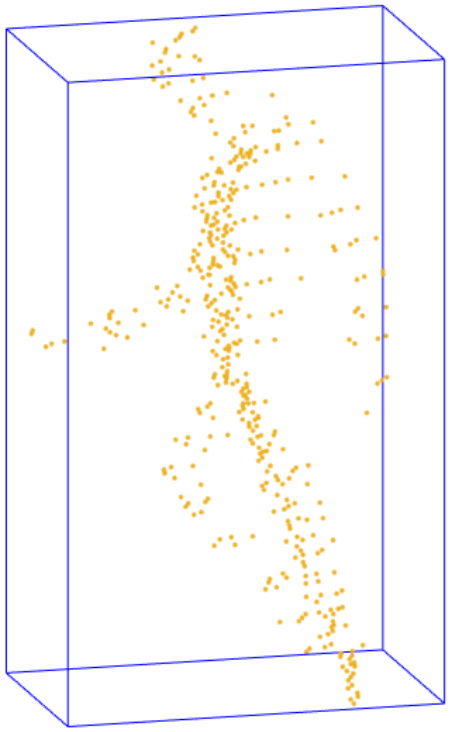
CHALMERS
UNIVERSITY OF TECHNOLOGY



Object detection by cluster analysis on 3D-points from a LiDAR sensor

Master's thesis in Systems, Control and Mechatronics

DANIEL ERIKSSON
JONAS HARSTRÖM

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Object detection by cluster analysis on 3D-points from a LiDAR sensor

An evaluation of a LiDAR's suitability for autonomous drive purposes in a test track environment and the design of an object detection algorithm by ground plane estimation and cluster analysis

DANIEL ERIKSSON
JONAS HARSTRÖM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Signals Processing and Biomedical Engineering
Signal Processing Group
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Object detection by cluster analysis on 3D-points from a LiDAR sensor
An evaluation of a LiDAR's suitability for autonomous drive purposes in a test track environment and the design of an object detection algorithm by ground plane estimation and cluster analysis
Daniel Eriksson
Jonas Harström

Supervisor: Albert Lawenius, Volvo Car Corporation
Examiner: Karl Granström, Department of Signals and Systems

Department of Electrical Engineering
Division of Signals Processing and Biomedical Engineering
Signal Processing Group
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 10 00

Cover: A pedestrian detected by the algorithm developed in the thesis.

Typeset in LaTeX
Gothenburg, Sweden 2019

Object detection by cluster analysis on 3D-points from a LiDAR sensor
An evaluation of a LiDAR's suitability for autonomous drive purposes in a test track environment and the design of an object detection algorithm by ground plane estimation and cluster analysis

DANIEL ERIKSSON
JONAS HARSTRÖM
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Autonomous drive is one of the most discussed and researched areas within the automotive industry today, with several companies and universities conducting research and development to make it a reality. It may not only be beneficial for transportation, but for industrial purposes as well.

Volvo Car Corporation is interested in introducing autonomous vehicles which can perform endurance tests at their proving ground, where they want to explore the possibilities of using a LiDAR as an external reference input. This thesis aims to take the initial steps to make this a reality. First, an evaluation of the LiDAR's suitability at the test track is made and then an object detection algorithm is designed. The object detection algorithm is based on two key components, ground plane estimation based on RANSAC and then cluster analysis. This is a less common approach compared to commercial object detection algorithms which often utilises machine learning instead.

The resulting object detection algorithm proved to have some interesting characteristics with both advantages and disadvantages when compared to commercial algorithms and the conclusion of the LiDAR evaluation is that it is an appropriate sensor to use in a proving ground environment.

Keywords: Autonomous Drive, Object detection, LiDAR distortion analysis, Cluster analysis, Ground plane estimation, DBSCAN, RANSAC.

Acknowledgements

We would like to express sincere appreciation and thanks to all people who helped us throughout the thesis. Firstly we would like to thank our examiner at Chalmers, Karl Granström, for giving new perspectives of certain problems, propositions of solving methods and answers regarding general thesis questions. We would like to convey great gratitude to Volvo Car Corporation for the thesis opportunity. Especially our supervisor Albert Lawenius from the team responsible for the driverless, AD and ADAS proving ground environment, for support regarding practical questions and his engagement pitching this thesis to other departments of Volvo Cars, and a huge thanks to Elías Marel at active safety's sensor team for his personal interest in our work and guidance when we encountered technical issues, whether regarding the LiDAR sensor or code implementation.

Daniel Eriksson & Jonas Harström, Gothenburg, June 2019

Contents

List of Figures

List of Tables

AD	Autonomous Drive
ADAS	Advanced Driver-Assistance Systems
CNN	Convolutional Neural Network
DBDL	Density Based Detection for LiDAR
DBSCAN	Density Based Spatial Clustering of Applications with Noise
RANSAC	Random Sampling Consensus
EM	Expectation-Maximisation
FN	False Negative
FoV	Field of View
FP	False Positive
GNSS	Global Navigation Satellite System
GPE	Ground Plane Estimation
HPG	Hällered Proving Ground
IMU	Inertial Measurement Unit
INS	Inertial Navigation System
MTT	Multi Target Tracking
NN	Neural Network
RNN	Recurrent Neural Network
TN	True Negative
TCP	Transmission Control Protocol
TP	True Positive
VCC	Volvo Car Corporation

1
Introduction

The purpose of this chapter is to introduce the reader to the problems treated in the thesis, as well as contributing with some background, related work, how the problems are approached and in the end be able to state objectives and contributions.

1.1 Vehicle testing

In order to be able to guarantee vehicle customers the promised quality, automotive companies need to put their products through rigorous testing before releasing. Volvo Car Corporation (VCC) are currently performing these tests at Hällered Proving Ground (HPG), an enormous test facility with a total of 15 different test tracks. During the course of a couple of months, the cars are exposed to the same amount of stress they should experience in their whole expected lifetime and about 2000 tires are consumed each year during testing [3]. Every possible aspect are put to the test, high speed testing surpassing 250 km/h, handling under various conditions, off-road testing and gravel tracks as well as tracks testing endurance and comfort.

Figure 1.1: An aerial shot of Hällered Proving Ground

All this testing requires a workforce of around 75 test drivers, which drives a total of 160 000 kilometres every year [3]. These kilometres are not only putting the cars under a lot of stress, but also the test drivers. In order to avoid work related injuries, the most

brutal endurance tests have limits for how much each employee can drive each day. Because of this, Volvo Car Corporation is investigating the possibilities of introducing driverless [25] tests in order to avoid the above mentioned constraints, spare the test drivers from physical stress as well as taking part of the economic advantages associated with automation.

1.2 Autonomous drive and LiDAR

Today, autonomous drive (AD) is one of the most prominent research areas within the automotive industry [15] with numerous automotive companies actively conducting research as well as companies exclusively focusing on developing solutions for autonomous drive [1].
The problem can be divided into four steps: detection, object tracking, motion forecasting and motion planning [28]. CUrrently, there exist a myriad of different solutions for detection, utilising a diverse set of sensor setups such as cameras, LiDARs, Radars, ultra-sonic sensors and various algorithms processing the input to identify crucial information about the environment [23].

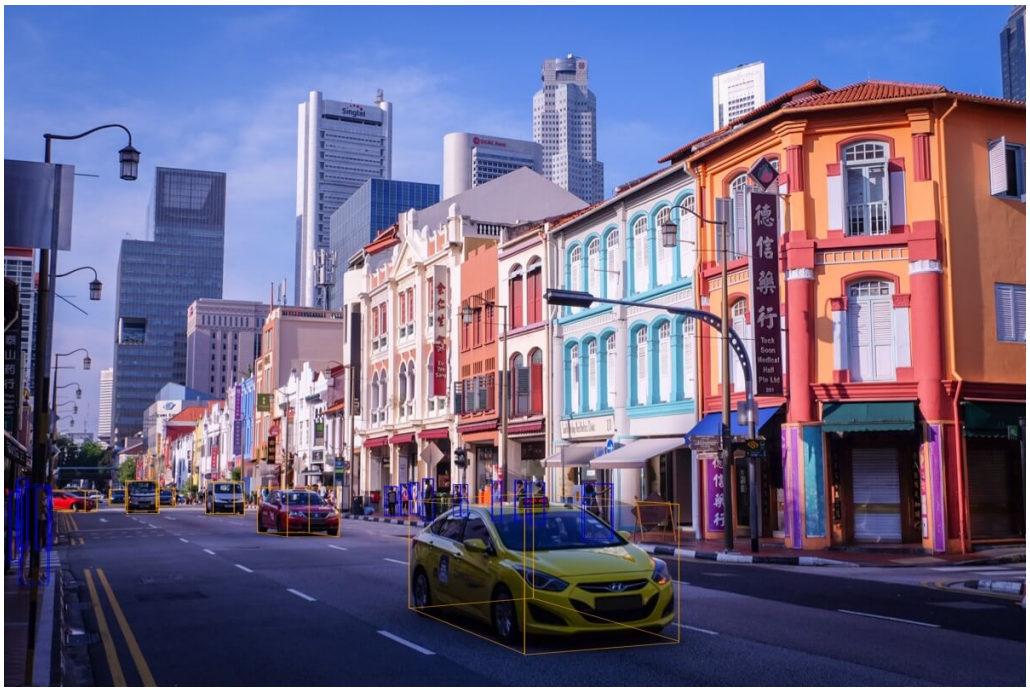


Figure 1.2: An example of what the output could look like after applying object detection on images with so called convolutional neural networks, a machine learning approach [18].

Volvo Car Corporation has previous experience with radar scanners and ultra sonic sensors, which is integrated in some of the active safety features of their current production cars on the market today [4], [5], [11], [14]. In recent years though, they have invested both interest and resources into LiDAR technology which they expect to bring value to the development of autonomous vehicles [13]. The LiDAR's advantage compared to other

sensor setups are its undisputed detail and accuracy representation of a 3D environment, which hopefully will bring the future of autonomous vehicles a little closer [23].

In case of VCC's ambition to introduce driverless tests though, the conditions are somewhat different compared to the usual public environment where most vehicles are driven. The concerns is that the LiDAR will not be able to cope with the harsh environment of a proving ground, where biggest worry is that the bumpy test tracks will distort the LiDAR scans and render the information inaccurate or in worst case, unusable.

1.3 Selection of object detection approach

One of the more common approaches for object detection is the utilisation of machine learning [33], which is commonly performed on ordinary camera images, but it is possible to apply on LiDAR scans as well. This approach has been proven successful but it is not without its flaws. For example, the algorithm described in [38] (a tailored convolutional neural network for LiDAR data) performs very well at vehicle detection but the score for animal detection is more than 70 times lower than the vechile detection, which is probably due to the fact that the dataset the network was trained on contained a small amount of animal data. This leads us to the next shortcoming; its inability to detect objects which the neural network has not been trained for. This is an immense problem since it is near impossible to cover every possible obstacle, which could block a vehicle's path on the road, in the training data [37]. Another backside of machine learning is difficulties to understand and control what is happening inside a neural network. It either identifies an object correctly or it does not, and it is very hard to identify the reason for a failed detection. That is why the thesis will explore object detection by cluster analysis, which will make it possible to analyse and control each function, and will not require data collecting.

1.4 Objectives

The thesis can be partitioned into two main tasks; the development of an object detection algorithm and an investigation of the LiDAR and how the placement copes with various road disturbances.

1.4.1 Object detection

The object detection problem with LiDAR can be defined as follows:

> *To, with the given 3D-point cloud from a LiDAR frame* $X = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$, $X \subseteq \mathbb{R}^3$*, identify objects* $O = \{o_1, o_2, \cdots, o_k, \cdots, o_K\}$ *which may obstruct the path of the vehicle where the entries of the vector* o_k *are properties of the detected object* k*.*

Since cluster analysis has been chosen as the main approach for object detection, the problem formulation can be narrowed down even further.

> *To, with the given 3D-point cloud from a LiDAR frame* $X = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$, $X \subseteq \mathbb{R}^3$
>
> 1. *Determine which points in* X *belong to traversable surface* $X_T \subseteq X$ *and which points does not (non-traversable)* $X_{NT} \subseteq X$.
> 2. *Identify objects* $O = \{o_1, o_2, \cdots, o_k, \cdots, o_K\}$ *and their properties in set of points* X_{NT},

Thus, the first task of the thesis will be to develop an algorithm which solves the problem defined above.

1.4.2 LiDAR placement and disturbance analysis

VCC is interested in an investigation which aims to bring further understanding of how the LiDAR mounting position impacts practical aspects as well as how the scans are influenced by disturbances induced by rough and bumpy terrain (henceforward referred to as road disturbances) depending on the placement. The term mounting position is here referring to key areas of the vehicle where the LiDAR may be positioned such as the roof or hood and does not involve fine adjustments such as angular position or i.e fine tuning the placement by a few centimetres.

The questions which are to be answered are:

> How much will road disturbances distort the LiDAR scans depending on the LiDAR mounting position?
>
> Disregarding influence of disturbance, what are the general advantages and disadvantages for each mounting position?
>
> Taking the two questions above into account, which mounting position can be regarded as optimal or be recommended for use?
>
> Is the LiDAR a suitable sensor to use for autonomous vehicles at HPG, given the harsh road conditions?

1.5 Related work

The subject of object detection using point clouds from a LiDAR sensor has been a common area of research for many engineers and scientists. Several published articles about object detection using LiDAR will be the foundation for this thesis, but with modified approaches from subjective interests and idea to fit the criteria set by the people involved in this project.

Asvadi et al. [16] released a relevant article about detecting obstacles from LiDAR scans, by dividing the scanned region into multiple regions before processing the data for object detection. With an improved point cloud, generated from comparing GNSS/INS data with successively scanned data frames, a ground plane estimation is performed. They are using the random sampling consensus RANSAC algorithm, which is widely used for finding patterns in noisy data, and is an adequate method when inlier/outlier label is unknown [20]. After outliers have been separated from the dataset, they are fed as input to a voxelisation algorithm which converts geometrical objects into voxels for a three-dimensional representation. The whole function chain can determine which points belongs to static or dynamic obstacles and also visualise them by plotting the voxels containing the obstacle data points.

Choi et al. [20] also uses the RANSAC algorithm to fit a plane to three-dimensional data points, but they apply an asymmetric kernel to RANSAC, which they explains as follows: "Our asymmetric kernel is inspired from log-likelihood of a 3D point with its unknown label. Since outliers always exist above the ground plane, our kernel has longer tail in negative domain. Such asymmetry enables RANSAC more robust to outliers, which was shown in our experiments and application" [20]. Their motivation for a modified RANSAC is that it gives more robustness to the functionality with less computational time.

Mufti et al. [32] have a similar approach as Choi et al. [20] where a likelihood function is integrated with the RANSAC algorithm. The outliers are then defined as either belonging to the ground plane or not, based on if it fulfils a threshold criterion for the distance to the approximated plane.

In the article "Background Filtering and Vehicle Detection with Roadside Lidar Based on Point Association" [41] Zhang et al. uses a clustering technique to detect vehicles using LiDAR point clouds. Associated data points are gathered into clusters by comparison between positional information and distance parameters.

Another approach, which differs from the aforementioned object detection methods, is by utilising machine learning, where it also becomes possible to perform target tracking, motion forecasting and classification. Luo et al [28] uses a 3D convolutional network to detect objects in a point cloud together with tracking and short time motion forecasting. The short explanation of their method is that they feed several voxelised 3D frames as a 4D tensor through a convolution network to extract features, determine labels and predict future states.

A combination between clustering analysis and machine learning is an alternative that Matti et al. [30] implement in their research. Region of interest is determined by clustering analysis to exclude unnecessary data points before classifying the objects using the high performing Residual Network, also known as ResNet [22].

1.6 Contributions

With the following tasks defined, this thesis aims to produce:

- A recommendation for cluster analysis technique for object detection with LiDAR.
- An object detection algorithm based on the recommended cluster algorithm which will act as input for either a future path planning or object tracking algorithm.
- An evaluation of how the object detection developed in the thesis compare to the commercial approaches and its potential place in the field of autonomous drive.
- Recommendations for future work in order to reach the goal of introducing driverless tests at HPG.
- An analysis of the LiDAR placement as well as verdict of the LiDAR's suitability in a test track environment.

2
Theory

In this chapter, the theory utilised throughout the thesis is presented. It commences with a brief explanation of how a LiDAR functions, then continues with theory for evaluation of object detection. This is then followed by three sections dedicated to explaining the theory behind the algorithm develop in the thesis; cluster analysis, traversable surface estimation and bounding geometry. Lastly, the theory behind the LiDAR mounting point investigation is explained.

2.1 Light detection and ranging, LiDAR sensor

The LiDAR sensor is an optical measurement unit which measures the distance to a reflective point by transmitting a pulsed laser [34]. It is possible to map the environment in 3D due to moving mirrors inside the LiDAR hardware, which creates a certain scanning pattern that can be seen in Figure 2.1a. The overlapping section in the middle of Figure 2.1a is due to two side-by-side mounted transmitters (Figure 2.1b), which increases the horizontal field of view.

(a) Approximate scanning pattern for the used LiDAR sensor

(b) Appearance of the Luminar LiDAR with two optical transmitters

Figure 2.1: Scanning pattern and LiDAR hardware

The resolution and refresh rate settings are adjustable parameters for the LiDAR that can be adapted for certain needs. A refresh rate of 10 Hz delivers approximately 60000 data points, including both snapback-points when the mirrors are resetting in the end of a scan and transmitted light with no received reflected light. The LiDAR's output is these data points structured as a 3D point cloud together with various parameters, such as intensity of the reflected light. Figure 2.2 shows an example for a scanned LiDAR frame and it is possible to distinguish several objects, such as people and buildings.

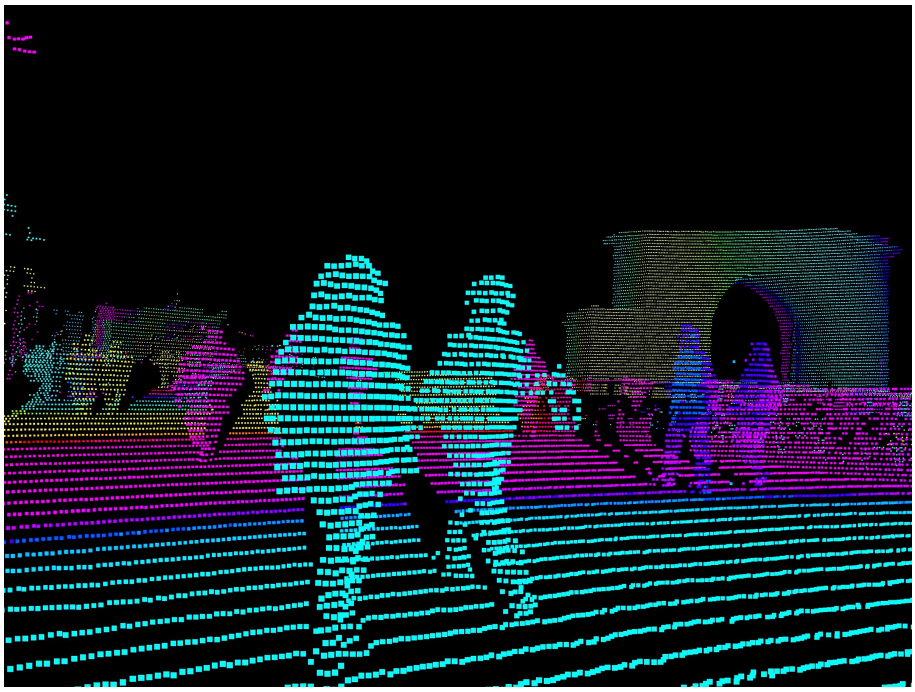


Figure 2.2: One frame from a LiDAR scan. Shows data points in different colours depending on the distance to the reflective point

2.2 Object detection

Object detection is a two-part process consisting of classification and localisation. Classification determines the object label, if it is a car, a pedestrian or an animal and so forth. Classification is a very important part of autonomous drive in commercial cars because the car should avoid colliding with aforementioned objects, but should maintain speed if, for example, a plastic bag blows across the road in front of the car [35]. Classification can be done in two separate manners. Binomial classification can decide if an object is an obstacle or not and multi-label classification can specify the label of an obstacle.

Localisation almost resolves itself when using LiDAR data. The data points' coordinates are relative the LiDAR, which gives the position vector instantaneously. It is also possible to extract additional information after performing clustering algorithms, such as cluster centre point. A geometry fitting algorithm can be convenient to form zones surrounding the obstacles and give information about their locations.

Classification and localisation are merged to construct complete functionalities of an object detection principle.

2.2.1 Multi-label classification

Machine learning is often applied when the aim is to give an object a specific label. It is in many cases, within the autonomous drive field, absolutely fundamental to distinguish between various types of objects and handle them differently. The convolutional neural network (CNN) returns probabilities for the input to be one of several predetermined labels [39]. Training the CNN is vital to maximise the classification accuracy for each

iteration. The accuracy can alter considerably depending on obstruction and amount of data the network was trained on. Collect and annotate data from the test track would be essential to use as training data, because of the specific driving environment, to increase the accuracy.

2.2.2 Binomial classification

Binomial classification is a task to classify an element into two groups, commonly in autonomous drive as an obstacle or not an obstacle. The outcome of object detection using binomial classification can be divided into four different conditions, described in Table 2.1 as a 2×2 confusion matrix [8].

		True condition	
		p	**n**
Predicted condition	**p′**	True Positive	False Positive
	n′	False Negative	True Negative

Table 2.1: Confusion matrix over the four different conditions, which results in classification as either *True* or *False*

- **True Positive (TP)**: The model correctly predicts an existing obstacle.
- **True Negative (TN)**: The model correctly predicts a non-existing obstacle.
- **False Positive (FP)**: The model incorrectly predicts a non-existing obstacle.
- **False Negative (FN)**: The model incorrectly predicts an existing obstacle.

TP and TN are successful outcome from binomial classification and the aim is to maximise their probabilities.

2.2.3 Mean Average Precision

The mean Average Precision (mAP) is a popular metric for calculating the accuracy for object detection. It is a measure of the average precision for each class, based on model predictions and is a function of the metrics precision and recall [6]. The formulas for precision and recall are:

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

Precision measures the accuracy for model predictions, i.e. the percentage of correct predictions and recall measures how well the model finds existing objects. In an ideal world both precision and recall equations would yield the result 1.0, but is not the case in reality.

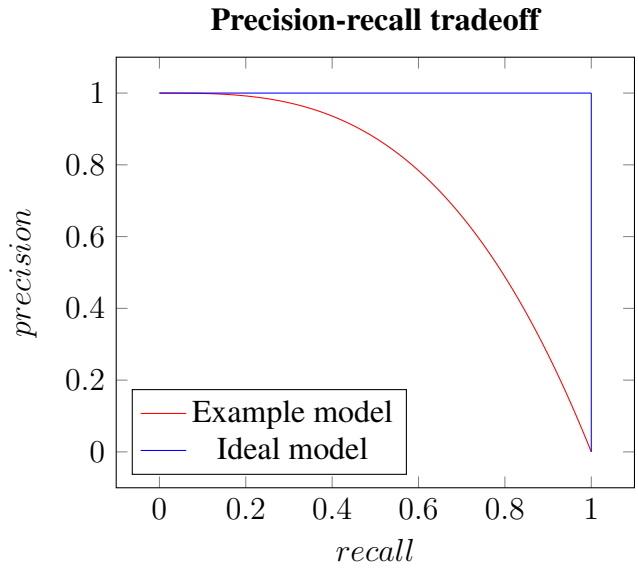


Figure 2.3: Graph showing ideal tradeoff versus example tradeoff for Precision-Recall

The trade-off graph between precision and recall are displayed in Figure 2.3, where the precision p is a function of the recall r. For simplicity, the example model is approximated as,

$$precision = -recall^2 + 1 \tag{2.3}$$

The general expression for average precision (AP) is the area beneath the model's trade-off curve,

$$AP = \int_0^1 p(r)dr \tag{2.4}$$

If precision and recall have been computed in a discrete set of points, AP can be computed as,

$$AP = \frac{1}{N} \sum_{n=1}^{N} \max_{n \leq i \leq N} p(r_i) \tag{2.5}$$

where N is the total number of discrete points and r_i is the recall value at time-instance i.

The AP values for all possible classes are consolidated and divided to get the mean value, mAP.

2.3 Cluster analysis algorithms

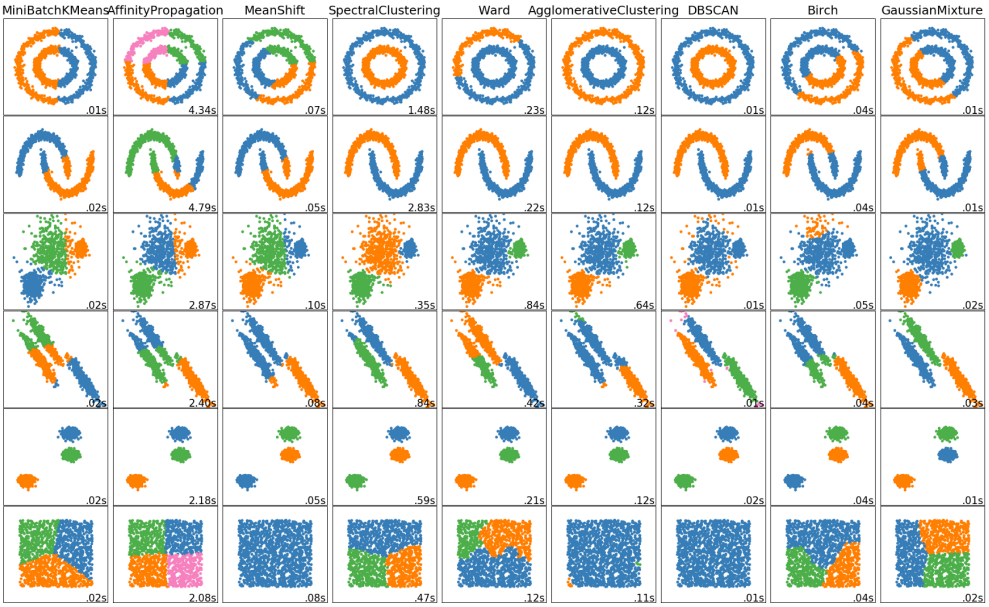


Figure 2.4: A figure picturing the results of various cluster techniques applied on datasets of varying properties [2]

Cluster analysis is the act of distinguishing, in a given set of data $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\}$, a set of K disjoint groups referred to as clusters $C = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_K\}$, where $\boldsymbol{x}_i$ is an n-dimensional feature vector. The number of clusters K does not necessarily need to be known in advance, but can be estimated by some cluster analysis techniques. The partitioning of the data is based on mutual similarities/dissimilarities of the observed data in the set, where data within a cluster are more similar than compared to data in other clusters. The benefits of cluster analysis is not only that of the identification of groups, but the result also allows us to represent the data in a more compact way [40]. Cluster analysis may be also be referred to as *unsupervised learning typology*, *Q-analysis*, *clumping* and *taxonomy*, depending on the domain where the clustering is applied.

There exists several methods of cluster analysis, all of which have got their appropriate application depending on the situation, but the clusters they produce all share some common properties .

- A cluster $\boldsymbol{c}_i$, $i = 1, \cdots, k$ is not empty.
- The union of every cluster $\bigcup_{i=1}^{k} \boldsymbol{c}_i$ forms the original dataset X.
- An observation $\boldsymbol{x}_i$, $i = 1, \cdots, N$ may only belong to one cluster $\boldsymbol{c}_i$, $i = 1, \cdots, k$.

The three cluster analysis algorithms that were chosen for evaluation is described more in depth below.

2.3.1 K-means clustering

The following text is based on the theory presented in [17]. The objective of the K-means algorithm is to partition a given set of N observations $\boldsymbol{S} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n, \cdots, \boldsymbol{x}_N\}$ of a

D dimensional, euclidean random variable $\boldsymbol{x}_n$ into K clusters (the number K is predefined by the user). The idea is that the euclidean distance in-between the points of a cluster is smaller than the distance to points belonging to other clusters. Though, the K-means algorithm will not consider every possible distance between every possible pair of points, but will instead compare an observation's distance to a set of vectors $\boldsymbol{\mu}_k$, $k = 1, ..., K$ which is representing the centres of each cluster. The decision variable for assigning each observation to a cluster will be $r_{nk} \in \{0, 1\}$ where $n = 1, ..., N$ and $k = 1, ..., K$ which will be 1 if observation n is assigned to cluster k and 0 otherwise [17].

The objective function which is to be minimised is:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2 \tag{2.6}$$

The function J will be minimised with respect to the variables r_{nk} and $\boldsymbol{\mu}_k$ through a 2-step iterative process, where the first step will minimise w.r.t r_{nk} and the second w.r.t $\boldsymbol{\mu}_k$.

Step 1: update r_{nk}:
Since the minimisation w.r.t r_{nk} is a decision process to minimise $\|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$, this is simply done by assigning each observation $\boldsymbol{x}_n$ to the closest vector $\boldsymbol{\mu}_k$, $k = 1, ..., K$. In other words:

$$r_{nk} = \begin{cases} 1 & \text{if } k = argmin_j \|\boldsymbol{x}_n - \boldsymbol{\mu}_j\|^2 \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

Step 2: update $\boldsymbol{\mu}_k$:
The cost function J is a quadratic function of $\boldsymbol{\mu}_k$, which can be minimised by setting the derivative w.r.t $\boldsymbol{\mu}_k$ to **0** and then solved for $\boldsymbol{\mu}_k$,

$$\nabla_{\boldsymbol{\mu}_k} J = 2 \sum_{n=1}^{N} r_{nk}(\boldsymbol{x}_n - \boldsymbol{\mu}_k) = \mathbf{0} \longrightarrow \frac{\sum_{n=1}^{N} r_{nk} \boldsymbol{x}_n}{\sum_{n=1}^{N} r_{nk}} = \boldsymbol{\mu}_k$$

Thus, the updated $\boldsymbol{\mu}_k$ is simply the mean of all observation belonging to cluster k.

2.3.2 Expectation maximisation

The following text is based on the theory presented in [17]. Expectation Maximisation (EM) is an iterative method for finding maximum likelihood solutions to models with so called latent variables. The number of clusters K is set a priori, a Gaussian distribution is initialised and then EM iterates over a given dataset to find the Gaussian mixture that best describe the dataset.

The likelihood function which will be the subject of the maximisation is:

$$J = \sum_{n=1}^{N} ln(\sum_{k=1}^{K} \boldsymbol{\pi}_k \mathcal{N}(\boldsymbol{x}_n; \boldsymbol{\mu}_k, \Sigma_k)) \tag{2.8}$$

where $\mathcal{N}(\boldsymbol{x}_n; \boldsymbol{\mu}_k, \Sigma_k)$ is the Gaussian distribution of $\boldsymbol{x}_n$ given the parameters mean vector $\boldsymbol{\mu_k}$ and covariance matrix Σ_k. When Equation (2.8) is at a local minimum, the derivatives of J with respect to $\boldsymbol{\mu_k}$, Σ_k and $\boldsymbol{\pi}_k$ all need to be zero.

Derivative w.r.t $\boldsymbol{\mu}_k$:

The derivative of J w.r.t $\boldsymbol{\mu}_k$ is

$$0 = -\sum_{n=1}^{N} \frac{\boldsymbol{\pi}_k \mathcal{N}(\boldsymbol{x}_n; \boldsymbol{\mu}_k, \Sigma_k)}{\sum_{j=1}^{K} \boldsymbol{\pi}_j \mathcal{N}(\boldsymbol{x}_n; \boldsymbol{\mu}_j, \Sigma_j)} \Sigma_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) = -\sum_{n=1}^{N} \gamma(\boldsymbol{z}_{nk}) \Sigma_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \tag{2.9}$$

After some rearranging to solve $\boldsymbol{\mu}_k$ we get:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(\boldsymbol{z}_{nk}) \boldsymbol{x}_n \tag{2.10}$$

where

$$N_k = \sum_{n=1}^{N} \gamma(\boldsymbol{z}_{nk}) \tag{2.11}$$

can be interpreted as the number of observations associated with cluster *k*, and

$$\gamma(\boldsymbol{z}_{nk}) = \frac{\boldsymbol{\pi}_k \mathcal{N}(\boldsymbol{x}_n; \boldsymbol{\mu}_k, \Sigma_k)}{\sum_{j=1}^{K} \boldsymbol{\pi}_j \mathcal{N}(\boldsymbol{x}_n; \boldsymbol{\mu}_j, \Sigma_j)} \tag{2.12}$$

is known as the posterior probability or responsibility.

Derivative w.r.t Σ_k:
By calculating the derivative of J w.r.t Σ_k, setting it to zero and solving it for Σ_k with similar reasoning as previously, we get,

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(\boldsymbol{z}_{nk}) (\boldsymbol{x}_k - \boldsymbol{\mu}_k)(\boldsymbol{x}_k - \boldsymbol{\mu}_k)^T \tag{2.13}$$

Derivative w.r.t $\boldsymbol{\pi}_k$:
Lastly, the derivative of J with respect to the mixing coefficients $\boldsymbol{\pi}_k$ is computed, set to zero and solved for $\boldsymbol{\pi}_k$,

$$\boldsymbol{\pi}_k = \frac{N_k}{N} \tag{2.14}$$

Unfortunately though, there does not exist any closed form solution since the posterior probabilities $\gamma(\boldsymbol{z}_{nk})$ depends on the parameters in a rather complex way. What is possible though is optimisation through an iterative process partitioned into two steps; Expectation and Maximisation (hence the name Expectation-Maximisation). In the Expectation step, the posterior probabilities $\gamma(\boldsymbol{z}_{nk})$, are computed with the parameters $\boldsymbol{\mu_k}$, Σ_k and $\boldsymbol{\pi}_k$. With the newly computed $\gamma(\boldsymbol{z}_{nk})$, the previously used parameters are updated in the so called

Maximisation Step. The complete procedure is described in Algorithm 1.

Algorithm 1: Expectation-Maximisation [40]

Data: N number of observations of the D dimensional random variable $\boldsymbol{x}_n$ and the number of clusters K
Result: A cluster distribution described as a Gaussian mixture $\boldsymbol{x}_n$
initialization;
Choose the number of clusters K, with associated parameters $\boldsymbol{\mu_k}$, Σ_k and $\boldsymbol{\pi}_k$.
while $\delta J > tolerance$ **do**
  Expectation; Compute $\gamma(\boldsymbol{z}_{nk})$ with eq. (2.7)
  Maximisation; Update $\boldsymbol{\mu_k}$, Σ_k and $\boldsymbol{\pi}_k$ with eq. (2.5), (2.8) and (2.9)
end

One could also use a set number of iterations for the loop instead of a convergence condition to get more consistent computing times, which may be desirable when implementing real-time applications.

2.3.3 Density based spatial clustering of applications with noise

The following text is based on the theory presented in [40]. As the name suggests, this algorithm will recognise a cluster in spaces where the observation density is high enough. The key metric for the algorithm is the number of points within a neighbourhood of the point of interest. Due to the fact that DBSCAN is not minimising with respect to a certain function (i.e. minimising the distance to a centroid like the K-means algorithm or fitting a mixture of Gaussian like EM), the algorithm is very versatile when it comes to recognising clusters of different shapes and sizes [40].
In order to explain the algorithm more in depth, a number of definitions will be made. In the text below, the set S is the N number of observations of the D dimensional random variable $\boldsymbol{x}_n \in \mathbb{R}^D$ or in other words, $\boldsymbol{S} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\}$, $S \subseteq \mathbb{R}^D$.

1. $d(\boldsymbol{x}_a, \boldsymbol{x}_b)$ is the Euclidean distance between the two points $\boldsymbol{x}_a, \boldsymbol{x}_b \in \mathbb{R}^D$.
2. The ϵ-neighbourhood of a point $\boldsymbol{x}_n \in S$ is defined as:
   $N_e(\boldsymbol{x}_n) = \{\, \boldsymbol{x} \in \mathbb{R}^D : d(\boldsymbol{x}, \boldsymbol{x}_n) \leq \epsilon \,\}$.
3. An observation are classified as one of the following classes
   (a) An observation $\boldsymbol{x}_n \in S$ is called an internal point if its ϵ-neighbourhood contains more than n_{min} number of other points.
   (b) An observation $\boldsymbol{x}_n \in S$ is called a borderpoint if $0 < |N_e| < n_{min}$ and at least one of its neighbouring point is classified as an internal point.
   (c) An observation $\boldsymbol{x}_n \in S$ is regarded as noise/outlier if it is not classified as either an internal point or borderpoint.
4. Let $\boldsymbol{x}_a, \boldsymbol{x}_b \in S$ be two observations where $\boldsymbol{x}_b \in N_e(\boldsymbol{x}_a)$, $\boldsymbol{x}_a$ is classified an an internal point and $\boldsymbol{x}_b$ is classified as a borderpoint.
   $\boldsymbol{x}_b$ is then said to be *directly density-reachable* from $\boldsymbol{x}_a$. Note however that the reverse is not true ($\boldsymbol{x}_a$ directly density-reachable from $\boldsymbol{x}_b$). Thus, the relationship is not necessarily symmetric.

5. Let $\boldsymbol{x}_a, \boldsymbol{x}_b \in S$ be two observations and there exists a sequence of observations $\boldsymbol{x}_1, ..., \boldsymbol{x}_j$ where $\boldsymbol{x}_a = \boldsymbol{x}_1$ and $\boldsymbol{x}_j = \boldsymbol{x}_b$ and each point $\boldsymbol{x}_{i+1}$ is directly density-reachable from x_i, $i = 1, ..., j-1$. $\boldsymbol{x}_b$ is then *density-reachable* from $\boldsymbol{x}_a$. The relation is, just like direct density-reachability, not necessarily symmetric.
6. Let $\boldsymbol{x}_a, \boldsymbol{x}_b, \boldsymbol{z} \in S$ and let $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ be directly density-reachable from $\boldsymbol{z}$. $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ are then *density-connected*.

Observations belonging to the same cluster generated have the property that they are all mutually density-connected.

Given the clarifications above, DBSCAN is presented in Algorithm 2.

Algorithm 2: DBSCAN

Data: N number of observations of the D dimensional random variable $\boldsymbol{x}_n$, $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ... , \boldsymbol{x}_N\}$ and the parameter ϵ

Result: A partition $C = \{c_1, c_2, ... , c_k\}$, where k is the number of clusters

1: *Initialisation*: Evaluate each point and classify them as one of the three above defined classes.

2: Remove all the points classified as noise/outliers

Draw a line between points P_1 and P_2 (Divides the whole set into two subsets, S_1 and S_2)

3: Connect all the neighbouring internal points with an edge and create a cluster of these neighbouring points

4: Assign the remaining border points to one of the clusters created in step 3.

2.4 Traversable surface estimation

In the following section, theory used when estimating the traversable area is presented. Firstly, RANSAC, which is the key component of the ground estimation, is introduced, followed by least squares fitting on a set of points and computation of a plane based on three points.

2.4.1 Random sample consensus

RANSAC is an iterative method for finding parameters to a pattern which is assumed to exist within an observed set of data which contains noise. In each iteration, the algorithm selects a predefined number of points randomly (the number of points depends on the degrees of freedom of the pattern) and estimates the parameters from the selected points with the assumption that they are inliers. Then, the algorithm analyses how many of the remaining data points which can be regarded as inliers to the currently computed pattern [21]. The biggest advantage of RANSAC is that it is very robust to noise/outliers, although, it may require a higher number of iterations for the algorithm to find the pattern when the data is very noisy. Compared to if one would for example estimate the parameters through, for example, by computing a least squares solution, it would be very sensitive to noise. The drawback though is that it is not exactly deterministic. Since the points are picked at random, the parameters which RANSAC yields may differ slightly every time one runs the script, especially if the number of data points in the set is high.

A simple problem where RANSAC can be utilised is when one has to fit a line to a set of observed 2-dimensional data $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ... , \boldsymbol{x}_N\},\ X \subseteq R^2$ with noise.

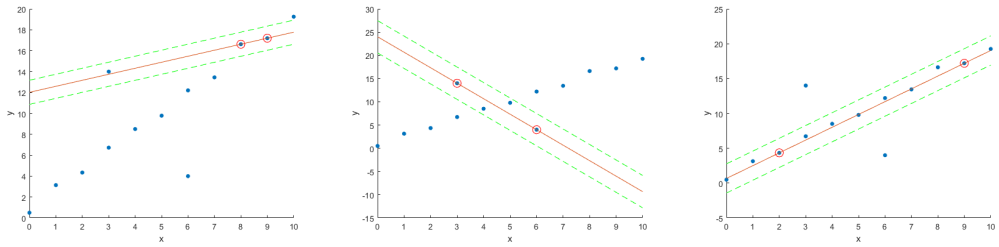


Figure 2.5: An example of three possible outcomes for a RANSAC iteration

To compute a line, one needs to pick two points (if the pattern sought instead was a plane, it would need three points), thus, the number of points picked at random each iteration, in this case, are two. In each subfigure, the points chosen at random are highlighted with red circles, the estimated line in red and the upper and lower thresholds are in green. In the first subplot, the total number of inliers are three (which in this case is a fairly poor result). In the second, the number of inliers are four, which is better than the previous picture but still not very good. Finally, in the third subplot, the algorithm seems to have found a reasonably strong linear pattern.

The general RANSAC algorithm is summarised in Algorithm 3:

Algorithm 3: RANSAC

Data: A data set of observed variables $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ... , \boldsymbol{x}_N\},\ X \subseteq R^n$, a tolerance t and the number of iterations n_{iter}
Result: The parameters $\boldsymbol{\pi}_{best}$ which yield the highest amount of inliers
Initialization; Set the values $\boldsymbol{\pi}_{best} = \emptyset$ and $n_{best} = 0$
for n_{iter} **do**
  Pick the number of required points randomly from X and compute the parameters $\boldsymbol{\pi}$. Check how many points in X which lies within the given tolerance t and assign this value to $n_{inliers}$.
  if $n_{inliers} > n_{best}$ **then**
    $n_{best} \leftarrow n_{inliers}$
    $\boldsymbol{\pi}_{best} \leftarrow \boldsymbol{\pi}$
  end
end

It is common to estimate a new set of final parameters $\boldsymbol{\pi}_{LS}$ by computing the least squares solution from the points which are classified as inliers to $\boldsymbol{\pi}_{best}$, to get an even better estimate of the parameter. This procedure is described in Section 2.4.2.

A question which naturally arises is, how many iterations would one need to perform in order to get an adequate result from RANSAC? One answer is given by the formula,

$$k = \left\lceil \frac{log(1 - P_{success})}{log(1 - w^n)} \right\rceil \tag{2.15}$$

where k is the number of iterations, $P_{success}$ is desired probability of a successful outcome, w is the proportion of inliers and n is the degrees of freedom (which is three for a plane).

2.4.2 Least square solution for plane fitting by singular value decomposition

When fitting a plane $\boldsymbol{\theta} = [a\ b\ c\ d]^T$ to a set of points $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\},\ X \subseteq \mathbb{R}^3$ and $\boldsymbol{x} = [x\ y\ z]^T$, the optimisation problem to solve is:

$$\begin{aligned} \underset{\boldsymbol{\theta}}{\text{minimise}} \quad & f(\theta) = \sum_{i=1}^{N}([\boldsymbol{x}_i^T\ 1]\boldsymbol{\theta})^2 = \sum_{i=1}^{N}(ax_i + by_i + cz_i + d)^2 \\ \text{subject to} \quad & g(\theta) = a^2 + b^2 + c^2 - 1 = 0 \end{aligned} \tag{2.16}$$

The cost function is derived from the formula for the distance between the plane and a point $[\boldsymbol{x}_i^T\ 1]\boldsymbol{\theta}$ and the constraint $g(\theta)$ is needed to exclude the trivial solution $\theta = \mathbf{0}$. After rearranging the cost function and eliminating the variable d in $\boldsymbol{\theta}$, since it is just a scale factor, the minimisation problem can be solved by finding the eigenvector $\tilde{\theta} = [a\ \ b\ \ c]^T$ which corresponds to the smallest eigenvalue in the eigenvector problem:

$$\sum_{i=1}^{N} \begin{bmatrix} \tilde{x}_i^2 & \tilde{x}_i\tilde{y}_i & \tilde{x}_i\tilde{z}_i \\ \tilde{y}_i\tilde{x}_i & \tilde{y}_i^2 & \tilde{y}_i\tilde{z}_i \\ \tilde{z}_i\tilde{x}_i & \tilde{z}_i\tilde{y}_i & \tilde{z}_i^2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \lambda \begin{bmatrix} a \\ b \\ c \end{bmatrix} = M\tilde{\theta} = \lambda\tilde{\theta} \tag{2.17}$$

$$(\tilde{x}_i, \tilde{y}_i, \tilde{z}_i) = (x_i - \bar{x}, y_i - \bar{y}, z_i - \bar{z}),\ \ (\bar{x}, \bar{y}, \bar{z}) = \mathbb{E}(X)$$

This originates from the general optimality condition $\nabla_\theta f(\theta) = \lambda\nabla_\theta g(\theta)$ for the minimisation problem above. This eigenvalue problem can in turn be solved by computing the so called *singular value decomposition* of the M-matrix $M = U\Sigma V^T$ and taking the last column of V which will be the eigenvector $\tilde{\boldsymbol{\theta}} = [a\ \ b\ \ c]^T$ that corresponds to the smallest eigenvalue.
The final plane $\boldsymbol{\theta}$ can then be constructed as:

$$\boldsymbol{\theta} = [\tilde{\boldsymbol{\theta}}^T\ \ d]^T,\ \ d = -(a\bar{x} + b\bar{y} + c\bar{z})$$

2.4.3 Construction of a plane from 3D-points

The degrees of freedom (DoF) for a plane is three, thus three points $\boldsymbol{x} \in \mathbb{R}^3$ are needed if one desires to construct a plane out of points. The equation of a plane is:

$$\tilde{\boldsymbol{x}}^T\boldsymbol{\pi} = 0, \quad \tilde{\boldsymbol{x}} = [\boldsymbol{x}^T\ 1]^T, \quad \boldsymbol{\pi} = [a\ b\ c\ d]^T \tag{2.18}$$

Here, $\tilde{\boldsymbol{x}}$ is the so called homogeneous coordinate. Lets denote the three chosen points in homogeneous form as $\tilde{\boldsymbol{x}}_1, \tilde{\boldsymbol{x}}_2$ and $\tilde{\boldsymbol{x}}_3$. If all of the points belong to the plane, then $\boldsymbol{\pi}$ should fulfil the following system of equations,

$$\begin{bmatrix} \tilde{\boldsymbol{x}}_1^T \\ \tilde{\boldsymbol{x}}_2^T \\ \tilde{\boldsymbol{x}}_3^T \end{bmatrix} \boldsymbol{\pi} \triangleq A\boldsymbol{\pi} = \mathbf{0} \tag{2.19}$$

Thus, to compute the plane $\boldsymbol{\pi}$, one simply has to compute the nullspace of matrix A, which is constructed by the three chosen points.

2.5 Bounding geometry algorithm

A minimum bounding geometry algorithm is a function which creates a minimal area border region, encircling every data point in a cluster. These functions works for both 3D spaces, as well as for 2D Cartesian coordinate systems, depending on area of application. For this thesis a 2D region is of interest with data point vector $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\}$, where the coordinate vector $\boldsymbol{x}_i \subseteq \mathbb{R}^2$. If the perimeter surrounding a cluster has the shape of a rectangle, there is a risk a lot of unnecessary space is within this region and classified incorrectly. The risk increases with the size of vector $\boldsymbol{X}$, therefore the number of elements in vector $\boldsymbol{X}$ determines which fitting method should be utilised. The possibility to extract extra positional information about obstacles comes from these bounding geometry algorithms.

2.5.1 Minimum-area enclosing rectangle

When representing small objects, a minimum-area enclosing rectangle algorithm is well-suited. It is an effective and quick iterating procedure for forming a border region. Firstly, four corner points is defined from the minimum- and maximum values in both x- and y directions for the dataset *S*. The rectangle area is computed and compared with the smallest area found so far to store the least area for next iteration. The cluster is rotated with a predefined angle and the process is repeated until the cluster is rotated $90°$ from original state and all possibilities have been tried. The box-fitting algorithm is described

in Algorithm 4

Algorithm 4: Minimum-Area Enclosing Rectangle

Data: Dataset *S* containing all points belonging to a cluster
Result: Minimum area rectangle region
i = i-th rotation of the dataset *S*
n = maximum number of rotations
while $i \leq n$ **do**
  x_{min} = minimum x-coordinate in cluster
  x_{max} = maximum x-coordinate in cluster
  y_{min} = minimum y-coordinate in cluster
  y_{max} = maximum y-coordinate in cluster
  $Area = (x_{max} - x_{min})(y_{max} - y_{min})$
  if $Area < Area_{best}$ **then**
    $Area_{best} = Area$
  end
  i++
end

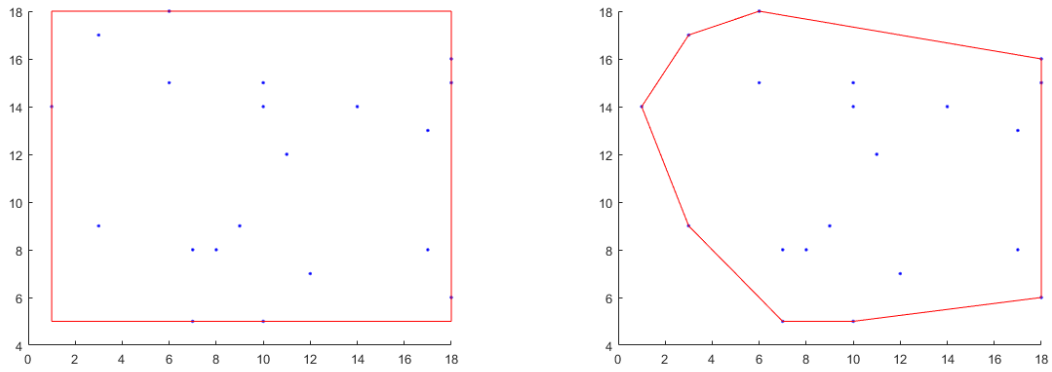
2.5.2 Convex hull

Convex Hull is a method for defining vertices in a polygonal bounding geometry for a given dataset. The output is the smallest possible convex region containing all input data points, which means that a line joining two arbitrarily selected data points never intersects with the region's border. Hence, the convex region set X is a subset to input dataset S, $X \subseteq S$ [27]. Quickhull is an algorithm, of algorithmic paradigm sort divide-and-conquer, for computing a convex region recursively. A divide-and-conquer algorithm breaks a problem down into two or more sub-problems that are smaller instances of the same problem, solve these recursively one by one until the results can be merged together to form a solution for the initial task [31]. The pseudocode for quickhull is shown in Algorithm 5

Algorithm 5: Quickhull

Data: Dataset S containing all points belonging to a cluster
Result: Convex region
Find point P_1 with minimal x-coordinate and add to convex hull vector
Find point P_2 with maximal x-coordinate and add to convex hull vector
Draw a line between points P_1 and P_2 (Divides the whole set into two subsets, S_1 and S_2)
while *points outside convex region exists for* S_1 **do**
    Find the point with maximal euclidean distance to the line and add to convex hull vector at position between the line endpoints
    Draw new lines from recently discovered vertex to its neighbour points in convex hull vector
end
while *points outside convex region exists for* S_2 **do**
    Find the point with maximal euclidean distance to the line and add to convex hull vector at position between the line endpoints
    Draw new lines from recently discovered vertex to its neighbour points in convex hull vector
end
Merge together vertex vectors from subsets S_1 and S_2

A comparison between the two different bounding geometry algorithms are visualised in Figure 2.6, for an example dataset.



(a) Geometry fitting using rectangle algorithm

(b) Geometry fitting using Convex Hull

Figure 2.6: Geometry fitting algorithms for an arbitrary dataset

2.6 Theory for analysis of LiDAR mounting point

This section is dedicated for explaining how the key metrics used in the LiDAR mounting point investigation are computed, which consists of the maximum displacement in one frame and homogeneous transformation matrix which is used to compute the states for each mounting point.

2.6.1 Scan for maximum LiDAR displacement in one frame

The dynamics that govern the distortion of the LiDAR frames are changes in the LIDAR's states $\boldsymbol{x}$ (position) and $\boldsymbol{\Theta} = (\theta, \phi, \psi)$ (angles) while the LiDAR is performing a scan. Thus, inherently, the states $\dot{\boldsymbol{x}}$ and $\boldsymbol{\omega} = \dot{\Theta}$ are of great interest, but they do not convey how much a frame in the end actually is distorted.
That is why a script was created to scan the collected data, given the time to perform one LiDAR scan, for the maximum change in the states $\boldsymbol{x}$ and $\boldsymbol{\Theta}$:

The algorithm is presented in Algorithm 6:

Algorithm 6: Max change scan

Data: The measurements of the state of interest $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_t, \dots, \boldsymbol{x}_{t_f}\}$ where t is the time index, the index difference $\delta i = \frac{t_{frame}}{\delta t}$ where δt is the time between the measurements and t_{frame} is the time to perfom one LiDAR scan.

Result: The maximum possible change in state $\delta \boldsymbol{x}_{max}$ which can occur in the given measurements

Initialization; Set the value and $\delta \boldsymbol{x}_{max} = 0$

for $t = 1, \dots, t_f$ **do**
  $\delta \boldsymbol{x} \leftarrow |\boldsymbol{x}_t - \boldsymbol{x}_{t+\delta i}|$
  if $\delta \boldsymbol{x} > \delta \boldsymbol{x}_{max}$ **then**
    $\boldsymbol{x}_{max} \leftarrow \delta \boldsymbol{x}$
  end
end

2.6.2 Homogeneous transformation matrix

A homogeneous transformation matrix is a utility for body frame representation in a consistent way and allows easy transformations between two body frames. The homogeneous transformation matrix T consists of a combination of the rotation matrix $R \subseteq \mathbb{R}^{3\times3}$, the translation vector $\boldsymbol{t} \subseteq \mathbb{R}^{3\times1}$, the zero vector $\mathbf{0}^{1\times3}$ and the homogeneous coordinate $1 \subseteq \mathbb{R}$ [19]:

$$T = \begin{bmatrix} R & \boldsymbol{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.20}$$

where,

$$R = \begin{bmatrix} cos\phi_i & -sin\phi_i & 0 \\ sin\phi_i \cdot cos\theta_{i-1} & cos\phi_i \cdot cos\theta_{i-1} & -sin\theta_{i-1} \\ sin\phi_i \cdot sin\theta_{i-1} & cos\phi_i \cdot sin\theta_{i-1} & cos\theta_{i-1} \end{bmatrix} \tag{2.21}$$

and that,

$$\boldsymbol{t} = \begin{bmatrix} a_{i-1} \\ -sin\theta_{i-1} \cdot d_i \\ cos\theta_{i-1} \cdot d_i \end{bmatrix} \tag{2.22}$$

In Figure 2.7 the transformed frame vector $\boldsymbol{v'} \subseteq \mathbb{R}^{4\times1}$ is described with T generated by the following procedure [26]:

1. Translate d_i along the z-axis
2. Rotate counterclockwise by ϕ_i about the z_i-axis
3. Translate by a_{i-1} along the x_{i-1}-axis
4. Rotate counterclockwise by θ_{i-1} about the x_{i-1}-axis

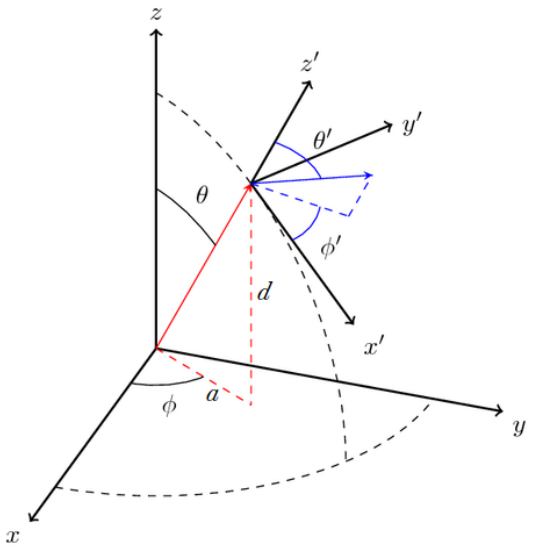


Figure 2.7: Transformed frame $\boldsymbol{v'} \subseteq \mathbb{R}^{4\times1}$ described from the reference frame $\boldsymbol{v}$, such that $\boldsymbol{v'} = T\boldsymbol{v}$.

3
Method

In this chapter, the methodology used in the thesis is presented. It initiates with a brief clarification of the utilised tools and software and then move forth to methodology for the thesis's two main tasks, LiDAR placement investigation and the development of the object detection algorithm.

3.1 Tools and software

Robot Operating System (ROS) is a framework for robotics and is described by the ROS development team as follows: "ROS is a flexible framework for robotics. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms." [9]. With ROS a two-way communication protocol can be established between the LiDAR sensor and the Linux based computer used for algorithm computations, in a simple and effective manner. ROS provides easy to use commands for initiating the data collection and automatically calling the node that accommodates the OD-algorithms. A computer running on a Linux kernel is preferable due to compatibility issues for other operating systems.

MATLAB is a useful tool for algorithm development. The number of libraries and functions are extensive, thus very helpful for implementation and testing of initial functions. The output can easily be evaluated numerically, analytically and visually using both integrated tools and functions.

Python is a user friendly programming language for integrating components in a real-time system. It is a open-source software, which means it is very extensive and flexible. This means that functions can be written in a short period of time. Conversion between MATLAB code and Python code is fairly effortless thanks to the Python library NumPy. NumPy allows vectorisation and faster operating code. SciPy is another convenient library for mathematical calculations and it also contains resolved functions for clustering algorithms.

3.2 LiDAR placement

In this section, the method for evaluating the LiDAR's suitability in a test track environment is presented. It is divided into two parts, one quantitative and one qualitative.

3.2.1 Disturbance analysis

The placement of the LiDAR is very important because the LiDAR scans may be influenced by the car's movement when driving on uneven roads. Three different positions are taken into consideration, the roof, the hood and the bumper (Figure 3.1). For simplicity, all positions are located without an offset in the lateral direction. The coordinate system can be seen in Figure 3.2.
Alteration of the position $\delta\boldsymbol{p}$ and angles $\delta\boldsymbol{\Theta}$ during a scan is what causes distortion of the frame. $\delta\boldsymbol{p}$ and $\delta\boldsymbol{\Theta}$ in turn originates from the movement of the LiDAR, $\boldsymbol{v}_{LiDAR}$, $\boldsymbol{\omega}_{LiDAR}$ which is attached to the body of the car. In order to obtain data about these velocities and positions, a high performance GNSS/INS will be used to measure the car's states from the position of the GNSS/INS; velocities $\boldsymbol{v}_{GNSS/INS}$, angles $\boldsymbol{\Theta}$ and angular velocities $\boldsymbol{\omega}$ (there is no need for the GNSS/INS subscript on the angles and angular velocities since they will be the same on every point on a rigid body). These measurements will then be used to calculate the velocity $\boldsymbol{v}_{pos}$ and position $\boldsymbol{p}_{pos}$ for the three positions of interest with the theory described in Section 2.6.

One modification is made to the measured data, which is setting the forward velocity of the GNSS/INS to zero (x-direction). This is because the forward velocity has got a high influence on the final speed of the placements, and the forward velocity will not differ significantly between the positions of interest.
The velocity $\boldsymbol{v}_{pos}$ will then be used to compute the speed of the position

$$s_{pos} = \|\boldsymbol{v}_{pos}\|$$

and a mean and max value will be computed for s_{pos} to describe the distribution.
The position $\boldsymbol{p}_{pos}$ will be scanned for maximum displacement $\delta\boldsymbol{p}_{pos}$ according to Section 2.6.1.

The GNSS/INS data will be collected from eight different trails at either the durability track or comfort track, and in two or three different speeds depending on the maximum allowed speed on the specific trails. The track trails have assorted kinds of road disturbances with unique characteristics that will cause rotations in all three dimensions.

There are also qualitative aspects to take into consideration when it comes to placement, such as visibility, range and disturbances caused by vibrations in the car. These aspects will be evaluated in a more subjective qualitative manner.

3.2.2 Analysis of the placement properties

It is not possible to evaluate if the LiDAR will be able to handle the disturbances from analysis of the velocities and displacements alone, it will only convey how the different placements compare to each other. That is why a subjective analysis of the scans will be done in order to assess if the LiDAR scans are too distorted to use, as well as feeding it into the algorithm to investigate if it will cause any new problems.
The outcome of this inspection should of course only be treated as guidance. In order to truly be able to say something about the amount of distortion the LiDAR experiences during a rough drive, a ground truth is needed (i.e., a scanned frame with no distortion).

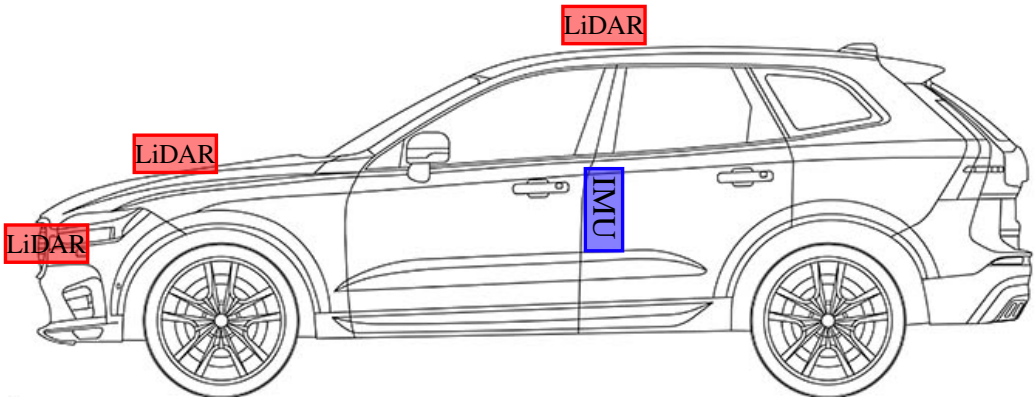


Figure 3.1: Three different LiDAR mounting points for evaluation relative the GNSS/INS in two dimensions. The mounting points are the roof, the hood and the bumper.

Figure 3.2: The GNSS/INS's approximate location together with its coordinate system. The GNSS/INS is placed as close to the centre of the car body as possible, just above the mid console and between front and back seats.

3.3 Development process of the OD-algorithm

The development of the object detection algorithm is an iterative process, starting with the construction of an initial version. This version will then go through a cycle of four phases; Testing, Evaluation, Research and Implementation, in order to produce new and improved versions. When a version of the algorithm is estimated as good enough, it will be integrated directly with the LiDAR in the Deployment phase,

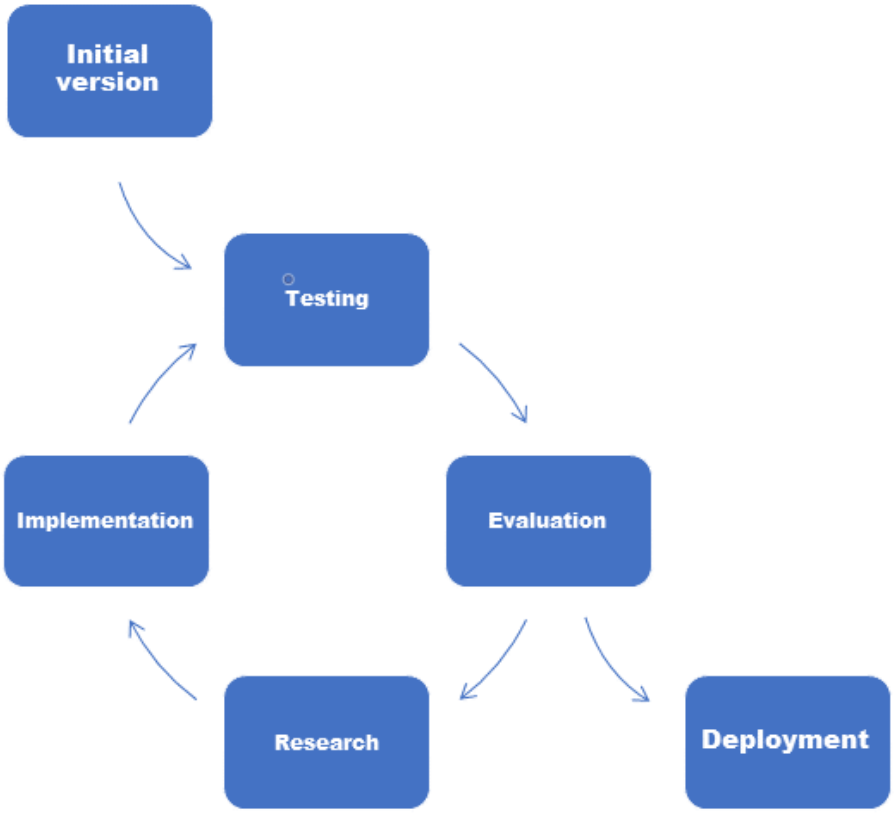


Figure 3.3: A flow chart of the development process.

3.3.1 Initial version

As outlined in *Objectives* section, the plan is to perform object detection by ground plane extraction with RANSAC and then clustering. When constructing the initial algorithm for the object detection, three clustering methods will be evaluated before selecting one which is deemed suitable.

3.3.1.1 Estimation of the ground plane

The first of the two sub-problems described in Section 1.4.1 is to estimate which points in a given frame $X = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\}$, $X \subseteq \mathbb{R}^3$ belongs to traversable surface X_D. This process will be referred to as *ground plane estimation* (GPE). The chosen approach to solve this problem is by estimation of planar surfaces through RANSAC described in Section 2.4.1. The reason RANSAC is chosen for the GPE process is because it is a relatively robust method to find the parameters of a pattern in data which contains a lot of noise. Noise in this case could be anything that does not follow a planar surface pattern such as objects on or alongside the road or uneven terrain.
The general process for the GPE will then be:

1. Perform RANSAC on the given set of points X for one frame.
2. Classify all points which lie within the tolerance of the obtained RANSAC-plane as X_D and all the other points as X_{ND}.

3.3.1.2 Selection of cluster algorithm

In the second of the two sub-problems, objects are to be identified in the set of non-traversable points X_{ND} estimated earlier. The chosen approach to solve the problem is by cluster analysis of the points X_{ND}.
Three different cluster algorithms are chosen and will be evaluated to see which is the most suitable for the task.
The following algorithms will be constructed in MATLAB and evaluted:

- Expectation-Maximisation
- K-means
- DBSCAN

Three point cloud sets will be used in the evaluation, where two of the sets are taken from individual frames in the scenes listed below with X_{ND} extracted and one which is from the course *Computer Vision* (course code EEN020. The motivation behind these scenes is the variance in number of points and setting:

Computer Vision Set: A set containing 797 points which is a 3D reconstruction of a stereo pair of images picturing the corner of a house.

Durability scene: A set containing 3564 points taken from a frame from the durability track scenario described below which has gone through GPE.

Pedestrian scene: A set containing 9042 points taken from a frame from the highway scenario described below which has gone through GPE.

The three cluster algorithms will then be applied on the three above listed sets where an average run time will be calculated over 10 runs for each algorithm and dataset, as well as maximum and minimum run time.
As described in Section 2.3.2 and Section 2.3.1, the EM and K-means algorithm needs a predefined number of clusters K and initial distribution of the clusters. The initial distribution will be created by distributing the K cluster uniformly over the scene. As for DBSCAN, the parameters will chosen by experimenting with different values until a reasonable end result was achieved. The parameters in DBSCAN does not affect computational time noticeably, which makes it possible to tune the parameters to optimise the end result only.
Apart from quantitative results, general advantages and disadvantages of the algorithm will be taken into account when choosing an algorithm.

3.3.2 Testing

In the testing phase, the algorithm is applied on a number of different testing scenarios in order to collect performance data and discover potential problems. The following scenarios will be recorded with the LiDAR in 10Hz and 30Hz, converted in Python and imported

to MATLAB as sequences of point cloud frames:

- **Durability Track**: A very rough and bumpy track at HPG with purpose of testing the car's durability. Here, there are two scenarios available:
  - **Human target**: A scenario where a human target doll of adult size is placed laying down in between some rather rough road with bumps. The recording starts at a distance of approximately 50 meters from the target and approaches it at a speed of 10 km/h. The purpose of this scene is to investigate if the algorithm is good enough to for example detect a passed out person in a challenging environment.
  - **A clean run**: This scenario is a bit longer than the human target and is an ordinary run of the track. The purpose is the examine two things; if the excitation of the car has any noticeable impact on the algorithm and if any of the various bumps are classified as objects.
- **Big Road Bump**: A scene containing one big road bump with the approximate size of 1.2 meter in height, 14 meters in length and 3.5 meter in width. This scene is especially challenging for the GPE since the traversable surface is deviating a lot from the standard road and thus risking to be classified as an obstacle.
- **Skidpad**: A big open area where it is possible to place targets without obstructing other cars performing tests. There are two scenarios available.
  - **Two human targets standing**: A scenario where two human targets, one adult size and the other size of a child, placed at a distance of approximately one meter from each other and approached by the LiDAR mounted car at a distance of 200 meters up until about 5 meters at about 40 km/h. The scenario aims to investigate when or if the algorithm will cluster the two targets together as one target, or when it recognises both of them.
  - **Traffic cones**: Small traffic cones with a height of 30 cm was placed at a distance of 200 meters and approached by the car at 40 km/h. Purpose was to examine at what distance the algorithm would recognise small objects.
- **Highway** A highway environment with one scene which contains a parting rail in the middle, where a person is walking towards the car with some targets in the background. This scene was not recorded by the authors, thus, information about it is limited. Still, it has proved to be a good scene to test the algorithm on various objects such as the human, the targets and the parting rail.

What will be regarded as an obstacle and what is not, is defined below. The definitions are based on intuition of what is and what is not regarded as an object and how much of a hazard the potential obstacles could pose.

- **Obstacles**: Anything which one intuitively would not consider ground, e.g humans, vehicles, animals, debris, buildings, vegetation (such as trees or bushes) and etc.

- **Traversable area**: Area belonging to the ground which may be traversable by the vehicle such as the road, and open spaces where it is safe to drive a vehicle. The algorithm for GPE wwill not be able to distinguish whether the extracted ground is paved road or for example a meadow if they both are sufficiently planar. One might think that a meadow is suitable for driving, but if it is not obstructing a path on the road, it does not pose any threat either. Hence, the rather inclusive definition.

- **Non of the above - Grey area**: It is not always obvious what should be regarded as traversable area. One example would be a ditch at the side of the road. The algorithm may find a planar pattern and classify it as traversable area, but it is not exactly intended for driving. As with the example with the meadow, it does not pose a hazard on the road either, which would make an obstacles classification not entirely correct either. Thus, a classification of a grey area as either traversable or non-traversable will not be regarded as incorrect.

3.3.3 Evaluation

In the evaluation, the following questions are considered:

- **How is the performance and the quality of the output when tested on the scenarios above?** Some of the following requirements below was developed in parallel with the work since the properties of the object detection and associated problems was not known at first. It is worth mentioning that these are not hard requirements and may change in between iterations.

  - **A computational time below 1 second**: Volvo has got a desired response time of 100 ms for the whole driverless solution algorithm, from detection to action (which would leave even less time for detection). This requirement may be hard to fulfil, considering that the authors of the thesis have little experience of code optimisation and are not fluent in fast programming languages such as C++, which is a common language within the field. The hardware specifications in terms of computational power for a final version of the driverless vehicle is not clear either, which unarguably have a significant impact on computational times. Thus, the desired limit of 100 ms is instead translated to roughly 1 seconds in MATLAB. This is not a hard limit though and may change if it assessed as unattainable.

  - **Keep false and omitted detections at a minimum**: Precision and recall is most commonly used to calculate mAP (described in Section 2.2.3) when evaluating the performance of various CNN's trained for object detection. Since evaluation is usually made with some commercially available dataset with ground truth (KITTI or NUscenes are examples) which is evaluated by some algorithm, the mAP is computed relatively fast and on a lot of scenes. Unfortunately, this is not possible to do in this thesis because the algorithm developed functions very differently from CNN's, especially what is and what is not regarded as an object. CNN's have very clearly defined labels on what it can detect and not (and thus what is regarded as objects), while the definition

of objects/obstacles in Section 3.3.2 is not defined as sharply. Thus, the metric studied was instead the number of false detections (noise) and omitted detections according to the thesis's definition, see Section 3.3.2, with an ambition to keep these at a minimum.

  - **Consistent partitioning of the clusters**: An important aspect of the clustering algorithm is how it chooses to partition the objects. It may for example decide that two objects which are very close together, or even entangled, belong to the same object, which could be the case when clustering vegetation. Or, it may instead choose to partition one big object into several smaller. In any case, it is desired that this partitioning is consistent since inconsistent partitioning may cause problems later if one would for example decide to feed the output data from the OD-algorithm to a Multi Target Tracking (MTT) process. This requirement does not have a defined metric to relate to, but an effort will be made to minimise this aspect, or at a minimum, keep it at bay.

- **Are there any scenes or specific cases which are associated with problems?**
  - How much impact does these problems have?
  - What could these problems originate from and are they possible to solve?
- **Is the algorithm good enough to be considered for Deployment?** Are the algorithm, considering all the above questions above, mature enough for deployment?

3.3.4 Research

In this phase, detection problems or lack of performance is studied more in depth. First, an analysis of the cause of the problem is done. For example, if computational times are the problem, the code is examined more in detail to find parts of the code which require the most computational time and analyse the reason why.
After the analysis of the cause, an investigation is made in order to examine if others have encountered similar problems and if and how they managed to solve it.
Lastly, ideas are generated and evaluated with the previous steps taken into account which hopefully will yield a contribution for the next phase.

3.3.5 Implementation

This phase is simply for implementing the solution or idea decided upon in the Research step which will be done in MATLAB.

3.3.6 Deployment

When the algorithm is assessed as good enough in the Evaluation phase, the code will be translated and implemented to work in conjunction with the LiDAR through ROS. The key functionality used in ROS is the ability to enable communication between hardware units, in this case the computer and the LiDAR. The object detection algorithm designed

in MATLAB will be written as a so called node; a script which runs independently and communicates with other Python scripts.

4
Results

This chapter will present results from testing the clustering algorithms and will be the basis when evaluating and determining the favourable clustering algorithm. Results for the Investigation with computations for the LiDAR placement and suitability are also given, likewise output from the deployed algorithm when feeding it with several frames from different scenarios.

4.1 Cluster algorithm selection

In this section, the computational times for the clustering algorithms are presented as well as advantages and disadvantages of the algorithms.

4.1.1 Computational results

In the tables below, results for the computational times for the algorithms and scenes described in Section 3.3.1.2 are presented.

Table 4.1: Computer Vision Set

	DBSCAN	K-means	EM
Average time(s)	0.1344	0.0169	8.2107
Maximum time(s)	0.19	0.027	8.302
Minimum time (s)	0.109	0.014	8.119

Table 4.2: Pedestrian scene

	DBSCAN	K-means	EM
Average time(s)	3.7613	0.1394	163.6915
Maximum time(s)	4.187	0.158	200.473
Minimum time (s)	3.616	0.124	149.07

Table 4.3: Durability scene

	DBSCAN	K-means	EM
Average time(s)	0.67892	0.1329	42.8662
Maximum time(s)	0.72	0.152	46.297
Minimum time (s)	0.597	0.125	41.617

4.1.2 Qualitative findings

When performing the tests above, an analysis of the clustering methods end result was done simultaneously. The findings are presented below. The summary is that DBSCAN has got some desired properties that the other algorithms do not possess and that K-means and EM both yield similar clustering results.

DBSCAN is able to filter out noise effectively: One of the biggest advantages of the DBSCAN algorithm is its ability to filter out potential noise. The noise in this context could for example be a misreading from the LiDAR, points which belong to the ground but is not classified as ground and reflections from signs which create "ghost points" (which was a palpable problem in some of the scenes). The noise often had a big impact on the properties of the K-means and EM clustering results, where obvious noise was assigned a cluster would alter the cluster properties significantly.

DBSCAN yielded more consistent partitioning: K-means and EM would often yield a result where it would partition an object (i.e. a person, target, etc) into two or more clusters. This was not the case for DBSCAN which, given easy objects such as those mentioned before, would easily recognise a single cluster. There were instances though where DBSCAN would partition objects into several clusters, for example the parting rail in the Highway scenario, but then again, so would K-means and EM.

DBSCAN would yield more consistent clustering: The K-means and EM algorithms initialisation had a significant impact on the clustering results, which could yield widely different clusters depending on if they were initialised in a uniform pattern of randomly chosen points (note though that the computational times was measured with uniform initial distribution). This was also the case when running K-means and EM through each frame sequentially in a scenario. The DBSCAN algorithm had similar problems, especially with objects far away, but not to the same extent.

4.2 LiDAR placement and suitability

In this section, the result from the LiDAR placement analysis is presented. As described in Section 3.2.1, the collected IMU data was used to compute the LiDAR's states for a set of three different positions. These position states were then analysed in terms of mean and max velocities, and maximum displacement for one frame which assumes a LiDAR frame rate of 10Hz.

4.2.1 Disturbance analysis results

The figures below presents some results for the disturbance analysis. Figure 4.1 displays the distribution of the LiDAR's speed for the three placements in terms of mean and maximum value while the Figure 4.1 displays the obtained maximum displacement for the three placement.

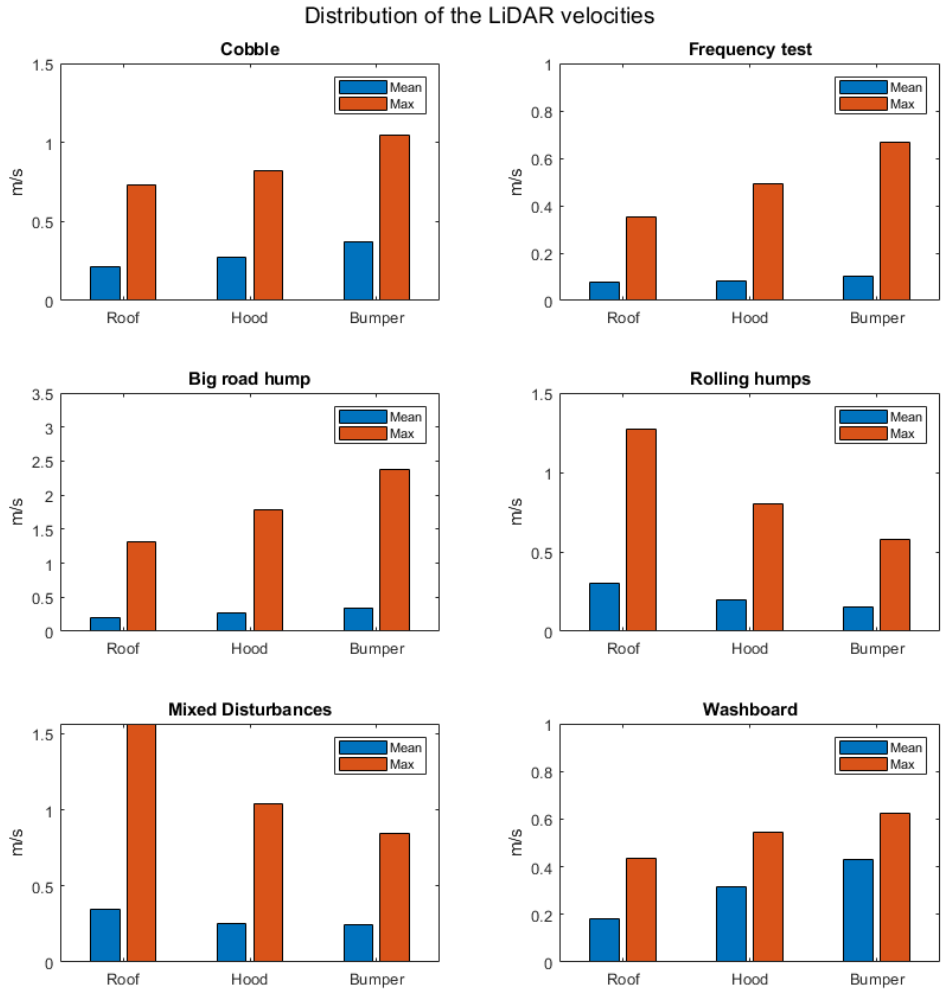


Figure 4.1: The distribution of the LiDAR velocities in terms of mean and maximum for 6 out of 14 scenarios collected at HPG. Lower values are better. For the complete set of data see Appendix A

Figure 4.2 displays the maximum displacement for the LiDAR sensor for the same six scenarios as in Figure 4.1.

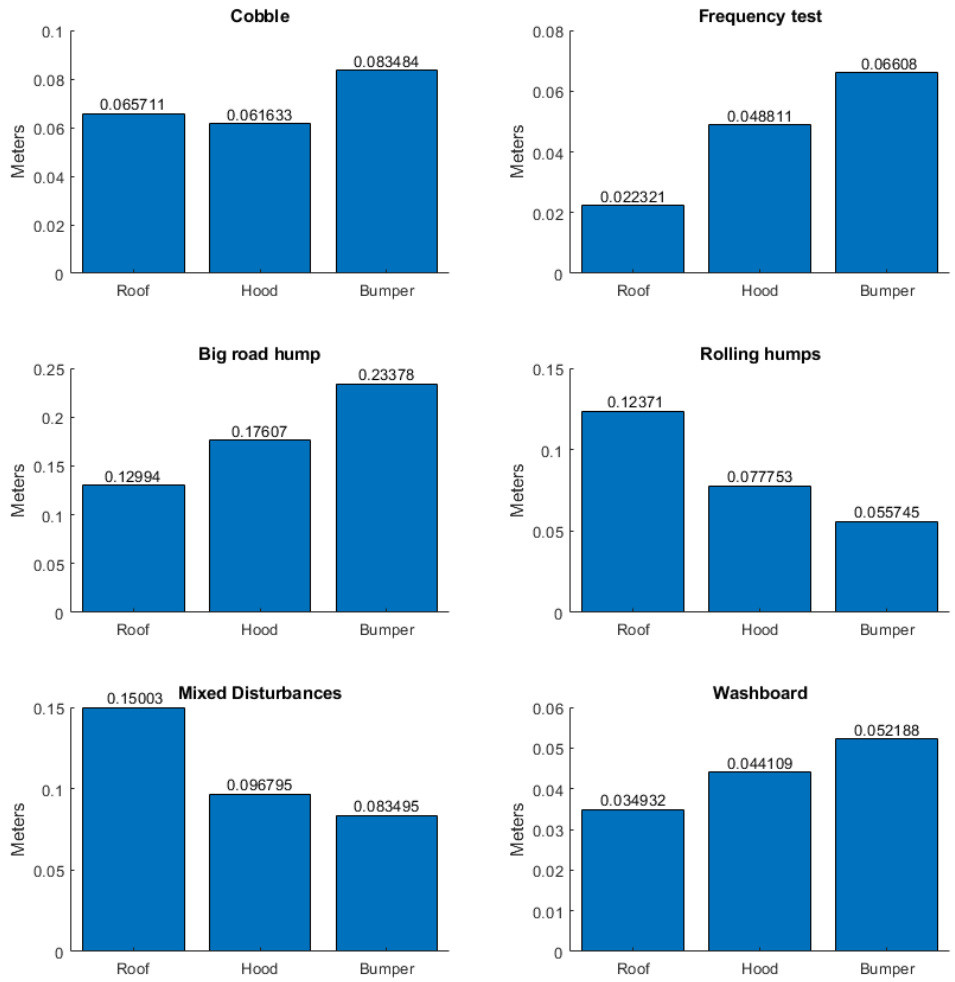


Figure 4.2: Maximum computed displacements for the three locations in 6 out of 14 scenarios collected at HPG. Lower values are better. For the complete set of data see Appendix B

4.2.2 Inspection of the scans

Over 200 frames from trails with rough road was collected, used in the development of the algorithm and inspected. Unfortunately though, is not very convenient to include all of these frames in the report and when inspecting the frames from scenarios with a lot of road disturbances, none of the frames seemed to suffer from distortion. Since every inspected frame seemed to lack distortion, only one frame is included for demonstration and is shown from two different perspectives in Figure 4.3 and Figure 4.4.

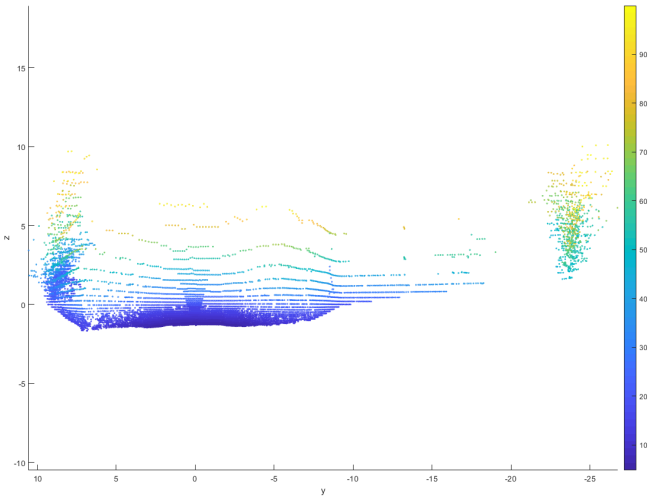


Figure 4.3: A frame from one of the worst scenarios disturbance wise from a horizontal view.

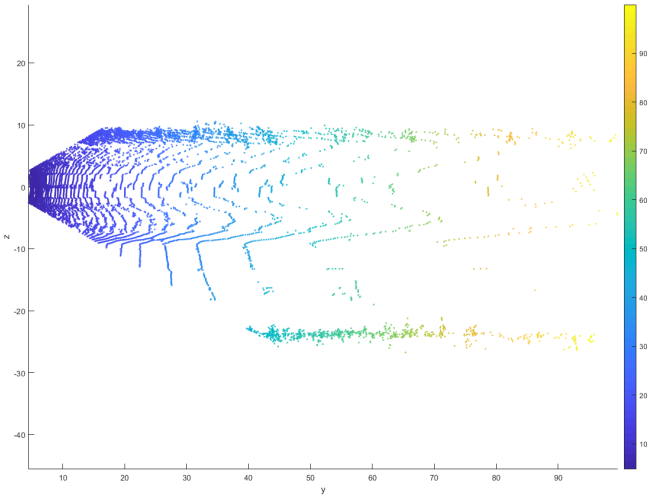


Figure 4.4: A frame from one of the worst scenarios disturbance wise from a top down view.

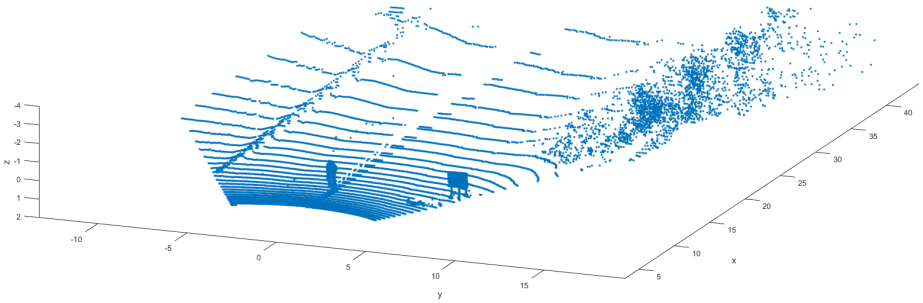
No apparent problems, which could be traced to distortion, was encountered when applying the object detection algorithms on frames from trails with rough road.

4.3 Final algorithm

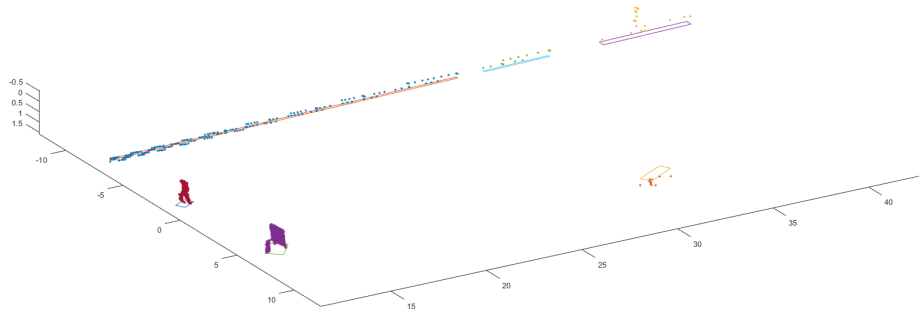
Hundreds of LiDAR frames was collected and processed by the final algorithm (which is presented more in depth in the next chapter) and it is not possible to include all of them in the report. The frames presented below are chosen to demonstrate some of the

characteristics and functionality of the algorithm.
The computational time for the algorithm varied between 0.5-2 seconds, depending on the parameters and the frame, but it was possible, with the right set of parameters, to get consistently below 0.7 seconds in computational time.

Below is an extracted frame (Figure 4.5a) from the pedestrian scene set, is fed as input to the DBDL-algorithm and Figure 4.5b shows corresponding output.



(a) Raw point cloud data from LiDAR



(b) Output from the final algorithm showing remaining data points representing obstacles

Figure 4.5: Figure 4.5a shows input data and Figure 4.5b shows DBDL output

Highly reflective objects such as road signs produces a lot of incorrect data points [10]. Figure 4.6 shows this issue in the LiDAR point cloud when passing a road sign.

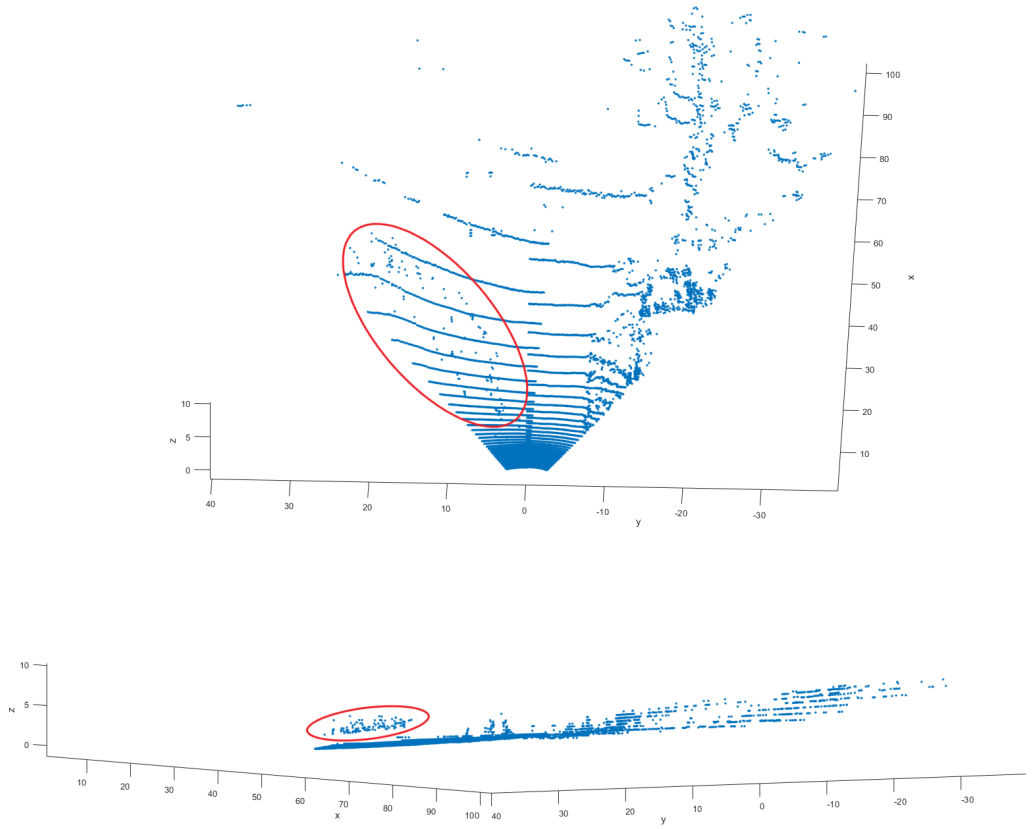


Figure 4.6: Reflective noise, highlighted in red.

Figure 4.7 displays the result of testing how small objects the DBDL can detect by performing the test scenario "*Skidpad*" described in Section 3.3.2. DBDL are able to classify the cones as obstacles first at a distance of 5 meter (with current parameter tuning). The red points corresponds to the obstacle cluster and the blue data points belongs to the ground plane according to GPE.

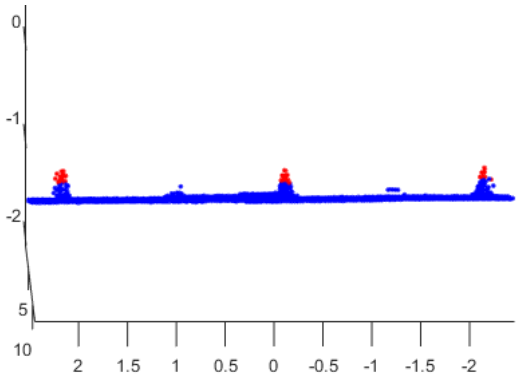


Figure 4.7: Detected traffic cones and classified as obstacles

The algorithm shows weaknesses when an object extends far into the longitudinal direction (Figure 4.8). In this example, the algorithm classifies the parting rail as several objects instead of one.

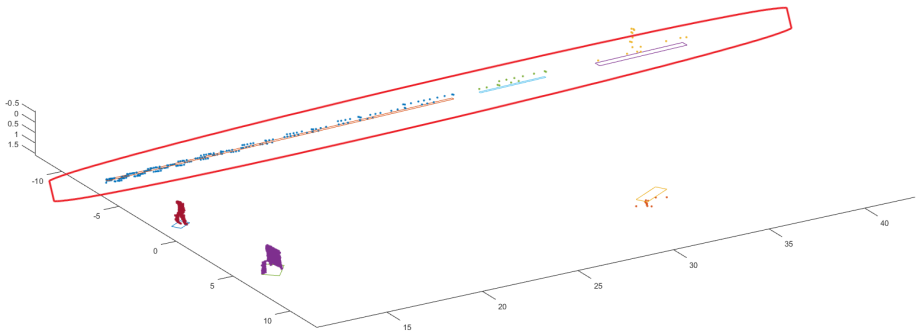


Figure 4.8: An example of when an object, in this case a parting rail, is classified as more than one object (highlighted in red).

The geometry fitting algorithm produces a bounding area for each obstacle and extracts information about the location for the vertices in the clusters. Figure 4.9 shows the characteristics of each fitting method and how the size of the cluster determines fitting method.

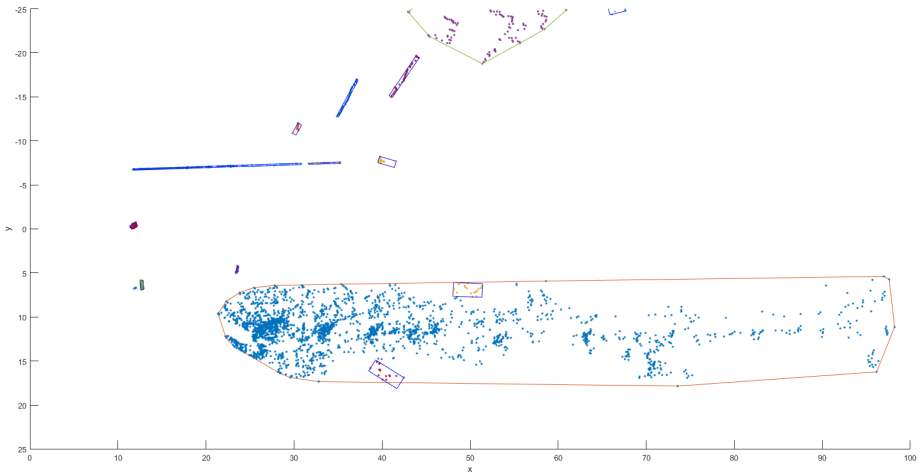


Figure 4.9: A birds eye view of a clustered frame. Smaller objects, such as those in the front are bounded by a box, while bigger cluster, such as the bottom one, is bounded by a convex hull.

Figure 4.10 demonstrates shortcomings of the ground plane estimation. In this case, the ground on the side of the road is too complex for the multi regional RANSAC, and some of the points, which clearly belongs to the ground, are classified as non-traversable.

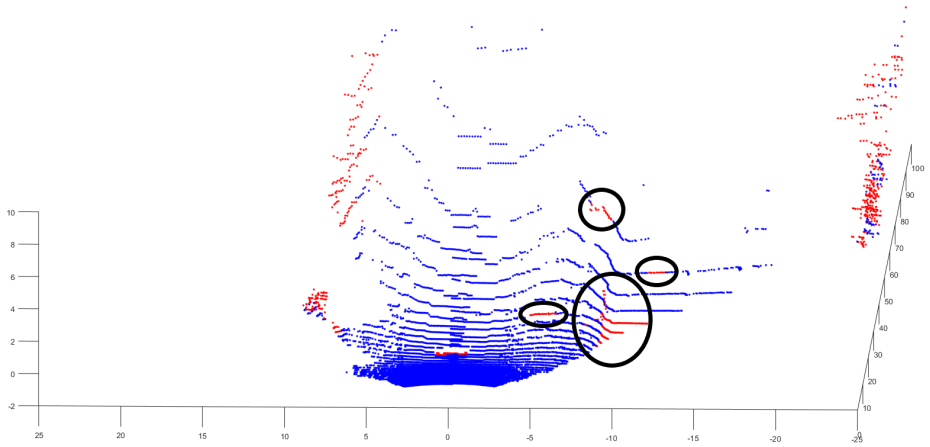


Figure 4.10: False detections arising due to shortcomings of the ground plane estimation

5

Final algorithm, Density based detection for LiDAR

This chapter is dedicated to the resulting final algorithm which is named Density Based Detection for LiDAR (DBDL). First, there is a motivation for the chosen cluster algorithm and then a more detailed explanation of the full algorithm is presented.

5.1 Cluster algorithm selection

With the quantitative and qualitative results, as well as the algorithms known properties (mentioned in Section 2.3) taken into account, the final choice of cluster algorithm was DBSCAN. The arguments behind this choice is:

- The number of clusters does not need to be set beforehand.
- It does not need initial clusters.
- It is relatively robust to noise.
- Less and more consistent partitioning of objects.
- More consistent clustering.

The drawback of DBSCAN though would be its slightly higher computational cost compared to K-means and its computational complexity which is proportional to $\mathcal{O}(n^2)$ where n is the number of points, which resulted in a lot of effort spent on creating solutions which aimed to reduce the number of points before the DBSCAN call.

5.2 The algorithm in detail

A short summary of the chosen components in the final algorithm and recap of their functionality:

- **Preprocessing filters**: filtering out unnecessary data points to scale down the computational time at the cost of a more narrow field of view (FoV).
- **Multiregional RANSAC**: Performs RANSAC to find the ground planes in $m \times n$ regions
- **Pre-clustering filtering**: Decreasing the number of data points in the dataset to increase the speed of the DBSCAN algorithm.

- **DBSCAN**: Creates clusters by finding neighbour data points within a certain threshold.

- **Bounding Geometry Fitting**: Finds the vertices for each cluster to specify obstacles.

A quick overview for the final algorithm is displayed in the flow chart in Figure 5.1.

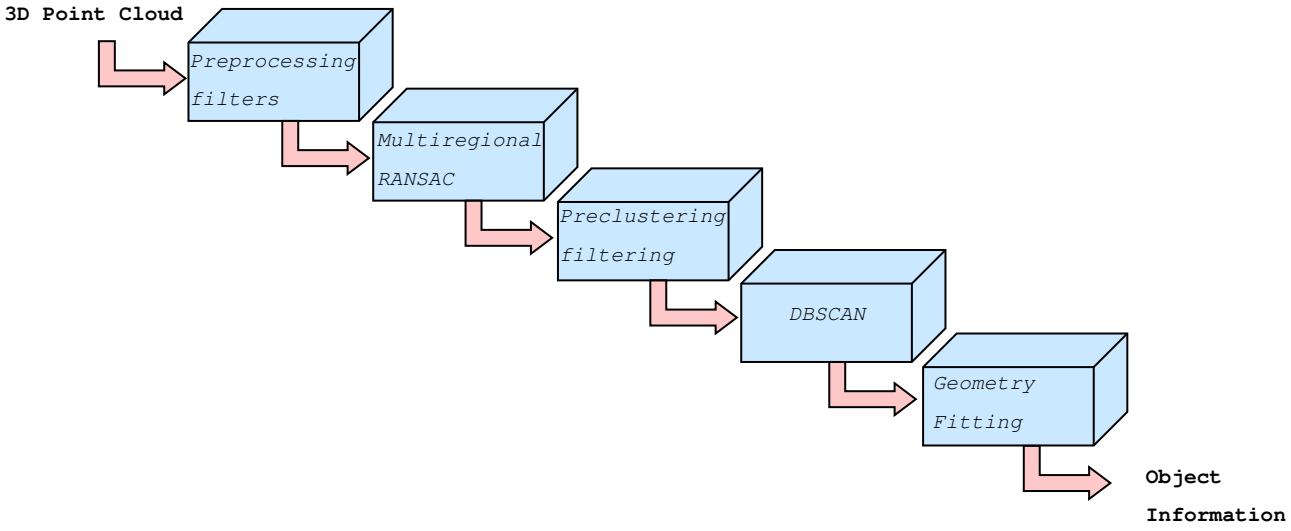


Figure 5.1: Comprehensive overview of the components in the deployed algorithm.

5.2.1 Preprocessing filters

Not all points in a LiDAR scan will be of interest for the algorithm and some are therefore removed in preprocessing. The initial filter consists of limits for azimuth angle ϕ, elevation angle θ and distance from the sensor r. The azimuth angle (horizontal FoV) could be limited, since, according to the employees at HPG, a wide FoV is not crucial in the test track environment since it is not as dynamic as an urban traffic environment and it is desirable to obtain low computational times.

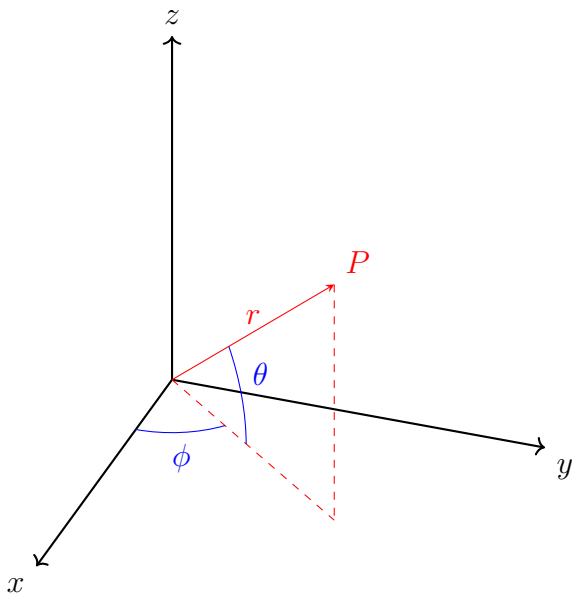


Figure 5.2: Spherical coordinate system visualising how the data point P, can be described with the azimuth angle ϕ, the elevation angle θ and range r.

In Figure 5.2 the coordinate system is shown with spherical polar coordinates for one point cloud data point. The sensor's original FoV is very generous, which gives approximately 60000 data points when using a LiDAR update frequency of 10Hz and both GPE and clustering algorithms have difficulties handling this big amount of data points in reasonable time. Thus, a limitation of the azimuth angle removes certain data points and forces a more narrow FoV with negligible performance losses. Same idea applies to the maximum elevation constraint. Scanned data points above a certain elevation angle is not for consideration. All solid obstacles at HPG are at ground level, leading to a restriction of elevation will not cause any differences in functionality. There is also a regulation for the LiDAR range. Every data point registered at a distance above 100 meters is removed from the dataset. The limit of 100m comes from the calculations below.

The stopping distance d_s, can be calculated as:

$$d_s = d_r + d_b \tag{5.1}$$

where the reaction distance is,

$$d_r = \frac{s_{max} \cdot r}{3.6} \tag{5.2}$$

and the breaking distance is,

$$d_b = \frac{s_{max}^2}{250 \cdot f} \tag{5.3}$$

The maximum speed s_{max} for durability and comfort tests at HPG is 70km/h, reaction time r is 0.1s to match Volvo's safety requirement of 10Hz for real-time systems and the friction coefficient is set to 0.72 [12].

The total stopping distance with these parameters becomes:

$$d_s = 29.17m \tag{5.4}$$

For safe margin, the distance limit is set to 100 meters in this thesis to include more data from the surroundings.

The last filter handles transmitted light that did not return to the photodetector. These are classified as "unsuccessfully received" and are easily removed. Filters for the dataset reduces the number of elements and therefore lower the complexity for later iteration processes.

5.2.2 Multi regional RANSAC

Performing a singular RANSAC (Algorithm 3) for the whole dataset was not robust enough. The algorithm processing a dataset region that is approximately $2000m^2$ large. Fitting a single plane would not be accurate enough to match the real world and could trigger a chain reaction of other problems, such as incorrect classification of data points. Instead the region is divided into smaller subregions where RANSAC is performed individually. The subregions are defined using uniform intervals for the azimuth angle and a distance limit which increases with increased longitudinal distance from the LiDAR. The increment is visualised in Figure 5.3 and is a general expression for the division of the subregion's distances, where the number of data points should approximately be the same for all regions if they were at ground level. The ground angle α_g is determined by the LiDAR's vertical field of view. n is the number of longitudinal regions excluding the "invisible area" l_g outside the FoV range. The distance h is the height of the LiDAR mounting point. The lengths l_i are defined with formula $l_i = h \cdot tan(\alpha_i)$, where the angle α_i is formulated as: $\alpha_i = \alpha_g + i \cdot \Delta\alpha$. The angle $\Delta\alpha$ is identical for all regions and can be described as follows: $\alpha_{i+1} - \alpha_i = \Delta\alpha = \alpha_{i+2} - \alpha_{i+1}$ for $i = 1 \cdots n - 2$.

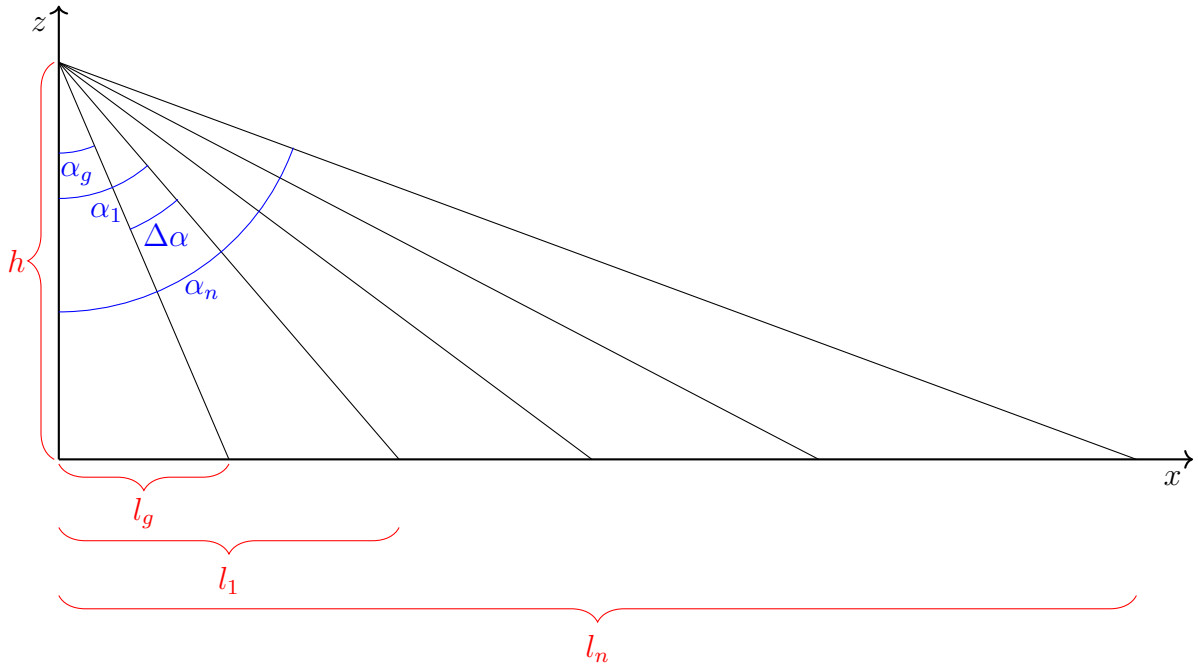


Figure 5.3: Visualisation of how the distance l_i to each subregion is defined from the LiDAR sensor using trigonometry with elevation angles α and the mounting height h

The original RANSAC algorithm is also slightly modified in order to increase perfor-

mance and the independence of the parameters, where one new parameter, plane angle tolerance ϕ_{RANSAC}, is introduced and the ϵ parameter is split into two separate, ϵ_{RANSAC} and ϵ_{final}.
The purpose of ϕ_{RANSAC} is to restrict the plane angle for each iteration, where a plane with an angle above ϕ_{RANSAC} will not be considered.
The reason for splitting ϵ into two, is because there was a need to separate the performance of the identification of planar surfaces and what ultimately should be regarded as ground. When increasing the original ϵ in order to handle situations where the ground was not exactly planar (i.e., very bumpy) the RANSAC algorithm started to prefer objects such as trees or other vegetation rather than planar surfaces. Thus, ϵ was split into ϵ_{RANSAC} and ϵ_{final}, where ϵ_{RANSAC} is only utilised when looking for planar structures in RANSAC, and ϵ_{final} is used to determine what should be regarded as ground, after the plane has been determined.
The final algorithm for the multiregional RANSAC is described in Algorithm 7.

Algorithm 7: Multi regional RANSAC

Data: The point cloud $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N\},\ X \subseteq \mathbb{R}^3$
Result: The least squares planes $\boldsymbol{\pi}^i_{LS}$ for each region, traversable and non-traversable points X_T and X_{NT}.
initialisation;
Partition X into Z number of subsets X^i, one subset for each subregion, according to the gridify algorithm described in REF.
for *i = 1:Z* **do**
  Perform standard RANSAC which yields inliers.
  $X^i_{inliers} := RANSAC(X^i; \epsilon_{RANSAC})$
  Compute the least squares plane on the inliers.
  $\boldsymbol{\pi}^i_{LS} := LS(X^i_{inliers})$
  Classify points in $X^i = \{\boldsymbol{x^i_1}, \boldsymbol{x^i_2}, \cdots, \boldsymbol{x^i_j}, \cdots, \boldsymbol{x^i_{N_i}}\}$ as ground X_T if the distance $d(\boldsymbol{x^i_j}, \boldsymbol{\pi^i_{LS}})$ between the plane and the point is less than ϵ_{final}, and X_{NT} otherwise.
end

5.2.3 Pre-clustering filtering

The computational cost for DBSCAN is proportional to the number of points squared [7]. Thus, it is crucial to reduce the number of points which is fed into DBSCAN in order to reduce the computational time. Apart from the first filter which is applied before the multi regional RANSAC, another one is applied afterwards. The Pre-clustering filter makes use of the information which has been gained from the multi regional RANSAC, which are the estimations of the ground planes.
Since points which is located at a high altitude is not bringing very much value to the cluster analysis, these points are filtered out. This is done by, for every subregion and associated least squares plane $\boldsymbol{\pi}_{LS}$, filter out those points which is located at a distance greater than the parameter ϵ_{pcf}.

5.2.4 Clustering

The clustering is done by DBSCAN which is very much the same as described in Algorithm 2, save from the function which is identifying neighbours; *neighbour identifier*. Previously, this function simply computed which points lied within a radius ϵ from the current point of interest. The problem with this approach was that an object in the point cloud varied in density depending on distance and orientation. Therefore, DBSCAN had difficulties clustering objects which was located further away, and thus, of lower density.

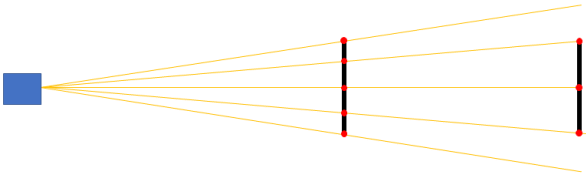


Figure 5.4: A visualisation of how the point density (points in red) will vary with distance when an objects is moved further away. As one can see, the points are much denser on the object to the left than on the right (and fewer as well).

In order to tackle this problem some changes was made to the *neighbour identifier* function. Instead of using a circular neighbourhood to identify neighbours, an ellipse was used, and the geometry of the two points was scaled down to a reference range r_{ref} to yield normalised metrics w_n and Δr_n.

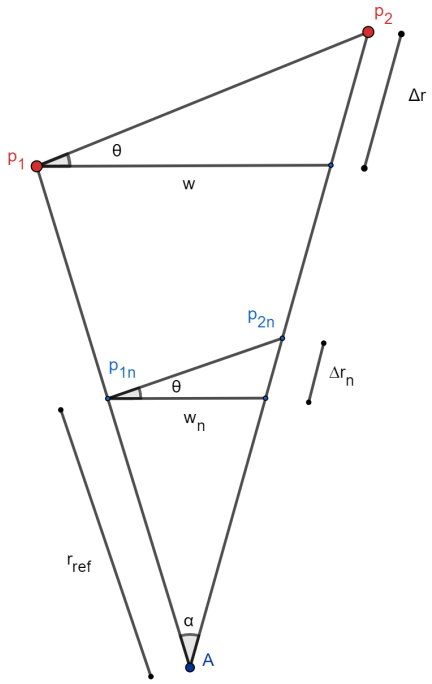


Figure 5.5: A figure visualising how geometry of the points p_1 and p_2 are normalised with a reference range r_{ref}

First, the width w_n is computed by the law of cosine with the parameter r_{ref},

$$w_n = \sqrt{2r_{ref}^2(1 - cos(\alpha))} \tag{5.5}$$

A scaling factor,

$$s = \frac{r_{ref}}{r_1} \tag{5.6}$$

is then computed and is the used to compute,

$$\Delta r_n = s\Delta r \tag{5.7}$$

The two points are then regarded as neighbours if the following condition is fulfilled,

$$\frac{(\Delta r_n)^2}{\epsilon_r^2} + \frac{w_n^2}{\epsilon_w^2} \leq 1 \tag{5.8}$$

where ϵ_r and ϵ_w are radii of the ellipse.
This approach is slightly more computationally heavy than the standard euclidean norm, but it yields much more consistent results.

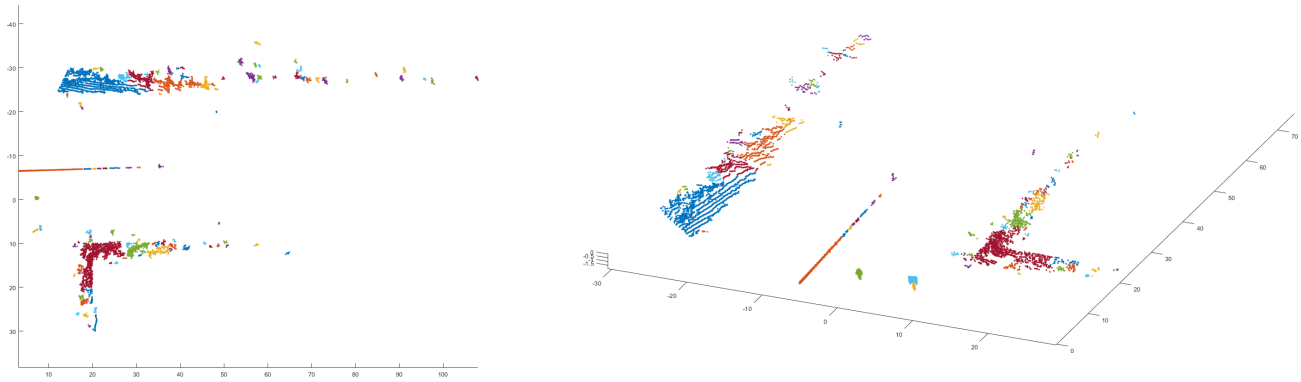


Figure 5.6: An example of the clustering result when using standard euclidean distance. As one can see, there are lots of smaller clusters when the distance from the vehicle increases.

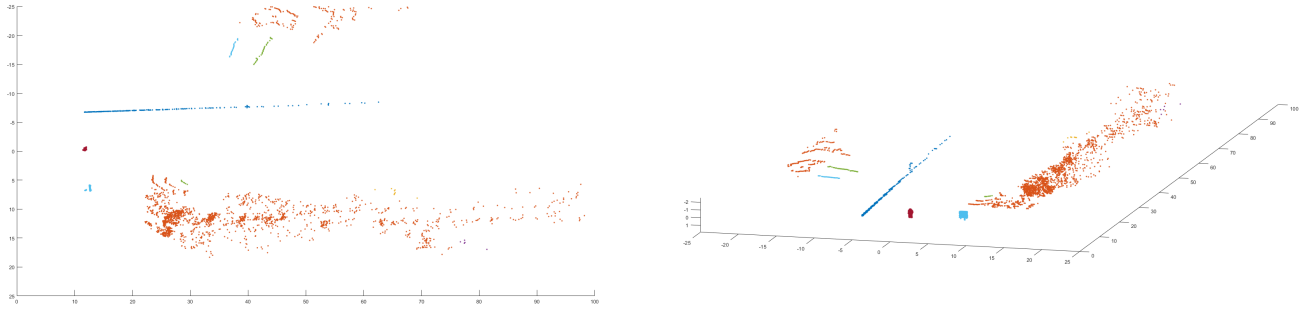


Figure 5.7: An example of the clustering result when utilising the new *neighbour identifier* function. This gives a lot more consistent cluster result.

5.2.5 Bounding geometry fitting

The cluster analysis yields information about which points (that does not belong to the ground) belong to which object. In other words the output from DBSCAN is $C = \{\boldsymbol{c}_1, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_i, \cdots, \boldsymbol{c}_K\}$ where $\boldsymbol{c}_i$ contains all the points that belong to object i. This is not a very simple representation of the object, and would make matters a bit more complex if one were to feed it to a multi target tracker or path planning algorithm.
A common output from neural networks are bounding boxes containing information about position and dimensions, which is a much more compact representation of an object. Because it is a convenient and common format, an algorithm for bounding geometry was created as described in Section 2.5.1.
The box was not always a good representation though. For example, when fitting boxes to big clusters such as those arising from vegetation, it could result in lots of dead space inside the box and the corners of the box could sometimes block the vehicle path, even though it in reality was clear.

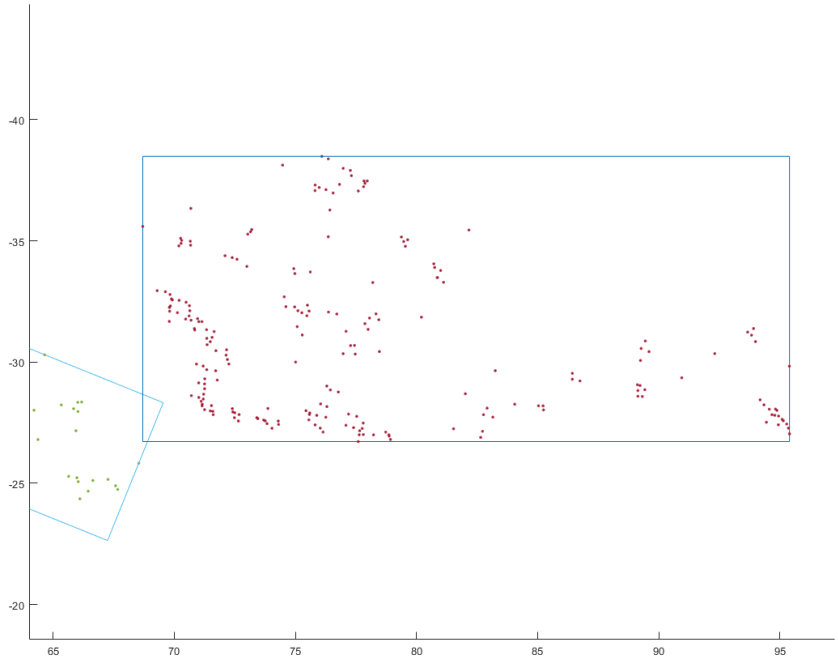


Figure 5.8: An example of a box fitting result to a cluster originating from vegetation. As one can see, there are lots of empty space in the box.

To tackle this problem, the area of the box was investigated. If the area of the box was greater than some tolerance A_{tol}, it computed the convex hull of the cluster as described in Section 2.5.2 and used that to represent the cluster.
One could argue that clusters with big areas originating from objects such as vegetation is rarely of interest anyway, since they very seldom block the intended path of the vehicle. Thus, it is possible to completely ignore clusters which exceed the area tolerance and save some computational power in later stages.

6
Analysis and evaluation

This chapter will present the analysis of the the final algorithm, DBDL, with it characteristics and strengths and weaknesses, as well as the analysis of the LiDAR placement data.

6.1 The final algorithm

The evaluation and analysis has been made with the properties of the commercial CNN-solutions in mind, since these are commonly deployed today, and a decent benchmark to aim for.

6.1.1 Characteristics

It may interpret one obstacle as several clusters: Figure 4.8 shows the issue when the algorithm interprets the longitudinal road rail data points as several clusters (the number of clusters may vary each callback). If one were to implement multi target tracking on the output of the algorithm, it would perform better if the number of clusters an object was partitioned into stayed consistent and did not change too much in between frames. This issue has been somewhat mitigated by the modified DBSCAN function described in Section 5.2.4

It can detect arbitrary objects on the road of sufficient size: One of the algorithm's greatest strengths is that it is able to detect almost any object of sufficient size. The set of objects a machine learning solution would be able to detect would be limited to the training data and the chosen output labels.

The size of the smallest object the algorithm can register depends on several parameters. First and foremost, the object needs to be of sufficient density and the number of points which the object consists of needs to be sufficiently high. These two factors depends mainly on the distance to the object, and the LiDAR update rate (which governs the number of points in a scan). Then, the parameters of the algorithm needs to be tuned right. The parameters which have the biggest influence on the smallest possible object one can detect are ϵ_{final} and p_{min}. As described in Multiregional RANSAC, see Section 5.2.2, ϵ_{final} governs the distance a point may lie from the plane (found by RANSAC) before its regarded as non-traversable. For example, if this parameters is set to 0.3 meters, the algorithm will have a hard time detecting objects with a height above the ground smaller than 0.3 meters. If it is of sufficient size height wise, the number of points which the

object consists of needs to be greater than or equal to p_{min}.

When the parameters was tuned appropriately, a traffic cone of approximately 30cm could be detected at a distance of 5 meters (showed in Figure 4.7) when using an update frequency of 10Hz.

The drawback of having a low value for ϵ_{final} though is that noise originating from uneven ground is more easily picked up. If one would like to use the output from DBDL as input for a path planing algorithm false alarms, such as the noise previously mentioned, can be devastating since there is no filter in between which removes noisy detections.

It is not able to classify objects: This is one of the greatest strengths of a neural network. By being able to classify an object, one would gain access to the inherent properties of an object, such as maximum velocities, general behaviour and such. It would for example be possible to relieve some of the computational load for an MTT if the objects sent into the MTT only consisted of non-static objects.

6.1.2 Remaining issues

False detections may arise from bumpy terrain: In the durability track scenario, which contains very bumpy road, there was a risk for false detections in the area where the vehicle is supposed to drive. It can be mitigated by changing the parameters for the RANSAC tolerances, but it will still be present. The upside though is that these false detections often occur at a sufficiently long distance from the vehicle (approximately 20-30 m), and since the vehicle's speed is very low anyways because of the terrain (5-15 km/h), it will probably not pose a problem if instructed to approach the object slowly (since the detection probably will disappear as the distance to it reduces).
A similar problem occurred in the scenario with the big road hump, where it would recognise the hump as an object at a distance (since the RANSAC zone at that distance was too big to handle the complex geometry of the hump), but would recognise its traversability a lot better as it got closer and the RANSAC zones got finer.

Noisy LiDAR data caused by highly reflective objects: A LiDAR sensor is very sensitive for highly reflective objects, which causes noisy measurements. Figure 4.6 displays how a road sign influences the LiDAR measurements. Generally the DBSCAN have the ability to discard noise, but highly reflective surfaces seems to cause misreadings to enough amount of data points for the DBSCAN to interpret those noisy data points as obstacles. Heavy rain, fog and snowy weather conditions are previously known for also affecting LiDAR readings in similar manner. Thus, they were excluded from this thesis, but is still a large issue to be solved before HPG can be fully autonomous.

It may cluster two or more objects together or partition one object into several: This is one of the hardest problems to solve. It has been mitigated somewhat by introducing the novel distance function which is more suited for the distribution of the points from a LiDAR.

6.2 LiDAR placement and suitability

In the text below, the LiDAR's disturbance results presented in Section 4.2.1 are analysed and some practical aspects of the placements are presented as well.

6.2.1 Analysis of the disturbance influence

When inspecting Figure 4.1 (distribution of the speed of the LiDAR at different positions) and Figure 4.2 (max displacement of the LiDAR for the different position) one can see that the position influenced the least by the road disturbances depends on the track trail (lower values are better). Track trails which involves a lot of roll movement of the vehicle (Mixed Disturbances, Rolling humps in Figure 4.1 and Figure 4.2) mostly influence the roof placement. The other track trails, which mostly trigger a pitch movement of the vehicle, mostly influence the bumper instead. The placement which have the most consistent influence in the measured scenarios is the hood. In every single graph presented in Section 4.2.1, the hood placement falls in between the roof and the bumper and is thus never the best placement in a scenario, but not the worst either.
Thus, assuming that the vehicle will be subjected to a fairly balanced amount of pitch and roll movement, the hood may have a slight advantage if one desires a more consistent disturbance influence.

The above study of the LiDAR's velocity and displacement will unfortunately not convey how much the scans actually will be distorted, only how the three positions compare relative to each other. Thus, a subjective analysis of the distortion from the frames of one of the worst trails has been done, Mixed Disturbances, where Figure 4.3 and Figure 4.4 shows one of the frames.
When inspecting the full LiDAR recording from Mixed Disturbances, it was found that none of the frames looked particularly distorted, neither did the algorithm have any problems to process the frames. This is of course good news, but this subjective analysis can of course be nothing more than a course guidance.

6.2.2 Placement properties analysis

In addition to their different disturbance influence properties, the three mounting positions have other advantages and drawbacks.

- **Better overview on the roof:** The roof position, for example, have in general a better view of the environment (i.e it will have better perception over hill crests) and there is a lower risk that the view of the LiDAR is blocked by objects in front of the vehicle. In a scenario where there is for example another vehicle in front of a LiDAR mounted at the bumper, the vehicle could completely block the view. This would not be the case if it was mounted on the roof.
- **The roof is the most rigid:** The roof is one of the most rigid parts of the car body and is exposed to smaller amounts of vibrations such as those originating from the engine. The hood for example, is simply mounted with hinges on the chassi, and

is much prone to vibrations from the engine, which could influence the scans (not examined in this thesis) or in worst case damage the hardware (not investigated in this thesis either).

- **The roof is a more convenient place to mount the LiDAR:** When speaking to the team at VCC responsible for the LiDAR, they thought that the roof was a more convenient place to mount the LiDAR for two main reasons. Firstly, the roof of the car is already equipped with mounting brackets, which makes the mounting significantly simpler compared to the other positions.
  The other is that the cable management would be a lot more convenient as well since equipment such as the power supply and the computer reading the data is currently situated in the backseat or in the trunk.

7
Conclusion

In the sections below are the recommendation for the LiDAR mounting placement and a conclusion about DBDL's suitability as an object detection algorithm in the future autonomous test vehicles at HPG.

7.1 LiDAR placement recommendation

Taking all the material brought up in analysis into account, the placement recommended in the end, is the roof.
The motivation behind the choice is that it has got several practical advantages that the other placements do not; rigidness, convenient mounting, easier cable management, etc. Disturbance wise, it may be both the worst and the best placement depending on the trail. When inspecting the scans though, the LiDAR did not seem to be very affected by the heavy movement, and thus, the assessment is that the practical advantages outweigh the disturbance factor.

7.2 DBDL

The final algorithm has got some interesting characteristics which is very different from the commercial object detection algorithms in use today. It is not necessarily better or worse, but rather complement each other with their advantages and drawbacks. It would be interesting to implement a solution which utilises both technologies.

The final question which remains is, if the algorithm would be suitable to use at the proving ground for driverless cars.
As of now, there does not exist any MTT and was thus disregarded as an alternative, which leaves the option of using the output from DBDL as input for a decision making algorithm (thus, treating every object as static and with no filter for detections). This demands that the presence of false detections/noise in the direct path of the vehicle and missed detections are close to non-existent, since one false detection in front of the car may trigger an emergency brake and a missed detection could lead to a collision. It is theoretically possible to eliminate noise on the road by setting the parameter ϵ_{final} at a high value, with the penalty of increasing the size of the smallest possible object the algorithm can detect. Thus, the conclusion is that it is possible, but not optimal (depending on the size requirement of the smallest possible object the algorithm can detect), to feed the output of DBDL to a decision making algorithm.

8

Future Recommendations and Ideas

This thesis has presented an alternative approach to object detection, which hopefully can be used as a foundation for driverless applications within car testing, though, there is still some work to do before this can be realised. In the text below, suggestions for additional functionality and solutions to some of the problems encountered are listed.

Optimisation for real-time application: The ultimate goal is to have the object detection running in real time. The whole process, including decision making and the driving robot manoeuvre, have to be performed in a fraction of a second to fulfil safety requirements. Unfortunately, the equipment and software used in this thesis are not fast enough to satisfy these constraints. A part of a solution for this optimisation problem, would be to use a computer with a high performance graphics processing unit (GPU). As of now, code is implemented using Python, which is not sufficiently fast for complex real-time application. Translating the code to a more low level programming language, such as C++, would also reduce computation time. C++ is a dominating language for embedded systems and are a preferred language for many software engineers. A combination of a robust GPU and C++ code would hopefully improve the process speed considerably.

GNSS/INS compensation: A LiDAR exposed to shifts in rotation and translation during a scan, due to the irregular test track road, may produce distorted frames. If one have access to data about these rotations and translations though, they could be compensated while scanning, and the distortion could in theory be mitigated. Thus, it is recommended that the computer that is connected to the LiDAR have access to a GNSS/INS unit to perform this compensation.

Classification of objects after clustering: Classification of an obstacle is very useful for autonomous drive applications and is an evident functionality for most OD-systems. The value it could provide, would be knowledge of the inherent properties of the detected object. For example, objects such as cars, humans and trees all have very different properties. If the algorithm classified an object as a tree, it would automatically know that it is a stationary object, and thus, does not need to be fed to a multi target tracking algorithm. Or if it recognised a human, it would know that the top speed is significantly lower than a car, and maybe have some other behavioural information.
Since the detected objects in the current algorithm store all the points which belong to the object, these points could theoretically be fed to a neural network for classification. The drawback though would be increased computational times, but it is still an interesting idea to evaluate.

Multi Target Tracking: MTT is a natural next step in the autonomous drive chain and is considered an absolute necessity when working in highly dynamic environments, for example to predict the intersecting area with crossing objects [36]. It would not only contribute with an estimation of the trajectories of the objects, but also act as a filter on the detections, which is highly desirable considering the occurrence of detections from noise. Thus, it is highly recommended to implement MTT on the results from DBDL.

Fusion of sensor data: The subject sensor fusion in general is contemporary for many researchers and is already widely used within active safety [29]. Fusing data from several separate sensors will reduce uncertainties in the information that each sensor provides when used independently. Many automotive companies have seen the benefits when utilising sensor fusion for active safety and could be an improvement for locating and tracking moving objects, as well as giving more robust and a higher performing solutions. For instance, sensor fusion between a LiDAR and a radar could be beneficial. The LiDAR provides information about location, whilst the radar can support with information about speed of an object in longitudinal direction [24]. Every type of sensor has its own functionality advantages/disadvantages and fusion with the used LiDAR could improve the deployed algorithm.

Integration: A communication protocol between the detection algorithm and the decision making algorithm have to be deployed for a fully working system, because the computations do not occur on the same computer units currently. Transmission control protocol (TCP) establishes connection between two different units over Ethernet and provides 2-way communications. Compatibility can cause issues for such communications and it is important to assure the information format is supported by both sides. With solved integration, the whole process chain can be deployed and tested in reality. From fetching information using LiDAR sensor to give driving commands based on the input data.

Bibliography

[1] 46 corporations working on autonomous vehicles. https://www.cbinsights.com/research/autonomous-driverless-vehicles-corporations-list.

[2] The 5 clustering algorithms data scientists need to know. https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68.

[3] At volvo cars' test track hällered the car is pushed to its limit. https://www.media.volvocars.com/global/en-gb/media/pressreleases/35453.

[4] Autonomous drive technology – long-range radar. https://www.media.volvocars.com/global/en-gb/media/photos/158307/autonomous-drive-technology-long-range-radar.

[5] Autonomous drive technology – ultrasonic sensor. https://www.media.volvocars.com/se/sv-se/media/photos/158310/autonomous-drive-technology-ultrasonic-sensor.

[6] Breaking down mean average precision (map). https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52.

[7] Computational complexity and performance of various clustering algorithms. https://hdbscan.readthedocs.io/en/latest/performance_and_scalability.html.

[8] Confusion matrix in machine learning. https://www.geeksforgeeks.org/confusion-matrix-machine-learning/.

[9] Description of ros. https://www.ros.org/about-ros/.

[10] Engineer explains: Lidar. https://blog.cometlabs.io/engineer-explains-lidar-748f9ba0c404.

[11] Sensors in the volvo v60 now detect cars coming right at you. https://www.techradar.com/news/sensors-in-the-volvo-v60-now-detect-cars-coming-right-at-you.

[12] Table with common coefficients for friction. https://www.engineeringtoolbox.com/friction-coefficients-d_778.html.

[13] Volvo cars and luminar show groundbreaking autonomous technology development at automobility la 2018. https://www.media.volvocars.com/global/en-gb/media/pressreleases/246168/volvo-cars-and-luminar-show-groundbreaking-autonomous-technology-development-at-automobility-la-2018.

[14] Volvo cars conducts research into driver sensors in order to create cars that get to know their drivers. https://www.media.volvocars.com/global/en-gb/media/pressreleases/140898/volvo-cars-conducts-research-into-driver-sensors-in-order-to-create-cars-that-get-to-know-their-driv.

[15] *National Highway Traffic Safety Administration Preliminary Statement of Policy Concerning Automated Vehicles*, 2013.

[16] Alireza Asvadi, Cristiano Premebida, Paulo Peixoto, and Urbano Nunes. 3d lidar-based static and moving obstacle detection in driving environments: An approach based on voxels and multi-region ground planes. *Robotics and Autonomous Systems*, 83:299 – 311, 2016.

[17] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[18] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[19] J. Cashbaugh and C. Kitts. Automatic calculation of a transformation matrix between two frames. *IEEE Access*, 6:9614–9622, 2018.

[20] S. Choi, J. Park, J. Byun, and W. Yu. Robust ground plane detection from 3d point clouds. In *2014 14th International Conference on Control, Automation and Systems (ICCAS 2014)*, pages 1076–1081, Oct 2014.

[21] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[22] Vincent Fung. An overview of resnet and its variants. https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035, July 2017.

[23] J. Giacalone, L. Bourgeois, and A. Ancora. Challenges in aggregation of heterogeneous sensors for autonomous driving systems. In *2019 IEEE Sensors Applications Symposium (SAS)*, pages 1–5, March 2019.

[24] Daniel Goehring, Miao Wang, Michael Schnurmacher, and Tinosch Ganjineh. Radar/lidar sensor fusion for car-following on highways. pages 407–412, 12 2011.

[25] Kanwaldeep Kaur and Giselle Rampersad. Trust in driverless cars: Investigating key factors influencing the adoption of driverless cars. *Journal of Engineering and Technology Management*, 48:87 – 96, 2018.

[26] Steven LaValle. The homogeneous transformation matrix. http://planning.cs.uiuc.edu/node111.html, 2006.

[27] Linna Huang and Guangzhong Liu. Proved quick convex hull algorithm for scattered points. In *2012 International Conference on Computer Science and Information Processing (CSIP)*, pages 1365–1368, Aug 2012.

[28] Wenjie Luo, Bin Yang, and Raquel Urtasun. Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[29] Panagiotis Lytrivis, George Thomaidis, and Angelos Amditis. *Sensor Data Fusion in Automotive Applications*. 02 2009.

[30] D. Matti, H. K. Ekenel, and J. Thiran. Combining lidar space clustering and convolutional neural networks for pedestrian detection. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6, Aug 2017.

[31] E. Mucke. Computing prescriptions: Quickhull: Computing convex hulls quickly. *Computing in Science Engineering*, 11(5):54–57, Sep. 2009.

[32] Faisal Mufti, Robert Mahony, and Jochen Heinzmann. Robust estimation of planar surfaces using spatio-temporal ransac for applications in autonomous vehicle navigation. *Robotics and Autonomous Systems*, 60(1):16 – 28, 2012.

[33] T. Okuyama, T. Gonsalves, and J. Upadhay. Autonomous driving system based on deep q learnig. In *2018 International Conference on Intelligent Autonomous Systems (ICoIAS)*, pages 201–205, March 2018.

[34] M. Pavelka and V. Jirovský. Lidar based object detection near vehicle. In *2017 Smart City Symposium Prague (SCSP)*, pages 1–6, May 2017.

[35] G. Prabhakar, B. Kailath, S. Natarajan, and R. Kumar. Obstacle detection and classification using deep learning for tracking in high-speed autonomous driving. In *2017 IEEE Region 10 Symposium (TENSYMP)*, pages 1–6, July 2017.

[36] Ba-ngu Vo, Mahendra Mallick, Yaakov Bar-shalom, Stefano Coraluppi, Richard Osborne III, Ronald Mahler, and Ba-tuong Vo. *Multitarget Tracking*, pages 1–15. American Cancer Society, 2015.

[37] Gustav von Zitzewitz. Survey of neural networks in autonomous driving. 07 2017.

[38] Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[39] Erin Welling. Convolutional neural networks in autonomous vehicle control systems. 2017.

[40] Slawomir Wierzchon and Mieczyslaw Kłopotek. 01 2018.

[41] Z. Y. Zhang, J. Zheng, X. Wang, and X. Fan. Background filtering and vehicle detection with roadside lidar based on point association. In *2018 37th Chinese Control Conference (CCC)*, pages 7938–7943, July 2018.

A
Appendix 1

Cobble			Frequency Test		
	Mean	Maximum		Mean	Maximum
Roof	0.2145	0.7337	Roof	0.0779	0.3527
Hood	0.2746	0.8231	Hood	0.0839	0.4941
Bumper	0.3743	1.0448	Bumper	0.1032	0.6682
Big Road Hump 20 kmh			Big Road Hump 40 kmh		
	Mean	Maximum		Mean	Maximum
Roof	0.1195	0.6633	Roof	0.2029	1.3141
Hood	0.1252	0.7399	Hood	0.2697	1.7773
Bumper	0.1491	0.9233	Bumper	0.3477	2.3691
Submersion 30 kmh			Submersion 50 kmh		
	Mean	Maximum		Mean	Maximum
Roof	0.1403	0.567	Roof	0.1175	1.1273
Hood	0.1422	0.7006	Hood	0.118	1.4114
Bumper	0.1726	0.8847	Bumper	0.1379	2.0316
Mixed Disturbances 10 kmh			Mixed Disturbances 20 kmh		
	Mean	Maximum		Mean	Maximum
Roof	0.3458	1.5645	Roof	0.3649	1.5675
Hood	0.253	1.0405	Hood	0.2688	0.9002
Bumper	0.2441	0.8444	Bumper	0.2599	0.8527
Smooth Humps 50 kmh			Smooth Humps 70 kmh		
	Mean	Maximum		Mean	Maximum
Roof	0.1423	0.68	Roof	0.2589	0.9803
Hood	0.1398	0.7667	Hood	0.2448	0.9566
Bumper	0.159	0.9276	Bumper	0.2693	1.2355
Rolling Humps			Washboard 30 kmh		
	Mean	Maximum		Mean	Maximum
Roof	0.3029	1.2695	Roof	0.1109	0.2285
Hood	0.1984	0.7993	Hood	0.1841	0.2417
Bumper	0.153	0.582	Bumper	0.2372	0.3025
Washboard 50 kmh			Washboard 70 kmh		
	Mean	Maximum		Mean	Maximum
Roof	0.1674	0.3108	Roof	0.1837	0.433
Hood	0.2919	0.3906	Hood	0.3168	0.5435
Bumper	0.3797	0.4977	Bumper	0.4317	0.6227

Figure A.1: The distribution of the LiDAR velocities in terms of mean and maximum for all 14 recordings

B

Appendix 2

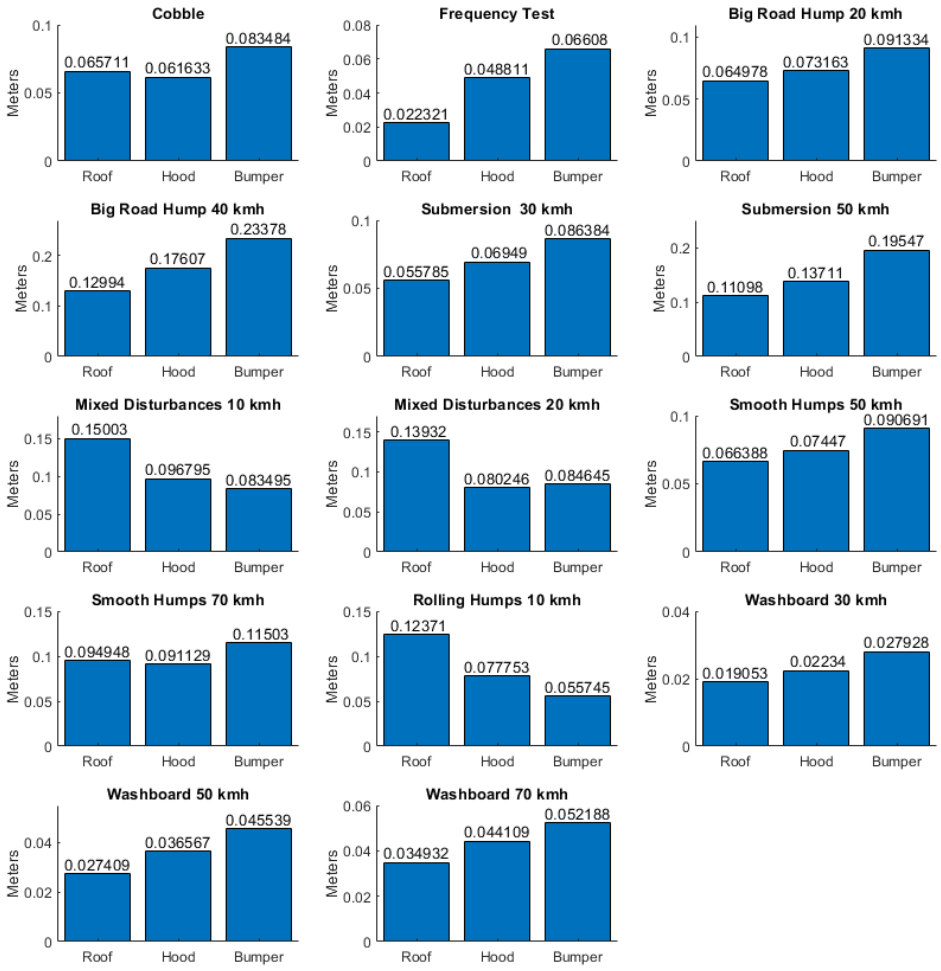


Figure B.1: The computed maximum displacement for all 14 recordings