14: Graphs of the exponentially averaged boundary and the real curb location

This method of point cloud reduction works well in real-world scenarios. In the test data provided by QNX, the method was able to keep up with all changes in curb location without any errors like illustrated in **Figure 6.14 b)**. This is most likely due to the continuous nature of curbs, and the low likelihood of extreme discontinuities. Speedup for segment-based reduction is presented in **Section 6.8.3**. Due to the relatively large buffer zones, the speedup is marginal, but for a real-time system, may be necessary to meet deadlines.

6.8.2 Rear-Environment Recall Point Cloud Reduction

The Rear-Environment Recall Point Cloud Reduction method takes advantage of the short time lapse between snapshots combined with the vehicle's movement to reduce the computation time per snapshot. In this method, the vehicle's Inertial Measurement Unit (IMU) is used in conjunction with the LiDAR sensor. The IMU data provides the necessary information for calculating the exact translation of the vehicle between frames of the LiDAR data. It is hence possible to reuse already processed LiDAR data from the previous frame by adjusting for the translation and rotation of the vehicle. It is worth noting that this method is only enabled when the vehicle is moving as there is no purpose in real-time acceleration for curb detection when

the vehicle is stationary. **Figure 6.15** shows how the Rear-Environment Recall method reuses the previous LiDAR frame to reduce the number of points to be processed in the second frame.

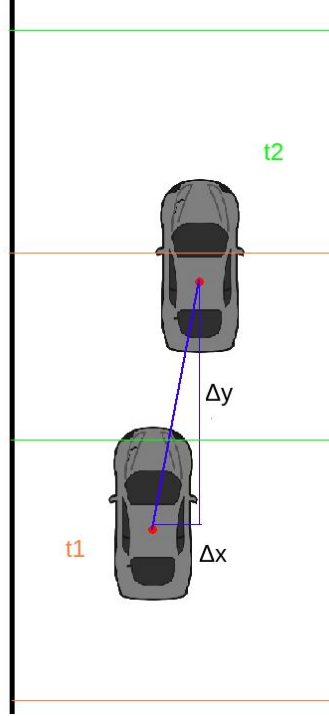


Figure 6.15: Frame to frame vehicle translation

In **Figure 6.15**, the first LiDAR frame, t_1 , is delimited by the orange lines while the second LiDAR frame, t_2 , is delimited by the green lines. The region where the frame 1 and frame 2 LiDAR point clouds overlap is the subset of points that are reusable for frame 2. When frame 2 is received from the HDL64E LiDAR, the environment in that subset is already known and processed from frame 1. There is no need to reprocess these points. The vehicle's position will however have changed by some Δx and Δy values thus requiring the already processed points in frame 1 to be translated to align with the point cloud in frame 2. These values can be calculated using the following kinematics equations:

$$\Delta x = v_x * t + \frac{1}{2} a_x * t^2 \quad (15)$$

$$\Delta y = v_y * t + \frac{1}{2} a_y * t^2 \quad (16)$$

where v_x , and a_x are the velocity and acceleration in the x direction, v_y and a_y are the velocity and acceleration in the y direction, all of which are given by the IMU; and t is the time lapse between frames equal to 100ms. By adding these Δx and Δy values to the x and y values of each point in the overlapping region, the points are aligned with the point cloud in frame 2. When reusing LiDAR points from memory, it is also necessary to consider potential rotation of

the vehicle between frames as well. **Figure 6.16** shows how the vehicle may rotate between frames.

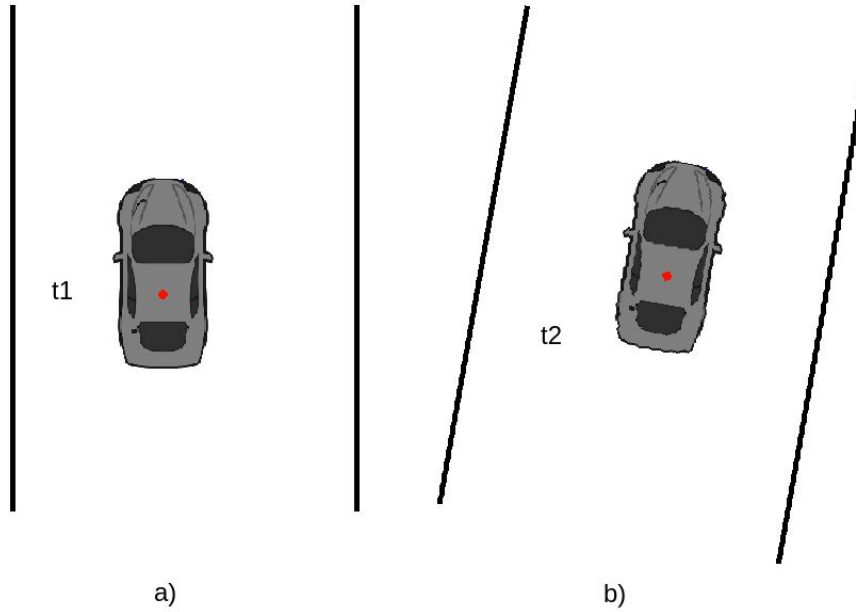


Figure 6.16: Before and after rotation of the vehicle. In a) the vehicle faces directly forward. In b), the vehicle has rotated -10° along the yaw axis.

In the case of **Figure 6.16**, the vehicle has rotated by an angle of -10° along the yaw axis between frame 1 and frame 2. For the points in frame 1 to be reused during the frame 2 computation, all overlapping points between the two frames must also be rotated by the same angle. This is achieved by applying the following rotation matrix to all these points:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (17)$$

where x' and y' represent the new x and y value for each point after the rotation, and θ is the angle of rotation of the vehicle along the yaw axis. Once the rotation and translation of points in the overlapping region between frames have been accounted for, the points become aligned with the point cloud of the second frame. Since these points have already been processed and are all situated behind the vehicle in frame 2, all the points behind the vehicle in frame 2's LiDAR point cloud can be removed and replaced with the already processed LiDAR points. This reduces the size of the point cloud behind the vehicle from approximately half of the total point cloud to a few hundred curb points that were detected in the previous frame. The computational burden is thus significantly decreased, since the curb detection method only needs to process the points in front of the vehicle rather than the entire point cloud. As will be discussed in the next section, the Rear-Environment Recall Point Cloud Reduction method has yielded desirable results.

6.8.3 Speedup Results of Real-Time Acceleration

The two real-time acceleration methods outlined above were developed and tested in separate environments: The Segment-Based method was developed and tested in Python while the Rear-Environment Recall method was developed and tested in C. Both methods were run in Linux, and the C environment used a QNX virtual machine. Segment-Based reduction was run on an 8th generation i5 processor, while the Rear-Environment reduction was run on an 8th generation i7 processor. Since C is a compiled language, and Python is an interpreted language, it is expected that the computation time per snapshot will be significantly faster in C. Given the differences in environment, it is also useful to outline the method of testing used for each speedup scheme. The segment-based scheme was tested by passing sets of 4 consecutive snapshots to the detection algorithm with point reduction either turned on or off. Given that the first snapshot can not be used for speedup calculations, it was ignored when calculating average runtimes for the results. To get the speedup, the average runtime of each snapshot was taken across multiple trials for both the full-sized and reduced snapshots. The speedup was then calculated according to the following equation, where t_{off} is the runtime without acceleration, and t_{on} is the runtime with acceleration.

$$Speedup = \frac{t_{off} - t_{on}}{t_{on}} \cdot 100\% \quad (18)$$

Rear-environment recall was tested in much the same way, however, instead of running on a set of 4 consecutive snapshots, it was run on multiple different sets of consecutive snapshots. Rear-environment recall was run a total of 500 times, and segment-based reduction was run for 40 trials. **Table 6.1** shows the speedup results of both real-time acceleration methods:

Table 6.1: Real-Time Detection Acceleration Results

Method of Real-Time Acceleration	Average time to detect curbs without speedup	Average time to detect curbs with speedup	Speedup Difference	Speedup Percentage
Segment-Based Point Cloud Reduction	4.837s	3.945s	0.892s	22.61%
Rear Environment Recall Point Cloud Reduction	35.02ms	14.71ms	20.31ms	138.07%

Considering that the execution time is highly variable and dependent on the computer environment, it is important to note that the times of detection are not as significant as the speedup difference and the speedup percentages. The Rear-Environment Recall and Segment-Based methods effectively reduced the computation time of a single snapshot, on

average, by 138% and 22% respectively. This result is desirable as it leaves the curb detection algorithm almost 80ms of leeway. The curb detection method could thus be expended for sensor fusion and decision-based detection given the extra execution time available.

Chapter 7: Road Detection and Corroboration of Curbs

The method in [22] was implemented as an independent method, however there is a benefit in corroborating this method with another which focuses strictly on curb detection. By performing both methods on the same environment, it is expected to see similarity between the edges of the detected road surface and the outcome of the curb detection method.

7.1 Road Detection

One method of finding the road's edges is by detecting the drivable space and then classifying the edges of this space as the edges of the road. To detect the drivable space on the road, adjacent lasers from the LiDAR are examined. Each laser sampled from the LiDAR sensor will return the distance (radius) that the laser travelled before hitting and being reflected by an object in the environment. As the LiDAR sensor's lasers scan the road, as long as the road is flat with minimal deviation, the radius of each laser should remain constant. When the lasers reach the edge of the road and encounter a curb or gravel edge, the radius changes and no longer remains constant. This creates an inflection point in the data. Due to the fact that the radius of one laser will be relatively constant along the road, we expect the difference between the radii of two adjacent lasers to be relatively constant along the road as well. When examining the difference between the radii of two adjacent lasers, the inflection points at the road edges are amplified due to the significant variance. This allows easier detection of these inflection points and subsequently, the road edges. Using this method iteratively over every pair of adjacent lasers allows the extraction of the road edges along the path of the vehicle. The space between the road edges is classified as the drivable road [42].

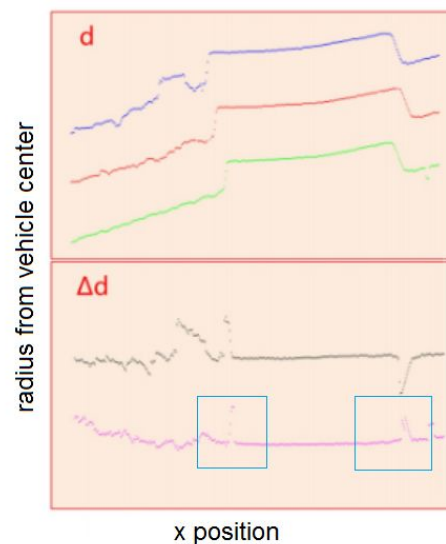


Figure 7.1: Adjacent Laser Radii and their difference [42]

The top half of **Figure 7.1** showcases the changing radii, d , of 3 adjacent lasers as they scan across the road. The constant portion of the data in the middle of the plot is where the road was being scanned, and the portion where the data starts to change rapidly represents the road edges. Shown in the bottom half on **Figure 7.1** are two lines; the black line representing the difference in radius between the blue and red line, and the pink line representing the difference in radius between the red and green line. Taking the difference between two adjacent lasers gives a more flat and constant representation of the road portion, while accentuating the variance of the road edges.

Inflection points are simply a point on a curve at which the concavity changes sign (i.e. from concave to convex or the opposite). Two example inflection points are marked by blue boxes in **Figure 7.1**. In this case, an inflection point represents a change in the road surface which indicates its edge. The data being examined is the difference in radii between two adjacent lasers. The difference shows the constant delta between each laser's radius while on the road and then shows the drastic change in that delta once the road edge is encountered. There are multiple ways to detect inflection points along a curve. **Sections 7.1.1** and **7.1.2** explain two different methods which were used. An issue with this method of road detection is the fact that inflection points don't solely come from curbs or road edges. Any disturbance or object on the road can create an inflection point. This necessitates some way to determine which two inflection points represent the actual road edges. This can be done using a loss function. By calculating the loss of each pair of inflection points, the pair which is most probably to represent the actual road edges can be found. This is done by choosing the pair of inflection points which has the lowest loss. Further details of the loss function are described in **Section 7.1.1** along with a method to find inflection points using surface variation. An alternative method for calculating the inflection points using the averaged difference in radii is described in **7.1.2**.

7.1.1 Surface Variation Approach

One approach to finding inflection points is using surface variation. This method is similar to that of the road surface filter. One main difference however is that the road surface filter is performed on 3D LiDAR data where this method is used on 2D data. To find the inflection points, this method uses a sliding window approach. A specified number of points are examined in each iteration and whether they are classified as an inflection point is decided by the surface variation value. If it lies above a specified threshold value, then it is classified as an inflection point. For each iteration of the sliding window, the 2D covariance matrix is calculated. From this, the two eigenvalues are also calculated. Then the surface variation is calculated using equation 19.

$$\frac{\lambda_0}{\lambda_0 + \lambda_1} \quad (19)$$

Any portion of the difference in radii data which has a surface variation value above the threshold is classified as an inflection point, meaning there could be more than 2 inflection

points found in the data even though there are only 2 edges to the drivable road. These additional inflection points are due to objects or other variations in the road. As mentioned in **Section 7.1**, a loss function is used to find the pair of inflection points which is most probable to be the actual edges of the road.

The loss function used is shown in equation 20, where n and m are the pair of inflection points, Δd_i is the difference in radii between the two adjacent lasers at index i , and $\overline{\Delta d}$ is the average difference in radii between the two adjacent lasers. The loss function promotes inflection point pairs which are far apart and have little variation in between them.

$$\mathcal{L} = \frac{1}{n - m} \sum_{i=m}^n \|\Delta d_i - \overline{\Delta d}\| \quad (20)$$

Once all inflection points are found, and the loss function calculates the loss for each inflection point pair, then the pair with the lowest loss is selected and classified as the road edges.

7.1.2 Averaged Difference of Radii Approach

The previously mentioned road detection algorithm is inherently complex and computationally intensive when determining inflection points. Experimentation was done on alternative methods of determining inflection points which could simplify computation.

The averaged difference of radii approach is a simplified version of the inflection point detection algorithm which involves using mathematical averaging to determine the prominent inflection points. This method is designed to ideally detect solely the inflection points nearest to the left and right of the vehicle's path. These inflection points should correspond to the curbs or to any vehicle that is along the path of the laser. After the difference in radii, Δd , is calculated, an assumption is made that there will be two inflection points along the road; one to the left of the vehicle, and one to the right of the vehicle. The x component of the direction in which the vehicle is moving is calculated to narrow down the boundary for where inflection points could exist. This is used since there will typically be curbs to the left and right of the vehicle. If the vehicle is travelling parallel to the two curbs, then this center would be immediately between the two curbs.

The method consists of traversing Δd values starting from the center and traversing to each side of the vehicle until the inflection points are found. Referring to the equation below, groups of four Δd values are averaged and then compared to their preceding group of four Δd values. This process is performed iteratively for each grouping of four Δd values until the inflection points are detected. The value four was chosen as it rendered the best result for change in Δd .

$$\Delta d = \frac{\Delta d_i + \Delta d_{i+1} + \Delta d_{i+2} + \Delta d_{i+3}}{4} \quad (21)$$

Referring to the equation below, If the difference in these averaged values is greater than a threshold, an inflection point exists between the two values.

$$\Delta d_{diff} = \frac{\Delta d_{av(j)} - \Delta d_{av(j-1)}}{2} > Threshold \quad (22)$$

The threshold is adjusted to account for noise along the road, but it is assumed that there are no disturbances in the road that would cause false positive inflection points. Once the algorithm reaches a curb, there will be a significant change in Δd averages and an inflection point will be detected. This inflection point is then classified as a potential curb. The same process occurs on the other side of the vehicle.

To validate that the inflection points are correct, the detected inflection point could be compared to the adjacent pair of lasers' inflection points as an improvement. This will ensure continuity. If continuity is present, then it is more probable that the inflection points correspond to a curb, or a distinct straight lined obstacle. This method returns a set of (x, y) points that correspond to the locations of the curbs for each set of adjacent laser scans.

7.2 Results on Road Detection

Both the surface variation approach and the averaged difference in radii approach yielded inflection points that corresponded to the curbs. The surface variation approach detected all of the inflection points and then returned the pair with the lowest loss as being curb points. The averaged difference in radii approach detected solely the pair of points corresponding to the curb. Referring to **Figure 7.2** below, the red portion corresponds to the detected road surface between each detected inflection point pair. The blue portion corresponds to the rest of the environment. Both of these methods detected the inflection point pair and corroborated each other.

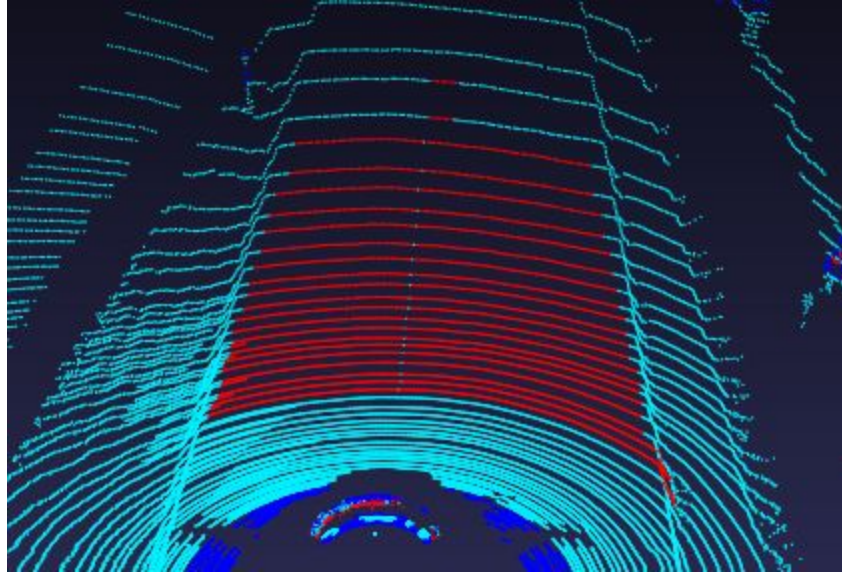


Figure 7.2: Road surface detection using inflection points

Both methods were able to detect the road surface well but they suffered when adjacent lasers were spread further apart. This caused the delta in the radii to have more variation which would yield false positives for inflection points. The road surface method is most applicable to detection closer to the vehicle, which from testing was within 15-20 metres in front of the vehicle.

Chapter 8: Combined Methods and Results

The methods described in **Chapter 6** and **7** are two unique approaches for curb and road detection using LiDAR data. Both methods are independent and can be run in parallel. As described in **Section 6.7**, confidence values are added to detected curbs that measure the certainty of the curb detection method. The confidence values are based on the characteristics of the detected curb from a single method. Basing confidence of detected curbs on two independent methods improves reliability and robustness of curb detection. **Section 8.1** describes the corroboration of the two curb detection methods described above. In addition, further curb detection results are discussed in **Section 8.2**. Finally, detection on sparse point clouds will be discussed in **Section 8.3**.

8.1 Assigning Confidence to Corroborated Detection

To further improve the reliability of curb detection, the curb and road detection methods described above are combined into a single corroborated method. To achieve this, the road detection method is used to further give confidence for the curb segments detected by the curb detection method. To recall, the curb detection method outputs several curb clusters that contain curb segments. Each curb segment is denoted by a linear equation, the average x and y values for the segments, the standard deviations of x and y, and the maximum/minimum x and y values. The road detection method returns a set of inflection points that correspond to the locations of curbs. Each inflection point is measured by its x and y coordinate value. Therefore, an inflection point found using the road detection method can be compared with the curb segments.

The x and y coordinates of inflection points are compared to the maximum and minimum x and y values of each curb segment. If the inflection point falls within a curb segment, then that curb segment's confidence increases greatly. Curb segments that contain an inflection point are more likely to represent curbs since two independent detection methods resulted in the same detection. However, inflection points that do not fall within any curb segments are discarded. Due to the segment-based curb result representations, inflection points cannot be represented independently and must be corroborated with curb segments. Curb segments that contain no inflection points are still represented, albeit with lower confidence. The corroborated results give more accurate and reliable curb detection, since two independent methods outputted similar results. However, if both methods detect the same false positives, then these false positives will be represented with high confidence. This is uncommon, since both methods have different approaches for detection.

8.2 Results with HDL64E

The methods developed in this paper prove to be effective at detecting and extracting curbs from LiDAR point clouds. **Figure 8.1** below showcases the detection of the corroborated methods.

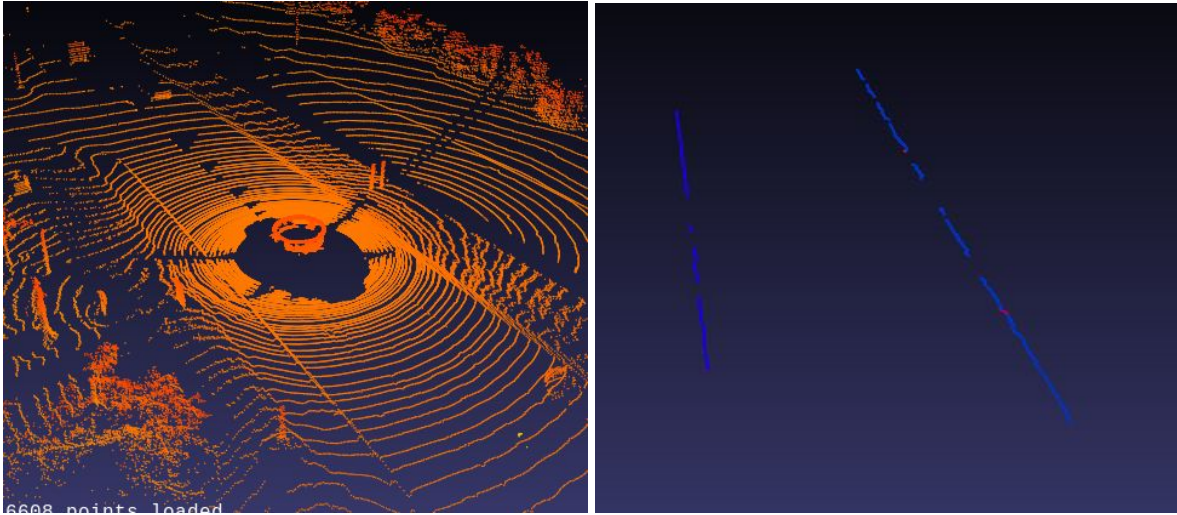
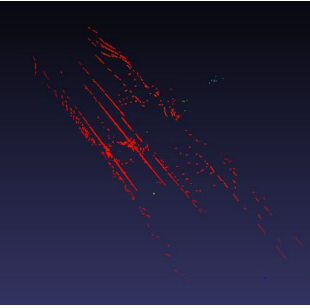
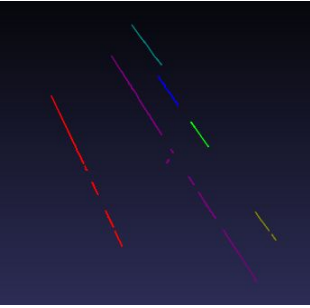
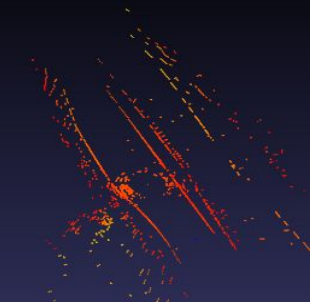


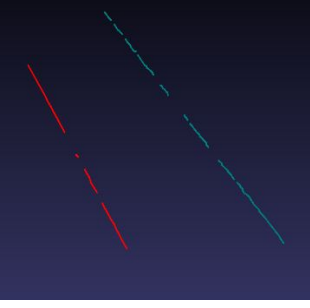
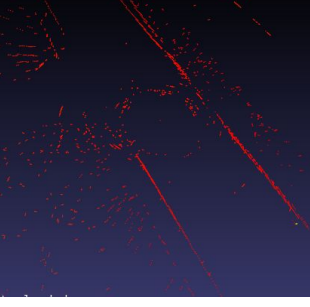



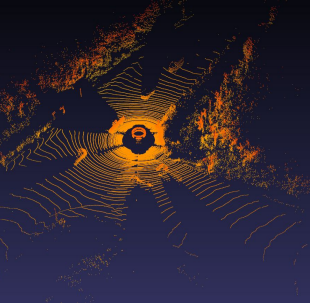

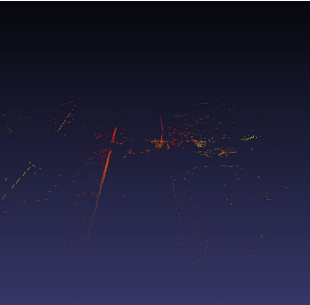
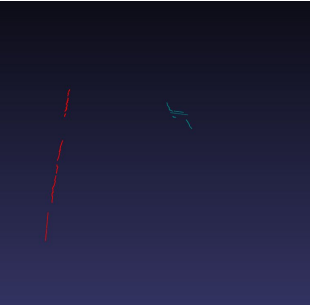
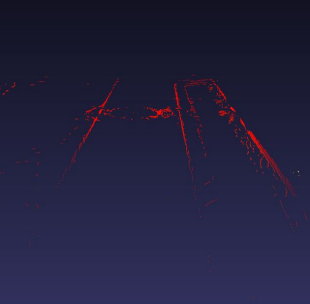
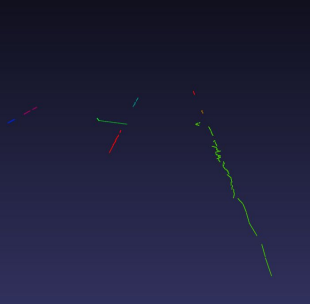
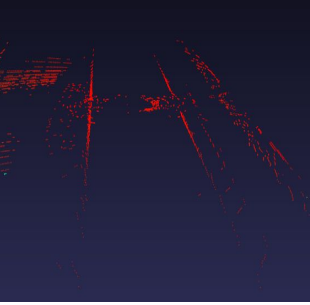
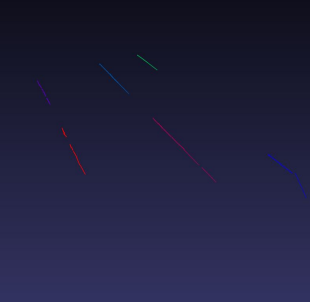
Figure 8.1: Raw point cloud (left) vs. detected curbs (right)

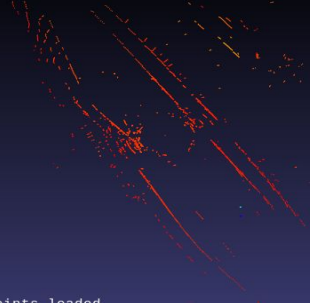

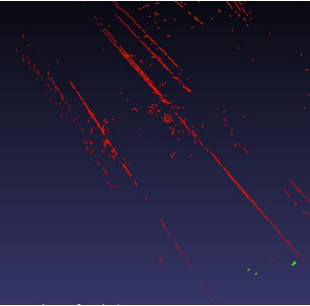
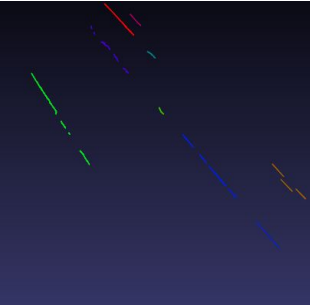
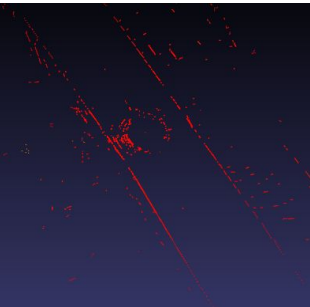
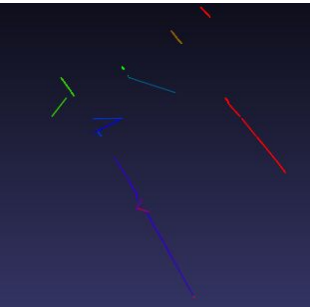
Figure 8.1 showcases accurate curb detection with no false positives. As shown from the detection, all points other than curb points are filtered out. The curbs detected correspond to the full length of both curbs seen in the raw point cloud. This detection shows that the corroborated method is resilient to false positives and noise, while also maintaining strong curb resolution.

The following table showcases results of the curb detection method on real world LiDAR snapshots. Each snapshot was extracted from a HDL64E LiDAR sensor. The table showcases images of snapshots after the filtering step, and after the full curb detection method. Curb clusters are coloured in the curb snapshot. The table also displays confidence values for both the right and left curbs. These confidence values are an aggregate score of the multiple confidence methods described in **Section 6.7** and **Section 8.1**. Finally, comments are showcased for each snapshot.

Table 8.1: Curb detection method results using HDL64E LiDAR sensor

Snapshot after filtering step	Curb Snapshot	Right curb confidence	Left curb confidence	Comments
		1	0.8	Curbs are detected with a few false positive detections.
		0.9	0.79	Curvature in the curb is detected. Some false positives detections, however false positive detections are behind the curb.
		1.0	0.788	Curbs are detected with no false positives. Noise on the curbs reduces confidence value.
		1.0	0.93	Intersection cuts off the left curb. Right curb is detected and the left curb before the intersection is detected.

		0.76	0.78	The car is turning into a new roadway. The right curb is not visible, and the left curb is blocked by cars. The left curb is correctly detected, but the right curb contains 2 correct segments and 2 false positives.
		1	1	Lack of curb on the right due to an intersection causes a small number of segments to be detected for the right curb. Confidence is high because the curbs detected showcase strong curb characteristics.
		0.77	0.77	Noise and remnants from the similarity filter lead to false positives detected next to the car on the left curb.
		0.76	0.80	False positives detected outside of road area on right and left side curbs.

		0.84	0.78	Noise on the left curb makes it difficult to detect the curb beside the vehicle. Right curb is detected well, however some of the curb is not detected due to noise.
		1.0	0.79	Right curb is detected fully with some false positives. Left curb is sparse and therefore not fully detected.
		0.48	0.6	Noise from ground points reduces detection confidence by introducing false positives on the road. Detection of curbs themselves are good.

From **Table 8.1**, several conclusions are made on the effectiveness of the curb detection method. First, the curb detection method detects that there are curbs on both sides of the vehicle for each snapshot. Second, as seen by some snapshots, curb resolution is not always maintained and some parts of the curb do not get detected. The false negative rate for curb detection is high in some snapshots, which raises safety concerns. Snapshots are received and analyzed every 100 ms, and therefore new data comes in of the surrounding environment. It is more likely that the curb will be detected fully over multiple snapshots. Third, false positives are also apparent in the detections. These false positives are not curbs, however the curb detection method classifies them as curbs. Mostly, false positives are detected behind the curbs and not on the road. This means that false positives do not cause hazards to the detection. Fourth, noise plays a prevalent part in the success of curb detection. It is seen in several snapshots that the curb detection method does not detect curbs with high noise to signal ratio.

8.3 Results on Sparse Point Clouds

Sparse point clouds provide a unique testing environment for curb detection methods. Point resolution on curbs and the surrounding environment is decreased, making it harder for the detection methods to extract and correctly identify curbs. Randomly removing points can be seen as a method of stress testing, since it decreases the accuracy of the model, and may even lead to a breaking point. These breaking points are important and useful to know. Testing the combined methods on sparse point clouds also allows for a measure of how effective detection is in high noise environments. It is expected that high noise environments (e.g. heavy snow / rain), will produce many scattered noise points which will be removed with preprocessing or the filters. By reducing the point cloud density before these steps, it is possible to test the entire method as if it were experiencing high noise. To create a sparse point cloud, a random number generator is initialized with a seed such as the current date and time. Then, indices are randomly selected from each laser line and the corresponding point is removed. This ensures that each laser line will see a reduction of points of the same magnitude in random positions. This is repeated for each reduction percentage. Each of these reduced point clouds is input to the curb detection method, and the results are recorded.

The testing process with sparse point clouds revealed some interesting results. To ensure the detected curbs were correctly identified, two labelled point clouds were used, and the similarity metric (see **Section 6.6**) was employed to give a measure of the accuracy of the detection. The results, averaged over multiple executions of both labelled snapshots, are illustrated in **Figures 8.2, 8.3, and 8.4**. It is important to note that three breaking points were identified during testing: 1) After about 40% point removal, the curbs start becoming more sparse, and miss-detections become common; 2) After 50% point removal, miss-detections become very common, but are essentially random and consist of few segments; and 3) 90% point removal reveals a total breaking point where no segments can be correctly identified. It is therefore recommended that the curb detection method is not used in any scenarios with greater than 30% reduction¹². In **Figure 8.2**, the average cluster size and the largest cluster size (as in the number of segments per cluster) is plotted against the removal percentage. It is evident that the clusters become consistently smaller over increased sparsity, but it isn't immediately clear why the average cluster size remains about the same. The latter case occurs because with low removal percentages, noise is still present to a high degree. The segmentation methods struggle to remove noisy clusters and small clusters that don't contribute to or corroborate detection. For example, in the 0% removal case, the largest cluster is often in the range of 10-15 segments, but there are many small clusters of size one or two, that contribute to a lower average cluster size. A problem with these small clusters is that they can often be highly variant, and susceptible to noise. This means that small clusters are expected to decrease accuracy overall. This is seen in the comparison between **Figure 8.3**, and **Figure 8.4**. The largest clusters often exhibit the best detection results, and are less susceptible to noise. As shown in **Figure 8.3**, the

¹² It is, of course, hard to identify situations where random point removal will occur in the real-world, but that said, this is a warning for high-noise usage of LiDAR-based detection.

largest clusters have high similarity with the known curb position (> 93% similarity for all between 0 and 30% removal). **Figure 8.4**, however, shows the results with all clusters taken into account (with the exception of the clusters of size one). It is evident here that the average similarity has decreased by over 10%, meaning some of the clusters are in fact not detecting the curb at all.

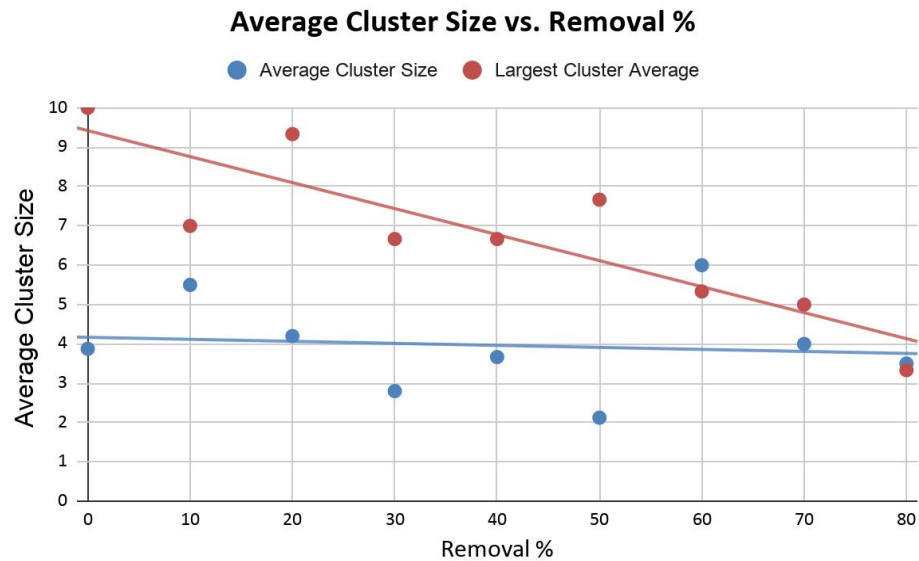


Figure 8.2: Cluster size versus removal percentage for sparsity testing

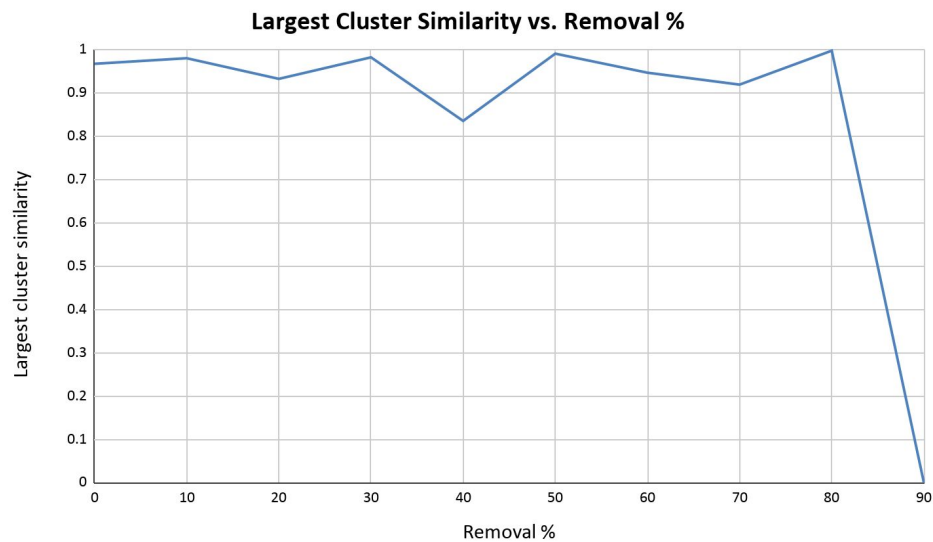


Figure 8.3: Similarity of largest cluster in detection to known curb versus removal percentage for sparsity testing



Figure 8.4: Average Similarity of all detected clusters in detection to known curb versus removal percentage for sparsity testing

The results presented here indicate a number of flaws with the detection method as it currently stands. Firstly, the segmentation technique, while novel and effective for curved and straight roads alike, is difficult to test. The labelled curbs used in the above tests were straight, uninterrupted curbs that had a straight line fitted via linear regression, using points that were hand picked. It is not clear at all how this method could be tested on curved roads, much less interrupted road sections, or bus lanes and other non-uniform road changes. The segmentation technique also suffers from false positives in a way that other methods of detection do not, namely, the segment grouping technique presented in **Section 6.6** does not effectively remove non-curb segments. This could, had it been implemented on time, be remedied by inter-cluster confidence, but fundamentally the errors are still present and need to be addressed with more in-depth research. Secondly, the confidence metrics presented are not effective at producing useful self-referential confidence levels. For example, some of the results omitted from this section include the various confidence levels against removal percentages. These results were omitted because, despite some segments incorrectly identifying curbs as perpendicular to the roadway and actual curb, the cluster was given 100% confidence. Similar results to these were identified in all levels of removal, and are concerning to say the least. Finally, the first few steps of the curb detection algorithm seem to allow too many points on the road. This is due to the augmented similarity filter of **Section 6.2.2**, which uses PCA to remove points within a few meters of the vehicle instead of the similarity filter. This causes quite a few errors, and noticeable faults with segmentation. These defects were all caught through the analysis of the method on sparse point clouds. It is clear however, that improvements must be made, and the presented curb detection method is incomplete.

Chapter 9: Future Work and Research Possibilities

The methods described thus far have proven effective at detecting curbs and corroborating detection. New methods have been introduced which add great value to curb detection and environmental perception. Detected curbs have been modeled using segments, which allow for a more accurate curb model and the reporting of instantaneous changes. Algorithmic real-time speedup methods were introduced which allow for faster execution of a single snapshot given the detection results of prior snapshots. Corroboration methods allowed for higher confidence levels and safer curb modelling. Finally, the curb detection method was shown to be effective with the HDL64E on full and sparse point clouds. These achievements have enabled the presented techniques to have high-confidence detections, and low false positive rates.

The methods and techniques presented have been developed with accuracy in mind based on the tools currently available. Improvements, however, can still be made. For example, the aforementioned real-time speedup methods are effective insofar as they provide algorithmic improvements, however, these methods don't speed up the processes of curb detection, they just remove points so that the workload for a given snapshot is reduced. Given that the curb detection method would need to work in a real-time embedded system with deadlines, and that the worst-case computational complexity must be assumed to accomplish the deadline, work must be put in to reduce the runtime with other methods. Additionally, the methods presented for clustering are incredibly effective for non-uniform point density, however, they can be improved to incorporate the expected features of a curb and thus allow the removal of some steps of the method such as denoising with PCA. In addition, machine learning, despite its flaws, should be explored as a detection or corroboration method. Finally, sensor fusion should be examined for higher accuracy detections, and more effective corroboration.

9.1 Real-Time Speedup

The real-time speedup methods presented in **Section 6.8** explore point reduction as a technique to lower runtimes due to reduced workloads. These methods are effective insofar as they have adequate information, i.e. previous snapshots' detection, to inform the current curb position. This means that, in the worst case, a curb detection algorithm will still have to run without speedup due to sufficiently changing road conditions, as explored in **Section 6.8.1**, such as large variations in the curb position. This worst-case scenario must be considered the norm when considering the use of curb detection with real-time deadlines in a safety-critical system like a car. The results presented in **Table 6.1** in **Section 6.8.3** were obtained using powerful desktop processors. The execution time of each method was on average 4.837 s and 35.02 ms for segment-based and rear-environment recall respectively (albeit running in different environments). It is not clear how these execution times would scale to a much less powerful embedded microcontroller, much less how these methods would work independent of any virtual machines or OS-dependant interruptions like context switching. Thus, two methods are

proposed to increase execution efficiency, without decreasing just the workload. These are multithreading, and hardware acceleration with edge computing.

9.1.1 Multithreading Speedup

The speedup results presented in **Section 6.8.3** were quoted based on a single thread of execution on powerful desktop machines. Given this, the runtime results are actually quite impressive: a snapshot can be partially processed in 35ms in a C-based environment, which itself is running in a Virtual Machine on top of the Operating System. The data reduction techniques alone achieved large speedups but, given that many of the steps of the algorithm can be executed separately or in distinct chunks, a case can be made for employing multithreading to reduce runtime. Each step of the curb detection algorithm, other than the clustering steps (clustering and segment grouping), can be individually parallelized. For example, the preprocessing step can be split into a number of subsections and each subsection can be iterated over to remove duplicates and employ the height filter. Then, for the similarity filter, a point cloud can be split into subsections and each segment can be iterated over to remove non-similar points according to the criteria set therein. Clustering, however, forms a sort of bottleneck which requires reconstruction of the entire point cloud into a vector form. Clustering can not be multithreaded due to its inherent requirement to access the entire point cloud, thus, this section must be executed sequentially. The steps after clustering, excluding segment grouping, are parallelizable like preprocessing and filtering.

The ideal speedup of a concurrent system with parallelizable and sequential sections is given by Amdahl's Law. Speedup increases with the number of concurrent tasks up to a certain maximum which is governed by the amount of sequential code. The speedup is given in the equation below, where f is the proportion of parallelized code, and N is the number of concurrent tasks [43].

$$S(f, N) = \frac{1}{(1-f) + \frac{f}{N}} \quad (23)$$

The speedup, however, is never ideal. Real-world factors such as congestion, memory access times, communication between threads and so on; affect the speedup as well. These tend to decrease the speedup with a greater number of concurrent tasks, and therefore the number of tasks should be carefully considered. Additionally, given that the curb detection algorithm is supposed to run on a car computer system, it is unlikely that a large number of threads would be available. That said, based on the computer systems observed at QNX, it can be expected that 4 threads could be running in parallel on a quad core system. The sequential portion of the algorithm can also be estimated, at a high estimate, at around one third of the runtime (clustering is afterall an intensive task). So, the speedup can be estimated at around $S(0.666, 4) = 2$. This speedup is certainly encouraging, and would merit testing of the really achievable multithreading speedup.

9.1.2 Hardware Acceleration

The architecture of the computer system used by QNX has all sensor fusion tasks running on a software stack on a central microcontroller. This architecture, while fine for small processing tasks such as processing data from an ultrasonic or RADAR sensor, is ultimately not suited for the heavy processing required for a large LiDAR point cloud. The execution time for a full snapshot should be evidence enough of this: a single snapshot takes 35ms to execute out of a potential real-time window of 100ms for the LiDAR alone. This however discounts the requirement of other sensor interfaces to run, for camera data to be processed, etc. Not to mention that if a camera were operating at 30 frames per second, and the LiDAR data were to be fused with the image, then the processing time for a single snapshot would have to be under 33ms if not less. While multithreading certainly seems promising, a more direct approach may in fact be required depending on the cost, performance, and power requirements of a solution.

Hardware acceleration using a Graphics Processing Unit (GPU), Field-Programmable Gate Array (FPGA) or Application-Specific Integrated Circuit (ASIC) would allow for high-performance computing of the curb detection algorithm and other sensor fusion tasks. In [44], multiple hardware architectures for autonomous vehicles are introduced. These systems use a selection of ASICs, FPGAs, CPUs and GPUs “to deliver enough computing power, redundancy, and security so as to guarantee the safety of autonomous vehicles” [44]. While autonomous driving is not strictly the purpose of this project, it does relate to ADAS insofar as many of the tasks are the same, just performed at different levels of autonomy. Hardware acceleration could be used in an ADAS to provide the necessary speedup for heavy processing tasks such as LiDAR, or vision processing. Additionally, hardware designs can take advantage of pipelining and parallel execution units to allow for concurrent task execution. This would allow for LiDAR processing much faster than could be provided by a central microcontroller. However, two factors reduce the utility of a hardware-accelerated solution: cost and complexity. If a hardware system costs too much or if the processing can be accomplished with a cheaper solution in software, then manufacturers will likely go with what’s cheaper. [44] for example, reports that Audi has used Altera’s Cyclone V board (which contains an FPGA) in its autonomous driving solutions. The implication here is of course that these hardware solutions are still too expensive for general production, or affordable brands. In terms of the complexity of the hardware solution, it is not entirely evident how clustering or other dynamic tasks could be accomplished within a hardware solution. Given the complexity of such a task, it may in fact be worthwhile to only accelerate the initial stages of the algorithm, leaving the rest for software implementation. In general, however, implementations in hardware seem to be effective and popular for ADAS.

9.2 Improvements to Filtering and Clustering Steps

As detailed in **Chapter 6**, several filtering and clustering steps were used in the curb detection methods. These steps were crucial in the initial point removal and the maintainment of curb points. Although a significant portion of the curb has been maintained, these filters can be improved further. For example, the cosine similarity filter step described in **Section 6.2** removes all adjacent curb points to the vehicle. An augmented similarity filter was introduced in **Section 6.2.2** which improved the retention of curb points adjacent to the vehicle, however it introduced more noise. An improved filter that maintains full curb resolution and reduces noise from passing through would be an ideal filter for this step in the curb detection method. Further improving the filtering steps will significantly improve the resolution and accuracy of the final results for the curb detection method.

The clustering step described in **Section 3** assumes a constant point density for curb clusters. This assumption is incorrect since point density decreases the further the points are from the vehicle. Therefore, a new clustering method should be proposed that is able to handle variation of point densities [23]. HDBSCAN is a clustering method that does not require the epsilon hyperparameter from DBSCAN, and is able to cluster points with varying densities. This improves the point clustering by clustering more of the curb into a single cluster. The improved clustering will positively improve the rest of the method since the point cloud will have a more coherent structure.

Another possible clustering algorithm is a variation of HDBSCAN that clusters points based on a group normal vector. As described previously, curbs are expected to have a strong normal vector component in the x direction, and a weak component in the z direction. Therefore, clustering based on normal vector components will have an improvement for clustering curbs specifically. In addition, clusters that have a strong normal vector component in the z direction can be removed. This will remove the need of the Denoise step from the curb detection method, making it more efficient.

9.3 Machine Learning for Sliding-Window Filters

As explored in **Chapter 5**, sliding window approaches can be used to extract features from a set of data. The methods presented in **Chapter 5** failed to produce effective detection results since they failed to effectively produce a classification mechanism. The curb detection method presented in **Chapter 6** uses a multi-feature loose threshold approach to delineate curb points from others. Based on the papers introduced in **Chapter 3**, it is already possible to create machine learning models, such as convolutional neural networks, to classify road points. It should also therefore be possible to create networks to classify curb points. One such example of a road detection network is presented in [18], where a Deep CNN is trained to fuse LiDAR and image data. The data is fused over multiple layers, and the result of the final fusion layer is

passed to a classification layer which identifies the road features, allowing for highly accurate road detection. The same process can be followed for curb detection: train a convolutional neural network to classify curb points and, as input, pass full LiDAR snapshots. The CNN uses filters which traverse the input data as a sliding window. The difference being that a CNN, during training, learns the weights for the filters such that they can be used to identify curbs or other objects. This would likely be a large improvement to the tested sliding window approaches since the filter would be learned directly from real data, not from an abstracted view of what curbs should appear like.

Neural networks are advantageous since, once they are trained, they only require a single pass through the network to classify a point or set of points [36]. One disadvantage with a neural network is the reliance on labelled training data which unfortunately does not exist in large enough quantities for curb detection. This may explain the heavy focus of many papers on road detection with LiDAR since datasets like KITTI provide labelled point clouds and images for roads and many obstacles. For the purposes of this project, it was sufficient to use a detected road edge to corroborate curb detection. As a research topic, road detection with CNNs in LiDAR is well developed, and could provide useful input to the corroboration of curbs. However, additional problems exist with CNNs: large computational complexity, memory usage, and power consumption. Without hardware acceleration, a real-time system using CNNs as a main detection mechanism is hard to foresee. Tesla, for example, has fabricated ASICs specifically for computer vision tasks since the sheer volume of information to process is immense [45]. CNNs for image processing can have many hundreds of thousands of weights, and may require millions of operations to complete a single detection. The same is expected for LiDAR-based CNNs, so it may not be practical to deploy neural networks as such to production vehicles.

9.4 Sensor Fusion

Chapter 3 introduced curb and road detection techniques ranging from camera- to LiDAR-based methods. The research conducted for this project revealed a trend in the highest-accuracy detection techniques: sensor fusion. When two sensors were used to corroborate one another, the curb and/or road detection was much more accurate. This was realized specifically with Progressive LiDAR Adaptation for Road Detection (PLARD), which fused cameras and LiDAR through a progressive mapping technique [18]. The same idea is most likely applicable to curb detection, with the caveat that many of these high-accuracy techniques employ machine learning which, as discussed above, is not suitable for curb detection due to the virtual non-existence of labelled datasets. However, it may be possible, such as was attempted in this project, to corroborate curb detection with road detection. In this case, the detection of a road and/or curb could be useful in increasing confidence scores, granted that a confidence metric could be created to handle such a task. A method of curb detection that should be investigated towards this end is introduced in [14], where stereo cameras are used to construct elevation maps where edges can be extracted, and curbs identified. Using sensor fusion techniques, the camera-detected curbs could then be overlaid

onto the LiDAR-detected curbs and corroborated. Many other schemes are possible, but this certainly seems to be viable, and worth investigation.

Conclusion

Advanced driver assistance systems are a key element of the modern vehicle. Their use cases extend from the mundane tasks: blind spot detection, lane departure warnings, and so on; to the complex tasks: adaptive cruise control, collision avoidance, and more. The common threads here: ADAS should aid in accident prevention, and it should be supplemental to the driver, not replacing the driver. That said, ADAS can be seen as a primary stepping stone towards fully autonomous driving. As they currently stand, these systems are not capable of handling dynamic driving tasks, nor are they capable of human-level decision making; however, research towards improvements in ADAS technologies leads closer to fully automated vehicles. This project strove to develop an environmental perception method which could be used to aid human drivers and ADAS in the safe completion of the driving task. Environmental perception is a necessary category of assistance systems as such. Without adequate perception of the environment, a system would struggle to make decisions, let alone safety-critical decisions. Thus, for the safety of vehicle occupants and the system as a whole, the methods were designed with real-time, reliable operation in mind.

Through a literature review, contemporary methods of environmental perception for road and curb detection were studied. Methods ranging from camera-based detection, to LiDAR-based detection and sensor fusion were investigated for their benefits and drawbacks. It was discovered that sensor fusion between LiDAR and cameras often heralded the best results for road detection; the main benefit being that the separate sensors could corroborate one another to produce a better, more confident detection. Although sensor fusion was not used in this project, the idea of mutual corroboration was instrumental in the development and integration of the two presented methods: curb detection, and road detection. The curb detection method used a filtering methodology whereby points were removed from a point cloud until the curb could be identified and extracted. Road detection on the other hand identified inflection points on the road, and used these to classify the edges of the road. Inflection points were used to corroborate the curb detection, and provide a higher confidence level to those segments that consisted of, or resided near, an inflection point. Using these methods, the edges of the road can be identified and reported to an ADAS for further processing and decision making. Considerable improvements can still be made to the presented methods to further increase detection accuracy, testability, and confidence.

The results presented in **Chapter 8** clearly indicated that the detection method could identify curbs with relatively high resolution, without many false positives in the roadway. Problems however arose with the method during sparsity testing, where false positives from early filtering steps in curb detection would lead to false positive segments within the roadway. Two improvements can be made towards this end: 1) improve the filtering steps presented in **Section 6.2** so that fewer points remain on the road, and 2) improve segment grouping and intra-cluster confidence to remove segments that are dissimilar. In addition, the confidence metrics often fail to produce usable confidence levels because they are only calculated within

clusters. Inter-cluster confidence must therefore be improved to give usable confidence levels. Finally, a method of comparing detection results to the real location of curbs must be developed in order to benchmark the detection methods. This is problematic as labelling curbs is a tedious task, and it is not clear how segments could be compared to the real curb location, nor how the confidence levels could be measured for accuracy. These issues and recommendations are left for further research.

As per the goal of this project, through the various implementations and designs, curb detection has been accomplished. Novel methods were created towards this goal. The filtering steps, both in the method and the approach, were either entirely new, or improvements on contemporary methods. The methodology towards filtering was identified from research in the field, and adapted to a new approach. The combination of multiple research papers and innovation on top of them led to the results herein presented. The method of segmentation was entirely novel, and while it still requires development, is a useful approach to reporting the curb location. Finally, the combination of multiple LiDAR-based methods for corroboration has not been identified in research, and so it is considered a novel method of fusion. Using the results of the curb detection method, an ADAS system would be able to identify curb locations with relatively high certainty, thus accomplishing the goal of the project, allowing for better environmental perception and a safer driving experience.

References

- [1] T. Eady, "Tesla's Deep Learning at Scale: Using Billions of Miles to Train Neural Networks", *Medium*, 2019. [Online]. Available: <https://towardsdatascience.com/teslas-deep-learning-at-scale-7eed85b235d3>. [Accessed: 21- Feb- 2020].
- [2] Transport Canada, "Canadian Motor Vehicle Traffic Collision Statistics," 2017. [Online]. Available: <https://www.tc.gc.ca/eng/motorvehiclesafety/canadian-motor-vehicle-traffic-collision-statistics-2017.html>. [Accessed Sept. 20, 2019].
- [3] Ontario, "Safe and Responsible Driving," 2019. [Online]. Available: <https://www.ontario.ca/document/official-mto-drivers-handbook/safe-and-responsible-driving>. [Accessed Sept. 20, 2019].
- [4] CAA National, "Distracted Driving," [Online]. Available: <https://www.caa.ca/distracted-driving/statistics/>. [Accessed Sept. 20, 2019].
- [5] Government of Canada, Royal Canadian Mounted Police, "Distracted driving and fatigued driving," [Online]. Available: <http://www.rcmp-grc.gc.ca/cycp-cpcj/dd-dv/index-eng.htm>. [Accessed Sept. 20, 2019].
- [6] CBC News, "Distracted drivers face licence suspension, increased fines starting Jan. 1, says MTO," [Online]. Available: <https://www.cbc.ca/news/canada/thunder-bay/distracted-driving-penalty-increase-1.4916615>. [Accessed Sept. 20, 2019].
- [7] Transport Canada, "Automated and connected vehicles 101," [Online]. Available: <https://www.tc.gc.ca/en/services/road/innovative-technologies/automated-connected-vehicles/av-cv-101.html>. [Accessed Sept. 20, 2019].
- [8] "Automatic emergency braking - Transport Canada", *Tc.gc.ca*, 2019. [Online]. Available: <https://www.tc.gc.ca/en/services/road/vehicle-technologies/automatic-emergency-braking.html>. [Accessed: 21- Feb- 2020].
- [9] "Adaptive Cruise Control - Transport Canada", *Tc.gc.ca*, 2019. [Online]. Available: <https://www.tc.gc.ca/en/services/road/vehicle-technologies/adaptive-cruise-control.html>. [Accessed: 21- Feb- 2020].
- [10] "Blind Spot Detection - Transport Canada", *Tc.gc.ca*, 2019. [Online]. Available: <https://www.tc.gc.ca/en/services/road/vehicle-technologies/blind-spot-detection.html>. [Accessed: 21- Feb- 2020].
- [11] "How does LiDAR work?", *Lidar-uk.com*, 2020. [Online]. Available: <http://www.lidar-uk.com/how-lidar-works/>. [Accessed: 21- Feb- 2020].
- [12] Velodyne Lidar, Inc., "HDL-64E S3 High Definition Real-Time Lidar Spec Sheet", 2018.

- [13] SAE International, "Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems", SAE International, 2020.
- [14] F. Oniga, S. Nedevschi and M. M. Meinecke, "Curb Detection Based on a Multi-Frame Persistence Map for Urban Driving Scenarios," 2008 11th International IEEE Conference on Intelligent Transportation Systems, Beijing, 2008, pp. 67-72.
- [15] A. Geiger, P. Lenz and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," 2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, 2012, pp. 3354-3361.
- [16] J. Fritsch, T. Kühnl and A. Geiger, "A new performance measure and evaluation benchmark for road detection algorithms," 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013), The Hague, 2013, pp. 1693-1700.
- [17] Lyu, Yecheng & Huang, Xinming. (2018). Road Segmentation Using CNN with GRU.
- [18] Z. Chen, J. Zhang and D. Tao, "Progressive LiDAR adaptation for road detection," in IEEE/CAA Journal of Automatica Sinica, vol. 6, no. 3, pp. 693-702, May 2019.
- [19] Caltagirone, Luca & Bellone, Mauro & Svensson, Lennart & Wahde, Mattias. (2018). LIDAR-camera fusion for road detection using fully convolutional neural networks. Robotics and Autonomous Systems. 111. 10.1016/j.robot.2018.11.002.
- [20] S. Gu, Y. Zhang, J. Yang, J. M. Alvarez and H. Kong, "Two-View Fusion based Convolutional Neural Network for Urban Road Detection," 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China, 2019, pp. 6144-6149.
- [21] S. Gu, Y. Zhang, J. Tang, J. Yang and H. Kong, "Road Detection through CRF based LiDAR-Camera Fusion," 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 2019, pp. 3832-3838.
- [22] F. Xu, L. Chen, J. Lou and M. Ren, "A real-time road detection method based on reorganized lidar data", PLOS ONE, vol. 14, no. 4, p. e0215159, 2019. Available: 10.1371/journal.pone.0215159.
- [23] G. Wang, J. Wu, R. He and S. Yang, "A Point Cloud-Based Robust Road Curb Detection and Tracking Method," in IEEE Access, vol. 7, pp. 24611-24625, 2019.
- [24] N. Charron, S. Phillips and S. L. Waslander, "De-noising of Lidar Point Clouds Corrupted by Snowfall," 2018 15th Conference on Computer and Robot Vision (CRV), Toronto, ON, 2018, pp. 254-261.
- [25] Y. Zhang, J. Wang, X. Wang and J. M. Dolan, "Road-Segmentation-Based Curb Detection Method for Self-Driving via a 3D-LiDAR Sensor," in IEEE Transactions on Intelligent Transportation Systems, vol. 19, no. 12, pp. 3981-3991, Dec. 2018.
- [26] G. Zhao and J. Yuan, "Curb detection and tracking using 3D-LIDAR scanner," 2012 19th IEEE International Conference on Image Processing, Orlando, FL, 2012, pp. 437-440.

- [27] "How does LiDAR work?", *Lidar-uk.com*, 2020. [Online]. Available: <http://www.lidar-uk.com/how-lidar-works/>. [Accessed: 21- Feb- 2020].
- [28] "(转)从零实现3D图像引擎 : (12)构建支持欧拉和UVN的相机系统 - CoolJie - 博客园", *Cnblogs.com*, 2020. [Online]. Available: <https://www.cnblogs.com/CoolJie/archive/2011/04/25/2027361.html>. [Accessed: 27- Mar- 2020].
- [29] City of Toronto, Transportation Services, "10.0 CURB EXTENSIONS GUIDELINE", City of Toronto, Toronto, 2017.
- [30] J. Tan, J. Li, X. An, and H. He, "Robust Curb Detection with Fusion of 3D-Lidar and Camera Data," *Sensors*, vol. 14, no. 5, pp. 9046–9073, May 2014.
- [31] Zhao Liu, Jinling Wang, and Daxue Liu, "A New Curb Detection Method for Unmanned Ground Vehicles Using 2D Sequential Laser Data", 2013. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3574724/pdf/sensors-13-01102.pdf>
- [32] Saad Bedros, "Hough Transforms and Thresholding" [online], Available at: <http://me.umn.edu/courses/me5286/vision/Notes/2015/ME5286-Lecture9.pdf> [Accessed 02-March-2020].
- [33] "Fourier Transform - an overview | ScienceDirect Topics", *Sciencedirect.com*, 2020. [Online]. Available: <https://www.sciencedirect.com/topics/engineering/fourier-transform>. [Accessed: 27- Mar- 2020].
- [34] Bazazian, D., Casas, J. and Ruiz-Hidalgo, J. (2016). Fast and Robust Edge Extraction in Unorganized Point Clouds. *IEEE*. [online] Available at: <https://ieeexplore.ieee.org/abstract/document/7371262> [Accessed 25 Nov. 2019].
- [35] José, I. (2019). *KNN (K-Nearest Neighbors) #1*. [online] Medium. Available at: <https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d> [Accessed 25 Nov. 2019].
- [36] A. Burkov, The hundred-page machine learning book, 1st ed. [S.l.]: Andriy Burkov, 2019, pp. 61-72.
- [37] MIT, "Power-Method", <http://web.mit.edu/18.06/www/Spring17/Power-Method.pdf>, Accessed April 4, 2020
- [38] E. Lutins, "DBSCAN: What is it? When to Use it? How to use it.", Medium, 2020. [Online]. Available: <https://medium.com/@elutins/dbscan-what-is-it-when-to-use-it-how-to-use-it-8bd506293818>. [Accessed: 27- Mar- 2020].
- [39] Al-Masri, A., 2019. How Does K-Means Clustering In Machine Learning Work?. [online] Medium. Available at: <https://towardsdatascience.com/how-does-k-means-clustering-in-machinelearning-work-fdaaaf5acfa0> [Accessed 4 April 2020].
- [40] Scikit Learn (2019). https://ogrisel.github.io/scikit-learn.org/sklearn-tutorial/_images/plot_cluster_comparison_1.png. [image].

- [41] A. Kassambara, "DBSCAN: Density-Based Clustering Essentials" [online], Available at: <https://www.datanovia.com/en/lessons/dbscan-density-based-clustering-essentials> [Accessed 25 Nov. 2019].
- [42] F. Xu, L. Chen, J. Lou and M. Ren, "A real-time road detection method based on reorganized lidar data", *PLOS ONE*, vol. 14, no. 4, p. e0215159, 2019. Available: 10.1371/journal.pone.0215159.
- [43] H. Wallace, "SYSC4507_LectureSlides_6_Winter2019", Carleton University, Ottawa, ON, 2020.
- [44] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang and W. Shi, "Edge Computing for Autonomous Driving: Opportunities and Challenges," in *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697-1716, Aug. 2019.
- [45] S. Hollister, "Tesla's new self-driving chip is here, and this is your best look yet", *The Verge*, 2020. [Online]. Available: <https://www.theverge.com/2019/4/22/18511594/tesla-new-self-driving-chip-is-here-and-this-is-your-best-look-yet>. [Accessed: 31- Mar- 2020].