Improving a Background model for tracking and classification of objects in LiDAR 3D point clouds

Seamus Doyle, Gustav Nilsson

Master's thesis
2022:E23

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Abstract

This thesis studied methods of improving a *background model* for a data processing pipeline of LiDAR point clouds. For this, two main approaches were evaluated. The first was to implement and compare three different models for detecting ground in a point cloud. These were based on more classical modeling approaches. The second was to use Deep Learning for semantic segmentation of point clouds and to use this information in a background filtering model with the hope of achieving better filtering of dynamic background. These methods were combined in a pipeline as an example of a possible application. The performance of the ground models was primarily evaluated based on their ability of classifying points as ground or non-ground. However, well performing ground models have further uses. Of the three models studied, the *Hybrid model* achieved most promising results. For semantic segmentation, RandLA-NET was used for its ability to process large scale point clouds at high speeds. Variations of the network was trained on simulated data for which all networks achieved similar good performance for classes *ground*, *vegetation* and *other*. When testing domain transfer to point clouds produced by a Real Physical LiDAR, mixed results were achieved with variations on a per-point-cloud basis. On a lot of instances however, very promising results could be seen. A background subtraction model based on a *3D Density Static Filter* was extended to include semantic information from the neural network. For this filter, voxels classified as *vegetation* and their neighbours, heavily filtered out points. This was to avoid issues of false detections caused by wind. The model was tested on parts of two LiDAR recordings and compared to the standard filter. Based on this, the extended model was found to better filter out vegetation in windy conditions.

Acknowledgements

We want to thank our supervisor Anders Heyden for continual support throughout this thesis. We also want to thank Erik for his help with the CARLA simulator and the *Graphic LiDAR plugin*.

List of Acronyms

DBSCAN Density-Based Spatial Clustering of Applications with Noise . 15

GPR Gaussian Process Regression 5

KNN k-nearest neighbours 22

LiDAR Light Detection and Ranging 2

LocSE Local Spatial Encoding 19

RANSAC Random Sample Consensus 10

RLWR Robust Locally Weighted Regression 11

RP-LiDAR Real Physical LiDAR 7

SGD Stochastic Gradient Descent 18

Contents

1 Introduction **2**
- 1.1 Task . . . 3
- 1.2 Limitation . . . 4
- 1.3 Related Work . . . 4
- 1.4 Statement of Contribution . . . 5

2 Background **6**
- 2.1 Background Model . . . 6
- 2.2 Ground Model . . . 6
  - 2.2.1 Semantic Segmentation . . . 6
- 2.3 LiDAR sensor . . . 7
- 2.4 Data . . . 7
  - 2.4.1 RP-LiDAR data . . . 8
  - 2.4.2 CARLA . . . 8
- 2.5 Semantic KITTI . . . 10

3 Theory **10**
- 3.1 Plane Segmentation with RANSAC . . . 10
- 3.2 Regression . . . 10
  - 3.2.1 Robust Locally Weighted Regression . . . 11
  - 3.2.2 Gaussian Process Regression . . . 13
- 3.3 Density-Based Spatial Clustering of Applications with Noise . . . 15
- 3.4 Background Filter . . . 15
- 3.5 Semantic segmentation . . . 16
- 3.6 Neural networks . . . 16
  - 3.6.1 Loss function . . . 18
  - 3.6.2 Optimizer . . . 18
- 3.7 RandLA-Net . . . 19
  - 3.7.1 Local Spatial Encoding . . . 19
  - 3.7.2 Attentive Pooling . . . 19
  - 3.7.3 Dilated Residual Block . . . 20
  - 3.7.4 Inference . . . 20
  - 3.7.5 K-d tree . . . 20
  - 3.7.6 Possibility iteration . . . 21
  - 3.7.7 Voting scheme . . . 21
  - 3.7.8 Grid subsampling . . . 21
  - 3.7.9 Network architecture . . . 22
- 3.8 Evaluation Metrics . . . 22
  - 3.8.1 Confusion Matrix . . . 24

4 Method **25**
4.1 Simulation of data . . . 25
4.1.1 CARLA LiDAR Data . . . 25
4.1.2 Graphic LiDAR Dataset . . . 25
4.1.3 Ground model Dataset . . . 26
4.2 Ground Models . . . 28
4.2.1 Implementation of RLWR-based Model . . . 28
4.2.2 Implementation of Hybrid Regression Model . . . 29
4.2.3 Implementation of Plane Model . . . 31
4.3 Test on Ground models . . . 32
4.4 Evaluation of Ground model on point clouds . . . 33
4.5 RandLa-NET . . . 34
4.5.1 Data preprocessing . . . 34
4.5.2 Training of network . . . 34
4.5.3 Testing of the network . . . 35
4.6 Improvement to the Background Density Filter . . . 35
4.7 Test on Background Subtraction . . . 36

5 Results **38**
5.1 Ground Models . . . 38
5.1.1 Hybrid Model on RP-LiDAR point clouds . . . 44
5.2 RandLa-NET . . . 46
5.2.1 Training . . . 46
5.3 Background Filter . . . 54

6 Discussion **57**
6.1 Ground Models . . . 57
6.1.1 Performance of Plane model . . . 57
6.1.2 Performance of RLWR-based model . . . 57
6.1.3 Performance of Hybrid model . . . 58
6.1.4 Further analysis of Hybrid Model . . . 59
6.1.5 Conclusions . . . 61
6.2 RandLA-NET . . . 61
6.2.1 Conclusion . . . 64
6.3 Background Filtering . . . 64
6.3.1 Conclusion . . . 65

7 Conclusion **65**

8 Future Work **67**

Appendices **71**

9 Appendix **71**
A F_1-scores and Skewed Datasets 71
B Gaussion Process Regression 71
B.1 Including noise 71
C Fitting a Gaussian Process Regression Model 72
C.1 Accounting for noise in Kernel 73
D Further Discussion of Ground models 74
D.1 Impact of specificity 74
D.2 Further Discussion of RLWR-based model's poor performance 74
D.3 How impact of sparse circles might have impacted GPR fit to be more conservative 74

1 Introduction

Light Detection and Ranging (LiDAR) is a sensor that utilises pulses of light to create a three-dimensional representation of its surrounding. This representation is commonly referred to as a point cloud and can be used for object detection and tracking.

A new and promising application is that of detection and tracking of people, animals etc. using a stationary LiDAR. Such applications have existed for some time and have become quite well established for conventional cameras. While these cameras have many advantages, they also have quite a few disadvantages. These cameras require a well lit environment, have no depth perception without adding multiple cameras and can struggle in differentiating where boundaries between objects occur in their images. LiDAR-technology has the possibility of solving all these problems. The sensor produces its own light-source and can therefore be used in complete darkness. Depth perception is inherent to producing point clouds and the LiDAR also saves the intensity in the reflected light beam. All this can be used in order to better distinguish different objects.

For the purpose of point cloud processing in the context of object tracking and detection, there are a few fundamental concepts that are needed for comprehension. Such a process (in this thesis referred to as a pipeline) typically follows a set of four processing steps that are performed for each point cloud frame. These are shown in Figure 1 as a flowchart. First, a raw point cloud is produced by the sensor. The initial processing step is to remove background points, and is thus called **Background Subtraction**. In the following **Clustering** step the remaining points are divided into groups according to their vicinity to each other and density. The purpose is for the points to be grouped into coherent dynamic objects such as pedestrians, cyclists, animals etc. Thereafter **Tracking** of each cluster between frames is done in order to impose continuity. This means that information regarding position over time of a certain dynamic object can be retained. Finally **Classification** is performed on each cluster, meaning that each cluster is passed through an algorithm that produces a guess of what the cluster represents, i.e. a pedestrian, cyclist and so on.

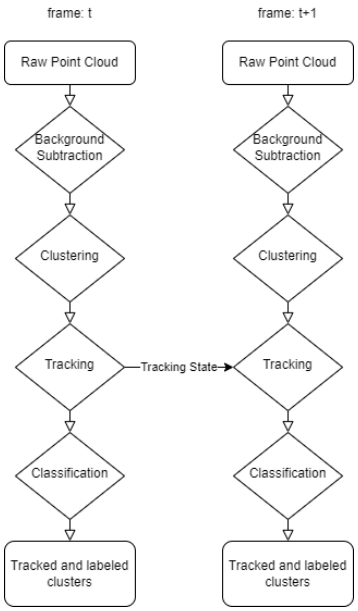


Figure 1: Flowchart of the main pipeline for processing point cloud data.

1.1 Task

The aim of this thesis was to study ways of improving background modeling in the **Background Subtraction** step. The goals were threefold:

- Find a method of capturing information of the contents of the background in a model for background subtraction.
- Find a method for performing semantic segmentation of large scale point clouds.
- Compare different methods for detecting and modeling ground in point clouds.

To complete these goals, the following was done. First, an attempt was made to perform semantic segmentation on point clouds of the background in order to encode information of the background such as ground, vegetation and other objects into a scene. This was done using Deep Learning using neural networks and incorporated into an existing background model. The second was to develop models for finding and estimating the height of the ground mainly using different regression methods. These two models were thereafter fused into a joint background processing step, for which, further logic can be integrated that is more specifically adapted for a given use case.

1.2 Limitation

Due to the wide array of possibilities in improving the background model, this project was limited to studying two main approaches and weaving them together. These approaches were (1) semantic segmentation of point clouds with an expanded background model to utilize the semantic information and (2) constructing ground models. Also, due to the lack of annotated data for real LiDAR sensors, the study was mainly limited to, and driven by, quantifiable results from simulated data. In the case of this thesis, a strong benefit of using simulated data was the simplicity of scaling up the size of training, testing and validation data. Among the disadvantages were the loss of the reflectance dimension and the data being less realistic.

1.3 Related Work

The work in this thesis builds on a preexisting body of work in the form of two previous master thesis projects, the first by Berntsson and Winberg [1] and the second by Bernståhle and Lind [2]. In these projects, the same technology was studied in the same setting, but where different parts of the process were focused on. In Berntsson and Winberg's thesis, a complete pipeline consisting of background filtration, clustering, tracking and classification was established and multiple methods were considered for many of the different processes. For all steps except the classification, classical statistical, methods were considered. In Bernståle and Lind's project however, the main focus consisted of designing and implementing trackers and classifiers based on Deep Learning models. This leaves room for improvement for primarily the background and clustering stages of the pipeline, and it is the background that this thesis aims to focus on improving.

Since the LiDAR considered in this application is stationary, background models assuming stationary backgrounds have been considered. Such methods are analyzed in [3], [4], [5], [6], [7]. In [3], researchers present a 3D density static filtering (3D-DSF) for filtering static points in point clouds by accumulating point densities in voxels and removing points when a threshold has been reached. In [4], this work is further improved by making the threshold variable for different distances and using histograms of point frequencies for different distances. This is to take into account variable point density for different distances caused by a LiDAR's scan pattern. In [5], the authors perform clustering with DBSCAN with different maximum distance between samples (ϵ) at different distances from the sensor and then merges these to account for decreased point densities at further distances. In [6], the authors develop a background filtering technique that they call the *Max-distance filter* where the distance to the closest point registered for each angle is determined as the distance to the background. For every such angle, any new points closer than this with an additional safety threshold are considered foreground and are not removed. In [8], the authors propose a background filtering method where mean background modeling is combined

with a background difference method. In the same article, a Hierarchical Maximum Density Clustering of Applications with Noise (HMDCAN) is proposed to cluster points.

In [9], [10], [11] and [12] different methods of creating ground models are explored. In [9], Robust Locally Weighted Regression (RWLR) is utilized together with down-weighting to estimate the ground level for slices in a point cloud along $x - z$ and $y - z$ profiles. Points classified as ground in both directions are determined as ground points. In [10], the authors combined Robust Locally Weighted Regression (RLWR) with Gaussian Process Regression (GPR) to produce a ground surface filtering method designed to handle more sparse point clouds with ground undulation and more occlusions. Projecting points into a polar grid map is also introduced. The authors of [11] created a neural network called GndNet that takes raw point clouds as input and produces ground level estimation in a grid together with segmentation of all points in ground and non-ground as output. In [12], the authors segment ground points by dynamic section division, height-based conditional filter and multi-lines linear regression.

Semantic segmentation of point clouds is studied in [13], [14] and [15]. Well established work within point cloud semantic segmentation, is the development of pointnet and pointnet++, see [13]. Pointnet consists of a set of Multi-Layer-Perceptrons (MLP:s) that produces a spatial encoding of each point in the point cloud. Pointnet++ uses a hierarchical structure to encode feature-information in local regions to progressively increase the receptive field. This structure consists of several sampling, grouping and Pointnet-layers. The sampling used was Farthest point sampling (FPS) which is useful to sample data with varying point density. Voxnet, see [14], is a 3D convolution network that utilises 3D convolutions over voxels to predict semantic labels. The point clouds are pre-processed to create a voxelisation which the 3D kernels can iterate over. KpConv, see [15], is a kernel point convolution network which uses convolutions over points instead of voxels. The point convolution kernel has its area of influence determined by a correlation function. The network adjusts its shape depending on the sparsity of a local points making it robust to varying point densities.

1.4 Statement of Contribution

Throughout the project, Seamus Doyle and Gustav Nilsson have been working very closely together. The work was however divided in such a way that Gustav focused on developing and evaluating methods for *Semantic Segmentation* using *Deep Learning* and *Visualisation*, whereas Seamus focused on developing and testing methods for a *Ground Model* and a framework for incorporating the semantic information in a *Bakground Model*.

2 Background

2.1 Background Model

In the model from previous master thesis projects, there is no information of what the different parts of the background are. Knowledge of this can greatly help in different ways. This will be explored in this section.

2.2 Ground Model

In [1], a simple ground model was used, where a plane was fit to the point cloud. This was done through a *plane segmentation filter* based on the RANSAC algorithm, see [16]. Whereas this works in many cases, ground is not always flat and so, more advanced models were constructed. The advantages of using such ground models are several. The first is that a layer of logic that uses the estimated ground for a given set of x and y coordinates can be applied. This would be specific for every application. An example of this would be to remove clusters with centroids far above the ground due to the only clusters of interest being objects on the ground such as pedestrians, cyclists, cars and more. Another example would be to remove clusters with centroids that are very close to the ground with the motivation that these are small objects which might not be of interest. These examples have different requirements on performance of such a ground model. The second advantage is that height information can be used as a feature in tracking and classification of clusters. The centroid of a person or a car is likely higher than that of an animal. Other advantages is that detection and possible removal of ground points could enhance other processes such as semantic segmentation. Processing large point clouds is costly and decreasing the sizes of these can lead to increased performance.

2.2.1 Semantic Segmentation

Using semantic segmentation for encoding information of the environment was mainly driven by the fact that background models in the previous pipeline fail to subtract points belonging to vegetation whenever there was wind. This is due to the relative sparsity of the point cloud in combination with these objects having dynamic parts (swaying branches etc.). A way to combat this, is to detect where in a scene this vegetation is located, so that nearby points are subtracted to a greater extent. Knowing whether points belong to other categories such as ground or other is further useful. Being able to segment ground is useful for the synergistic effects with the ground model. Further exploration of different categories can provide added benefits for visualisation and logic using information of the scene. One example of this logic is removing the effects of LiDAR-beams reflecting off reflective surfaces. Another example is using knowledge of a how an object in a class usually looks to complete the backside of the object in the point cloud.

2.3 LiDAR sensor

In [17], LiDAR is described as an active sensor that has its own light source. Compare this to passive sensors such as traditional cameras and human eyes. This has advantages since the data produced is less dependent on external variables such as light conditions and certain types of weather. There are two common wavelengths for LiDAR sensors, $905nm$ and $1550nm$, the latter of which is less limited to how much light it can emit since the wavelength is further from human visibility. The Real Physical LiDAR, which is used in this thesis uses $1550nm$ lasers. One way of aiming the laser is to use low-mass mirrors that move in order to angle the outgoing light in different directions.

The recorded data, for which this thesis aims to improve the pipeline for, was recorded with a LiDAR sensor mainly built for mounting onto cars. In this case it was used attached to poles with a vantage point. Thus, they were used in a stationary setting within the context of this thesis. The sensor contained two sensor modules, each covering a little more than half of a total 120° field of view and used scanning mirrors to angle the light. The input of each module is stitched together with some overlap. Further specifications are detailed in Table 2.3.

Maximum Range	500m
Horizontal FoV	120°
Vertical FoV	30°
Frames Per Second	1-30 fps
Minimum Horizontal Resolution	0.07°
Minimum Vertical Resolution	0.03°
Range Precision	0.01 m

It was possible for the LiDAR to use different vertical scan patters, meaning that the angle between scan lines can vary. Whereas a constant scan-line-density is common, most of the recordings utilized in this thesis used a trapezoidal scanning pattern. This has the effect of objects nearby the sensor having roughly the same point density as that of objects that are further away (often to about $100m$ away). This is due to the fact the trapezoidal maximum is aimed at the horizon where the distant objects often appear. This LiDAR will be termed the Real Physical LiDAR (RP-LiDAR) throughout this report to distinguish it from the simulated LiDARs also used in this thesis.

2.4 Data

In this thesis project, data came from three sources: unlabeled data from the RP-LiDAR, labeled data from a simulator called CARLA [18] and finally a labeled Benchmark dataset called Semantic-KITTI [19] from a spinning LiDAR of a different type (Velodyne) on top of a car.

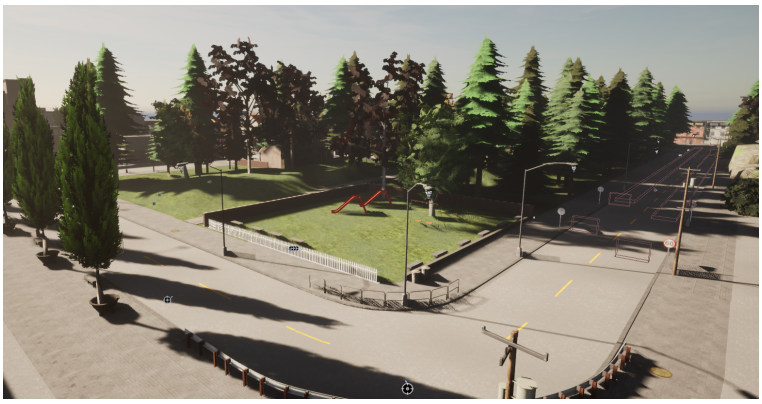
The data for the stationary LiDARs generally have the same structure. From the LiDAR's point of view, the z-axis is always the upwards direction, the x-axis is always the forwards direction and thus, the y-axis always points in the side-wise direction. Depending on the orientation of the LiDAR, the produced point cloud might have to be rotated for the z-axis to point upwards. This is usually the case as the LiDAR is generally mounted as to look down onto a scene from a vantage point.

2.4.1 RP-LiDAR data

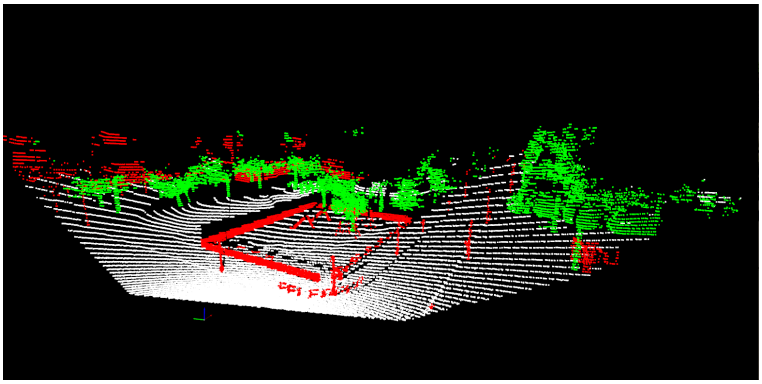
The RP-LiDAR data consisted of various recordings in the Skåne region where different settings were experimented with. Thus, the set of data contained different frame counts, frame rates, resolutions and scan patterns. This data was used to test and validate models trained on the other labeled datasets and the different settings were therefore useful in both deducing factors affecting domain transfer and exploring different ways of including the background model. The scenes mostly used from this dataset consisted of roughly $1000 - 3000$ frames, used a **trapezoidal scanning pattern** and had a frame rate of 10 frames per second. It will be mentioned whenever other settings were used.

2.4.2 CARLA

CARLA is a simulation platform based on Unreal Engine 4 and is according to their website, [20], developed to "support development, training, and validation of autonomous driving systems". It is open source and contains many digital assets such as urban environments, vehicles and buildings and support for autonomous agents with Reinforcement Learning. It also contains methods of simulating sensors such as LiDAR in its environment and this could be leveraged through a Python API, see [18].



(a) CARLA view



(b) Graphic lidar view

Figure 2: Rendered view in CARLA vs the output point cloud from the Graphic LiDAR plugin.

On top of providing its own LiDAR, CARLA as an Unreal Engine 4 based platform also provides support for third party plugins, and thus a LiDAR called "Graphics based UE4 LiDAR", or "Graphic LiDAR" for short, was also used with the CARLA assets. The main difference between the simulated LiDARs was that, whereas CARLAs LiDAR simulates beams that reflect off meshes, the Graphic LiDAR takes see-through textures into account and thus more closely resembles an actual LiDAR.

The LiDAR plugin was based on the rendering pipeline provided by Unreal Engine 4. Here, a scene is rendered from the viewpoint of the LiDAR and a post process shader is applied to encode depth, reflectance and object tag into a 16 bit float RGBA texture. 3D-coordinates of points is thereafter computed from data read from the texture. The CARLA lidar uses **uniform scanning pattern**.

For the purpose of this thesis, CARLA's environment assets together with the LiDAR and the Graphic LiDAR plugin was used to simulate labeled data.

2.5 Semantic KITTI

Semantic Kitti is a dataset released by scientists from the University of Bonn. It contains over 20 000 point cloud LiDAR scans from a velodyne LiDAR attached to a car driving in a city/suburb environment, see [19]. The data is recorded continuously at 10 Hz in a 360 degree horizontal field of view. The data has been annotated with 19 different classes and is often used as a benchmark to test different networks for semantic segmentation of point clouds.

3 Theory

3.1 Plane Segmentation with RANSAC

Random Sample Consensus (RANSAC) is a method of modeling data that contains a large amount of errors or outliers and finding the best possible fit. Thus, it can be used to find certain features in data such as ground planes in point clouds. In essence, the method works by fitting a model given a randomly selected small subset n of points in the point cloud. From this, all data within an error tolerance δ of the model are considered part of an inlier/consensus set, see [16]. Depending on the implementation, this is either repeated a fixed number of iterations t and the best fit is selected, as in [21]. Otherwise it is repeated until the inlier set is greater than some specified threshold, as in [16]. For the sake of plane segmentation, this corresponds to fitting a plane given n points, considering all points within distance δ of the plane as inliers and selecting the best such plane. *Open3D*'s implementation of RANSAC for plane segmentation uses the iteration procedure of a fixed number of t iterations and selects the best fit, see [21].

Such a plane is specified in Equation 1.

$$ax + bx + cz + d = 0 \tag{1}$$

The scalars (a, b, c) form together the normal vector of the plane and (x, y, z) is any point in the plane that satisfy the equation.

3.2 Regression

Regressions are supervised learning methods where continuous variables (dependent variables) are predicted from other quantities (independent variables). In this thesis, regressions were used for modeling ground in two of the ground models. The RLWR-based model used solely Robust Locally Weighted Regression, whereas the Hybrid model used both.

3.2.1 Robust Locally Weighted Regression

Robust Locally Weighted Regression, or RLWR for short, introduced by Cleveland in [22] is a method that locally smooths a scatterplot $(x_i, y_i), i = 1, ..., n$ in such a way that values are polynomial fits to data using weighted least squares. The weights are chosen with respect to the point distances. RLWR uses a robust fitting procedure that prevents outliers from having too great an impact on the fit. The procedure consists of computing a locally weighted scatterplot smoothing (LWR or sometimes lowess) and then following this by iteratively performing a step that makes the method more robust. Together, these two procedures form the robust locally weighted regression (Robust Locally Weighted Regression (RLWR)).

The method is designed to fit data in the form of Equation 2.

$$y_i = g(x_i) + \epsilon_i \tag{2}$$

Here, y_i are dependent variables, x_i are independent variables, g is a smooth function and ϵ_i are zero-mean random variables with constant variance. Assuming smoothness means that points in the neighborhood of (x_i, y_i) can be used to estimate $\hat{y_i}$ which is a an estimate of $g(x_i)$, see [22]. According to [9], it can also be assumed that $g(x)$ can be estimated well by a family of simpler parametric functions and any differentiable function can be locally approximated by a straight line according to Taylor's theorem. The scatterplot is assumed to consist of n points, i.e. $i = 1, ..., n$.

The LWR step is performed by first defining a fraction of points $0 < f \leq 1$ to which every fitted point (x_i, y_i) is to be compared to and r is the number of neighbours that it translates to $(r = round(fn))$. Thus, a large fraction (f) corresponds to a smoother fit. For every point, the local neighborhood $(N(x_i))$ is defined as the r closest points in x-space. The weight function used in the LWR-regression is the tri-cube weight function with $j = 1, ..., n$ is seen in Equation 3.

$$\omega_i(x_j) = \begin{cases} (1 - (\frac{|x_i - x_j|}{\max\limits_{k \in N(x_i)} |x_i - x_k|})^3)^3 & j \in N(x_i) \\ 0 & j \notin N(x_i) \end{cases} \tag{3}$$

In the equation $\max\limits_{j \in N(x_i)} |x_i - x_k|$ is the distance to r:th nearest neighbor. Thus all points further than this from x_i will not affect the local fit. The tri-cube weight function is chosen since it "enhances a chi-squared distributional approximation of an estimate of the error variance" (Belton et al., 2016, p. 2184) and also, it provides smooth results in most situations, see [9].

For every (x_i, y_i) $g(x)$, is fit $(\hat{g}_i)$ by weighted least squares with weight $\omega_i(x)$ by estimated fitted values $\hat{g}_i(x_k)$ that minimizes Equation 4.

$$\sum_{k=1}^{n} \omega_i(x_k)(y_k - \hat{g}_i(x_k))^2 \tag{4}$$

Since $\hat{g}_i(x_k)$ are approximated as d-order polynomials, this equates to, for each i, computing the weights $\hat{\beta}_l(x_i), l = 0, ...d$ of possible β_l:s that minimize Equation 5.

$$\sum_{k=1}^{n} \omega_i(x_k)(y_k - \beta_0 - \beta_1 x_k - ... - \beta_d x_k^d)^2 \tag{5}$$

Finding the minimizing set of $\hat{\beta}_l, l = 0, ...d$ for every i means having estimated every $\hat{y} = \hat{g}(x_i)$ and thus having smoothed the scatterplot with LWS.

Thereafter follows the robustification step. B is the bisquare weight function, which is defined in Equation 6 where z is an arbitrary scalar.

$$B(z) = \begin{cases} (1 - z^2)^2 & \text{for } |z| < 1 \\ 0 & \text{for } |z| \geq 1 \end{cases} \tag{6}$$

Define e_i as the residual for current fitted values in Equation 7.

$$e_i = y_i - \hat{y}_i \tag{7}$$

Let s be defined as the median of $|e_i|$. Thus, robustness weights δ_i can be defined as in Equation 8.

$$\delta_i = B(\frac{e_i}{6s}) \tag{8}$$

Given these definitions, the robustification step consists of refitting a d-th degree polynomial with weighted least squares using the new weight function of $\delta_i \omega_i(x)$ for each (x_i, y_i). Thus, the robustification step consisting of finding the parameters $\hat{\beta}_l$ that minimize Equation 9.

$$\sum_{k=1}^{n} B(\frac{e_k}{6s}) \omega_i(x_k)(y_k - \hat{g}_i(x_k)^2) \tag{9}$$

This robustification process is repeated a predetermined number of t times. In total, this amounts to 3 hyperparameters in need of specifying: the fraction f, number of iterations t and the polynomial order, see [22]. According to [9], the choice of weight function $\omega_i(x)$ can also be chosen freely, however, the tri-cube is the most common choice.

3.2.2 Gaussian Process Regression

A Gaussian process can be viewed as a distribution over functions and can be formally defined as "a collection of random variables, any finite number of which have a Gaussian distribution" (Rasmussen et al., 2006, p. 13)[23]. All content in this section is retrieved from this source. Such a process $f(x)$ is described fully by its mean function $m(x)$ and covariance function $k(x, x')$ as seen in Equation 10.

$$f(x) \sim \mathcal{N}(m(x), k(x, x')) \tag{10a}$$

$$m(x) = \mathbb{E}[f(x)] \tag{10b}$$

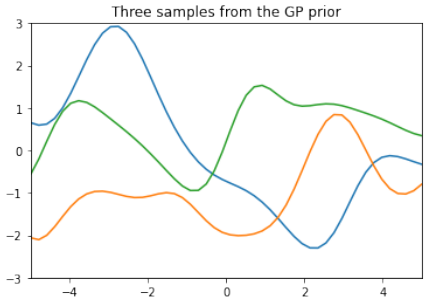
$$k(x, x') = \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))] \tag{10c}$$

Here, $N(*, *)$ denotes a Gaussian distribution, $\mathbb{E}[*]$ is the expectation and x and x' are independent variables for which $f(*)$ is the function value. Often, the data is adjusted to be mean zero, providing notational simplicity. In further descriptions, a collection of independent variables will be represented as $X \in \mathbb{R}^{n \times d}$, signifying both that every independent variable element may be multi-dimensional (dimension d) and that there are many (n) such variables.

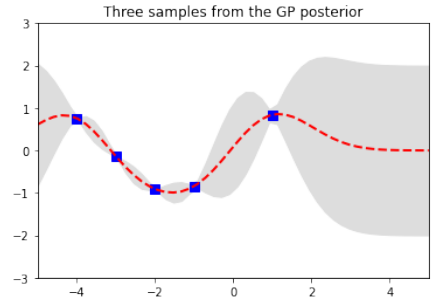
Specifying a covariance function implies a distribution over functions and so, choosing a number of input points X_* and writing out a corresponding covariance matrix $K(X_*, X_*)$ by comparing covariance functions elementwise, a random Gaussian vector can be drawn. This represents values from a possible function from the distribution of functions which is the Gaussian process. This can be done according to Equation 11.

$$\mathbf{f}_* \sim \mathcal{N}(0, K(X_*, X_*)) \tag{11}$$

Plotting the values generated as a function of inputs will provide a graph which will seem smooth when choosing a covariance function that is sufficiently differentiable. A squared exponential covariance function $k(x, x') = \sigma^2 \exp\left(\frac{||x - x'||^2}{2l^2}\right)$ is infinitely differentiable which causes the process to be infinitely mean-square differentiable. An example with three such samples are displayed in Figure 3a and are referred to as priors.



(a) Three samples of functions drawn from a Gaussian process with squared exponential covariance function where $l = \sigma = 1$.



(b) Posterior for a Gaussian process with same kernel given $X_{train} = [-4, -3, -2, -1, 1]$ and $y_{train} = sin(X_{train})$. The red line is the mean value for the posterior and the gray area marks $\pm$ 2 standard deviations

Figure 3: Examples of priors and posteriors for Gaussian processes with same squared exponential covariance kernel with $l = \sigma = 1$.

These functions are seen as priors in a Bayesian framework, since they are sample functions without any previous draws or data to compare with. A posterior, on the other hand, takes into account previous draws, i.e. given data, and fits the drawn function to these values. Thus, deriving a conditional distribution for predictions, given previous observations, gives the main equations for Gaussian Process Regression (GPR).

In the case when previous observations $\{(\mathbf{x}_i, f_i)|i = 1, ..., n\}$ (also referred to as training inputs/outputs) are known, the joint distribution for training and test inputs/outputs becomes that of Equation 12.

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right) \tag{12}$$

Test inputs/outputs ($X_*/\mathbf{f}_*$) refer to those drawn from the distribution given previous observations. Given n training points and n_* test points $K(X, X_*)$ is the resulting $n \times n_*$ covariance matrix evaluated between all respective pairs according to the covariance function and the other combinations of X and X_* follow the same structure.

The posterior distribution given the training data, consists of functions that agree with these observations. This can be likened to generating functions from the prior and disregarding those that do not agree with the observations. Conditioning the joint Gaussian prior distribution on the observations provides Equation 13.

$$\mathbf{f}_*|X_*, X, \mathbf{f} \sim \mathcal{N}\left(K(X_*, X)K(X, X)^{-1}\mathbf{f},\right.$$
$$\left. K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*)\right) \tag{13}$$

From this joint posterior, functions f_* for corresponding test inputs X_* can be sampled. The mean value of this posterior is sometimes referred to as the linear predictor ($\bar{f}_*$) and represents the best guess of $f(\mathbf{x})$ for a given $\mathbf{x}$. The variance is simultaneously useful for representing uncertainty of fit. An example of such a posterior is seen in Figure 3b where the mean is marked in red and ± 2 standard deviations are represented by the gray area. Note, that in the figure, the mean follows the sinusoidal shape, which is the true function that produced the observations, close to the observations. Also, the mean value function $f_*(x)$ returns to zero further from these.

The covariance function is often called a kernel due to it being a symmetric function of two arguments mapping a pair if inputs $\mathbf{x}, \mathbf{x}' \in \mathbb{R}$ to $\mathbb{R}$. A common choice of kernel is the squared exponential kernel. This is a kernel in the form of Equation 14.

$$k(x_i, x_j) = \sigma_f^2 \exp\left(-\frac{(x_i - x_j)^2}{2l^2}\right) + \sigma_n^2 \delta_{ij} \tag{14}$$

This covariance function has three hyperparameters: length-scale l, signal variance σ_f^2 and noise variance σ_n^2.

For further information of how noise is included in the model and how the hyperparameters are tuned using an optimizer, see Section B in the Appendix.

3.3 Density-Based Spatial Clustering of Applications with Noise

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is an algorithm that clusters spatial data based on their density, see [24]. The clustering method can cluster arbitrary shapes but requires that clusters have similar densities. One use of this algorithm is that it can be used to filter out noisy and sporadic points. There are two main parameters for the algorithm: ϵ is the maximum distance between samples for for them to be in the same neighbourhood and $minPTS$ is the minimum number of samples in a neighbourhood for a point to be considered a core point. For a more in depth explanation, see [2].

3.4 Background Filter

There are many different methods for filtering out background, which is evident in Section 1.3. One such filter is the 3D Density Static filter (also referred to as Density Filter), introduced in [3] and expanded on in [4] and [5]. The filter works by dividing volumetric space into cubes/voxels, for which all points in a

frame are mapped. Points are filtered based on the frequency of points in that 3D space. The idea is that space which contains many points tends to belong to the background and thus points appearing in this space is likely also background.

The Density Filter has to be trained to gain information of the point density statistics. This is done by aggregating point counts for every cube for a number of initial frames. The number of frames f necessary depend on a lot of different factors such as the side length of cubes l and the type of background. This produces a grid with accumulated point counts which can thereafter be used for filtering. Filtering is done by removing points in voxels that have a count over a threshold T.

Points are kept or removed on the basis of counts in 3D cubes. This means that side length l implies a certain resolution to the filtering. Herein lies a tradeoff: large side lengths cause lower spatial resolution, meaning that points that do not belong to background might be filtered out when close to the actual background. Short side-lengths however, require more memory for the model and are more affected by the LiDARs scanning pattern.

3.5 Semantic segmentation

Semantic segmentation is the task of determining the unique label of all elements in a set of data. This could be for a RGB-image to determine the specific label for each pixel. For point clouds, semantic segmentation instead means deciding what class each point belongs to. This is done using features such as spatial relations, RGB-values and reflectivity. The task is usually very computationally heavy due to the large amount of points, added depth dimension and the sparsity of point clouds. In this thesis, semantic segmentation was performed by a neural network called RandLA-Net.

3.6 Neural networks

Multilayer perceptrons (MLP) or feedforward neural networks are frequently used networks within data science with the goal of approximating some function. The following content on neural networks, loss functions and optimizers is from [25]. The idea is to map an input $\mathbf{x}$ to an output $\mathbf{y}$ (ground truth). This is done using a function $\mathbf{f(x;}\theta\mathbf{)}$ where θ are weights which are iteratively updated during training to minimize the difference between $\mathbf{y}$ and $\mathbf{f(x;}\theta\mathbf{)}$. The name neural networks comes from the fact that the networks are loosely inspired by the human brain. As the name suggest the model is a network, the network consists of several layers which are visualised as a directed acyclic graph describing the relationships between nodes in different layers. The different layers correspond to the depth of the model.

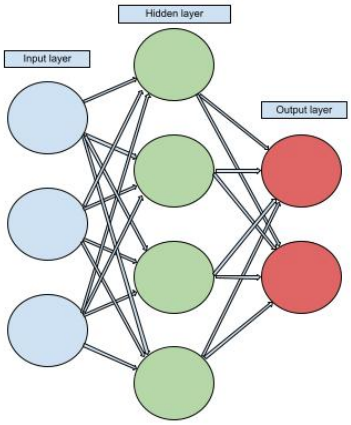


Figure 4: Layers of an multilayer perceptron shown as nodes in a directed acyclic graph.

Each component within a layer is a node that has an input and an output. The relationship between input x_j and output z_i can be seen in Equation 15 and Figure 5. The input to a node is multiplied with a weight w_i, j added with a bias b_i and passed through an activation function g. A commonly used activation function is the ReLU-function (shown in Equation 16). The activation functions used in this thesis is the leaky ReLU-function in Equation 17 and the softmax function in Equation 18. In equation 17, the coefficient α is generally small ($\alpha << 1$).

$$z_i = g(\sum_{j=1}^{N} w_{i,j} x_j + b_i) \tag{15}$$

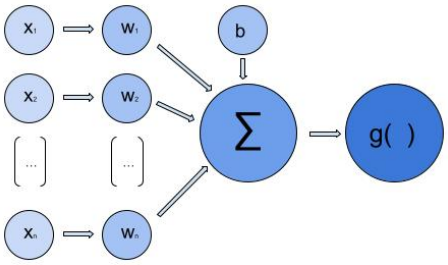


Figure 5: A single perceptron with weights and activation function.

$$g(x_i) = \begin{cases} x_i & \text{for} \quad x_i \geq 0 \\ 0 & \text{for} \quad x_i < 0 \end{cases} \tag{16}$$

$$g(x_i) = \begin{cases} x_i & \text{for} \quad x_i \geq 0 \\ \alpha x_i & \text{for} \quad x_i < 0 \end{cases} \tag{17}$$

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{18}$$

3.6.1 Loss function

As a deep learning network tries approximate the function **f(x:θ)** with respect to **y**, the way to measure the difference is by using a loss function **L(x:θ)**. RandLA-NET utilizes a loss function called **weighted cross entropy**, seen in Equation (19).

$$L_{wce} = -\frac{1}{N} \sum_{n=1}^{N} \sum_{k=1}^{k} w_k \cdot y_n^k \cdot \log\left(h_\theta\left(x_n, k\right)\right) \tag{19}$$

In Equation 19, from [26], w_k corresponds to the class weight, y_n^k true label value, $h_\theta\left(x_n, k\right)$ model with neural weights. The class weights correspond to the inverse of the class frequency which gives a larger penalty for missclassifying points that are unusual. This makes the weighted cross entropy loss function useful when there is a large class imbalance. This is the same in this case with point clouds to enforce more correct predictions on the vegetation class.

3.6.2 Optimizer

An optimizer is a rule of how to minimize the loss function with respect to the weights θ. A commonly used optimizer is Stochastic Gradient Descent (SGD) which updates the weights by the gradient of the loss function using a randomly selected subset of the data. This can be seen in Equation 20 where η is a parameter that decides the size of the update step and $\nabla L(\theta)$ is the gradient of the loss function.

$$\theta_t = \theta_{t-1} - \eta \nabla L(\theta) \tag{20}$$

RandLA-NET uses the Adam optimizer that uses estimations of first and second degree moment to smooth the stochastic gradient descent updates, see [27]. The update steps can be seen in Equation 21 where m_t and v_t are the first and second order moments which are calculated from the gradient $\nabla L(\theta)$.

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{21}$$

The gradients of a loss function with regards to the trainable parameters are calculated through an algorithm called backpropagation. Here, information flows backwards through the network by recursively applying a chain rule to elements in a computational graph.

3.7 RandLA-Net

RandLA-Net, described in [28] is a point cloud segmentation network that outperformed many other networks in speed and accuracy when it originally was invented in 2019. The key to the effectiveness of the network is the use of random sampling to down-sample the point cloud in several iterations. This make the network both memory and computationally efficient and able to run the point cloud in a single pass as the random sampling has time complexity O(1). However random sampling induces loss of information, especially on very sparse point sets. To compensate for this, a local feature aggregation module is added that retains spatial information of neighbouring points. This local feature aggregation consists of a local spatial encoding, attentive pooling and a dilated residual block.

3.7.1 Local Spatial Encoding

Local Spatial Encoding (LocSE) is introduced to find spatial relationships and dependencies between points in the point cloud. For a point cloud $\mathbf{P}$, where each point $\mathbf{p_i}$ has a set of features, the nearest neighbours are found using K-nearest neighbours (KNN). For each neighbouring point ($\mathbf{p}_i^k$) encoded point features ($\mathbf{r_i^k}$) are collected using an MLP on the relative point positions. This can be seen in Equation (22). Thereafter the encoded point positions $\mathbf{r}_i^k$ are concatenated with the local point features to output an augmented feature $\mathbf{f}$. These augmented features are further used to create a set of neighbouring features $\hat{\mathbf{F}}_i = \left\{\hat{\mathbf{f}}_i^1 \cdots \hat{\mathbf{f}}_i^k \cdots \hat{\mathbf{f}}_i^K\right\}$.

$$\mathbf{r}_i^k = MLP\left(p_i \oplus p_i^k \oplus \left(p_i - p_i^k\right) \oplus \left\|p_i - p_i^k\right\|\right) \tag{22}$$

3.7.2 Attentive Pooling

Attentive pooling aims to find an attenuation score for all features which is used to weight each feature based on its importance. This is shown in Equation 23 where Equation 23a finds the attention score using the function g which is an MLP (with trainable parameters $\mathbf{W}$) followed by a softmax function. In Equation (23b) the features are ultimately received using the attention score. Here, s_i^k are the unique attention scores for each feature and $\hat{\mathbf{f}}_i^k$ are the learnt attention scores through weighted summation.

$$\mathbf{s}_i^k = g\left(\hat{\mathbf{f}}_i^k, \boldsymbol{W}\right) \tag{23a}$$

$$\tilde{\mathbf{f}}_i = \sum_{k=1}^{K}\left(\hat{\mathbf{f}}_i^k \cdot \mathbf{s}_i^k\right) \tag{23b}$$

3.7.3 Dilated Residual Block

A dilated residual block is the concatenation of several LocSE and attentive pooling layers to greatly increase the receptive field of each point. According to the authors of RandLA-NET this residual block, for a point $\mathbf{p}_i^k$, with K nearest neighbours, increases the receptive field with K^2 points. More blocks increase the sphere of reach of each point, but it also makes the network more computationally heavy and likely to overfit. In [28], the authors advised using only two LocSE and attentive pooling layers in a block. Figure 6 shows the steps of random sampling and point information passing within a dilated residual block.

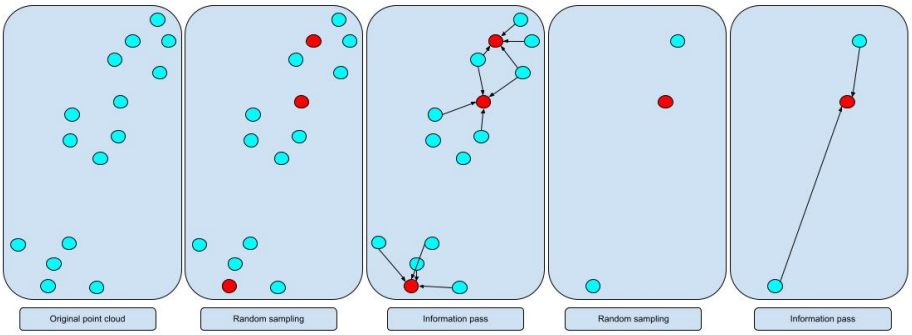


Figure 6: Random sampling and point features passing during two random sampling steps.

3.7.4 Inference

In RandLA-NET, inference is conducted by iteratively choosing center-points **w** to originate from. From each center-point, it computes a subset of the point cloud corresponding to the N closest points to the center point by using a k-d tree. This cloud of N points is then passed through the network. Depending on if the network is training or testing, the choice of center point **w** varies. Whilst testing, it is important to ensure that all points in the original point cloud are passed through the network at least once. Therefore, RandLA-NET uses a *possibility iteration scheme*, see Section 3.7.6. During training these center points are randomly selected. This procedure ensures that the sub-point-clouds that pass through inference remain the same size.

3.7.5 K-d tree

K-d trees are a data structure used to partition and store k-dimensional data into binary trees. They use hyperplanes in k-dimensions to divide the data in each node so that points in the left hyperplane are under the left child-node and respectively, points in the right hyperplane are in the right child node. K-d trees

are used in RandLA-NET to effectively select the sub-point-cloud consisting of the N closest points from a center point.

3.7.6 Possibility iteration

The center point **w** is selected using the previously mentioned *possibility iteration scheme* where at each iteration, the point with lowest possibility is chosen as the center point. Then a sub-point cloud is selected from this center point using the k-d tree and an inference is done. The possibility of all points in the sub-point-cloud is updated following Equation 24 where **w** is the center point, $\mathbf{P}_i$ is the sub-point cloud of index i and Δd_i is an array of all point distances to the center point in sub-point cloud i and **possibility** is an array of possibilities for every point. Inference is repeated until all points in the total point cloud have a possibility larger than a threshold ρ.

$$\Delta \mathbf{d}_i = \|\mathbf{P}_i - \mathbf{w}\|^2 \tag{24a}$$

$$\delta_i = \frac{1 - \Delta d_i}{MAX(\Delta \mathbf{d}_i)} \tag{24b}$$

$$\mathbf{possibility} += \delta_i \tag{24c}$$

3.7.7 Voting scheme

As RandLA-NET inference many sub-point-clouds so their resulting predicted labels must be stitched together. This is done using the voting scheme shown in Equation 25 where $P_{i,j}$ is the probability for point j from point cloud i, α is a smoothing constant, $Pred_{i,j}$ is the current inference prediction and $P_{i,j}^{init}$ is the initial value for $P_{i,j}$.

$$\begin{aligned} P_{i,j} &= P_{i,j} \cdot \alpha + Pred_{i,j} \cdot (1 - \alpha) \\ P_{i,j}^{init} &= 0 \end{aligned} \tag{25}$$

3.7.8 Grid subsampling

Before inference in RandLA-Net, the point clouds are preprocessed using a grid subsampling strategy. This is dividing the 3D space in small voxels and outputs only one point per voxel. This is effective in reducing the number of points and the point density in very dense areas.

3.7.9 Network architecture

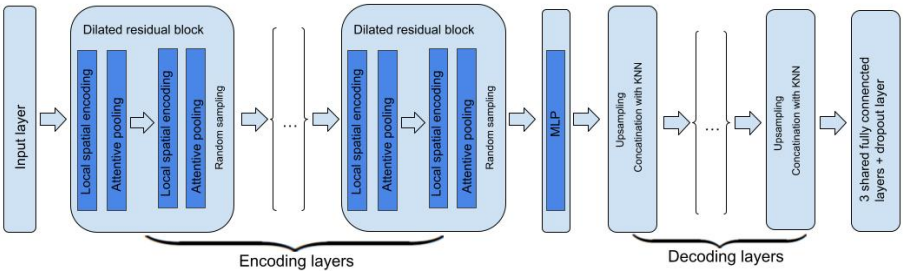


Figure 7: Network architecture of RandLA-NET.

Input to the network is a point cloud with dimensions N x d_{in} where N is the number of points and d_{in} are the point features that can vary between datasets. In the dataset from CARLA, d_{in} is the x,y,z coordinate values. Thereafter there are a number of **encoding layers**. These encoding layers consists several of *dilated residual blocks* together with random sampling. This random sampling layer down-samples the points with a factor of 4, at the same time the point feature dimension is increased with a factor of 4. Hereafter, the **decoding layers** are applied. Features from the closest point, chosen with k-nearest neighbours (KNN), in the upstream layer are applied to all points and concatenated from features from the encoding layer (through skip connections) and applied to an MLP. Lastly the prediction is done through three fully connected layers and a dropout layer. An image of the basic network architecture can be seen in Figure 7.

3.8 Evaluation Metrics

When evaluating models there are a few important metrics that are necessary to explain. In the binary case, when there are predictions of positive and negative outcomes, there are four possible cases: *True Positive* when a positive outcome is correctly classified, *True Negative* when a negative outcome is correctly classified, *False Positive* when a negative outcome is miss-classified as positive and *False Negative* when a positive outcome is miss-classified as negative. These cases are shown in a contingency table in Table 1.

Table 1: Table showing how binary classification is arranged in a contingency table.

Actual \ Assigned	Prediction Positive	Prediction Negative
Positive	TP	FN
Negative	FP	TN

In the multi-class case, these outcomes are defined individually for each class. In this case, TN refers to all correct cases of a model not predicting instances belonging to a class and TP is the same for instances belonging to the class. FP refers to incorrectly predicting instances belonging to a class when they do not and lastly, FN refers to incorrectly predicting instances belonging to a class as not doing so.

In [29], accuracy, precision, recall, specificity and F_1 scores are defined. Accuracy quantifies number of correct predictions compared to the total number of predictions. In the binary case, this is seen in Equation 26.

$$\text{accuracy} = \frac{TP + TN}{\text{TOTAL}} \tag{26}$$

Precision is a measure of how sure the model is that a positive outcome is correct: of predicted positive outcomes, how many positive predictions are correct. This is seen in Equation 27.

$$\text{precision} = \frac{TP}{TP + FP} \tag{27}$$

Recall, sometimes referred to as sensitivity, is a measure of how good the model is at correctly predicting positive outcomes: of actual positive outcomes, how many are correctly predicted as positive. This is seen in Equation 28.

$$\text{recall} = \frac{TP}{TP + FN} \tag{28}$$

Specificity, sometimes referred to as true negative rate, is a measure of how good a model is at predicting negative outcomes. How many of the negative outcomes were correctly classified as negative. This is seen in Equation 29.

$$\text{specificity} = \frac{TN}{TN + FP} \tag{29}$$

The F_1-score, sometimes referred to as simply F-score or F-measure is the harmonic mean of precision and recall . It is a value between 0 and 1 ($F_1 \in [0, 1]$) that signifies how well a model classifies positive cases. Further information is included in Section A, in the appendix. This is seen in Equation 30.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{30}$$

IoU, short for *Intersection over Union*, is a standard measure within point cloud semantic segmentation and is used to compare different deep learning architectures against eachother. It measures the quota between the true positives and the union of the predicted and true labels. This is seen in Equation 31.

$$IoU = \frac{TP}{TP + FP + FN} \tag{31}$$

The mean IOU is calculated as the mean of IoU over all classes. The mean forces all classes to be equally important regardless of the number of points. Similarily to the weighted cross entropy loss function, it enforces good prediction of all the classes and not only the most frequent.

Root Mean Square Error (RMSE) is a measure of the difference between observed values y and values predicted by some kind of an estimator $\hat{y}$. This is seen in Equation 32.

$$RMSE = \sqrt{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{32}$$

3.8.1 Confusion Matrix

Confusion matrices are matrices that show how the classification varies over different classes. The rows are the predicted label and columns are the true value. Each element on the diagonal are correctly predicted instances. It provides an easy interface to see which classes are mixed up in a prediction.

4 Method

4.1 Simulation of data

This section describes how data was generated using CARLA. Noise was added to the data to better resemble that of the RP-LiDAR and this was mostly done by adding noise in the radial direction (which is the most dominant noise direction for the RP-LiDAR).

4.1.1 CARLA LiDAR Data

Only a small number of point clouds generated by CARLA's native LiDAR were used. These were used for testing the ground model. The point clouds were produced by randomly spawning a LiDAR in different locations with random yaw angles and a pitch of $-15°$. Noise was added by first defining an attenuation where points were randomly removed with a probability that increases with distance according to an exponential distribution with $\lambda = 0.015$. Thereafter, uniform noise was added in all directions of $\mathbb{U}[-0.5\lambda, 0.5\lambda]$ with $\lambda = 0.05$ for all points but for two exceptions. Vegetation was given uniform noise with $\lambda = 0.25$ and poles were given Gaussian noise with standard deviation $\sigma = 0.03$. All units are in metres.

4.1.2 Graphic LiDAR Dataset

The Graphic LiDAR dataset, was designed for training, validation and testing a neural network for semantic segmentation of point clouds. Four point clouds were also used to test and compare ground models.

Data was simulated using the Graphic LiDAR plugin described in Section 2.4.2. Point clouds were produced by placing LiDAR-spawnpoints within a CARLA simulation. These spawn points were chosen with suitable starting position and orientation for its view to include several classes, but also to imitate how a real world stationary LiDAR would be placed. Then, the LiDAR iterated over the spawn points multiple times, with uniform random pitch between $-10°$ and $-15°$ and uniform random yaw between $-30°$ and $30°$, and capture one frame per iteration. 810 frames each, were simulated in validation and test sets, and 3885 frames were simulated in the training set. This amounted to 15 % validation, 15 % test data and 70 % training data. These numbers were chosen on the basis of being a common choice for this type of split. The frames captured from the test, validation and train datasets come from different simulation-worlds in order to decrease similarities and to avoid overfitting on given environments.

Table 2: Distribution of frames and targetpoints in different datasets simulated from carla. For the column "Class balance", the numbers describe the percentage of points belonging to classes ground, vegetation and other.

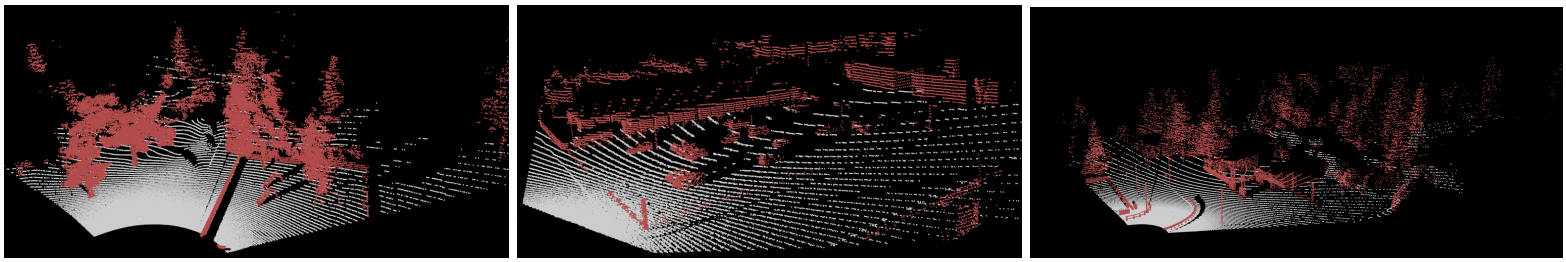
Test		Spawn points	Frames	Class balance
	Town 1	54	810	65.2 5.4 29.3
Total		54	810	
Val				
	Town 2	54	810	62.5 4.3 33.1
Total		54	810	
Train				
	Town 4	28	420	62.2 28.8 9
	Town 5	48	720	70.5 9.4 20.1
	Town 6	69	1035	71.0 12.7 16.2
	Town 10	114	1710	63.9 5.42 30.6
Total		259	3885	

Each frame contained roughly 45K points. This means that the entire dataset contained roughly $2.5 \cdot 10^8$ points.

Noise was added to all point clouds by creating a scrolling noise texture of correlated Gaussian noise to the depth value of points.

4.1.3 Ground model Dataset

For testing the ground models, six point clouds were chosen with different characteristics for comparison. These six were divided into two different sets of point clouds: *Flat terrain* and *Hilly terrain*. It is good to point out that only *parking lot* had truly flat ground. The other two *Flat terrain* point clouds had ground undulation of a few meters, which is much flatter than the hilly point clouds.



(a) Point cloud of a playground with surroundings.

(b) Point cloud of a parking lot.

(c) Point cloud of a petrol station.

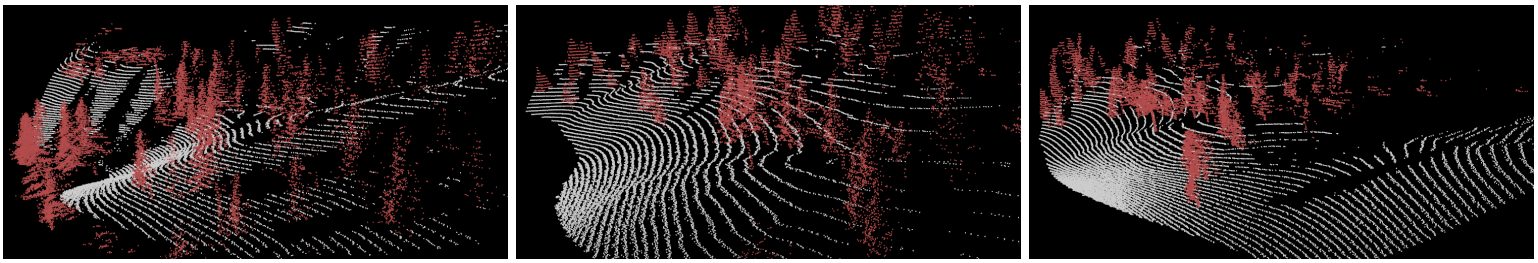
Figure 8: Images of the point clouds belonging to the flat dataset. White points are ground points and red points are other.

The *Flat* point clouds are seen in Figure 8 and *Hilly* are seen in Figure 9. In Table 3 below, the *Flat terrain*-dataset is described in more detail, partly with all points in the point clouds and partly with all points with distances between $2m$ and $100m$ from the sensor.

Table 3: Number of points for different categories for the different point clouds in the *flat terrain*-dataset. The subset-point clouds refer to all points between distances $R \in [2m, 100m]$ to the sensor.

Point cloud	Type	Total	Ground	Other
"Playground"				
	All points	92 041	49 471	42 570
	Subset	92 041	49 471	42 570
"Parking lot"				
	All points	45 914	34 983	10 931
	Subset	43 237	34 777	8 460
"Petrol Station"				
	All points	93 546	71 155	22 391
	Subset	93 546	71 155	22 391
Total				
	All points	231 501	155 609	75 892
	All points %		67.22%	32.78%
	Subset	228 824	155 403	73 421
	Subset %		67.91%	32.09%

All point clouds in the *Hilly terrain*-dataset were from the *Graphic LiDAR Dataset*, described in Section 4.1.2, from the CARLA's "Town 5" environment. They are of the terrains in CARLA that had most ground undulation and were quite similar. They were chosen for displaying different types of hills and can be seen in Figure 9.



(a) Hill 1: hill with trees furthest down. (b) Hill 2: long hill with trees on top. (c) Hill 3: hill with larger hill to the right.

Figure 9: Images of the point clouds belonging to the *Hilly dataset*. White points are ground points and red points are other.

In Table 4, the *Hilly terrain*-dataset is described in more detail.

Table 4: Number of points for different categories for the different point clouds in the *Flat terrain*-dataset. The subset-point clouds refer to all points between distances $R \in [2m, 100m]$ to the sensor.

Point cloud	Type	Total	Ground	Other
"Hill 1"				
	All points	42 504	19 507	22 997
	Subset	41 960	19 093	22 867
"Hill 2"				
	All points	33 630	20 851	12 779
	Subset	29 966	19 457	10 509
"Hill 3"				
	All points	42 132	28 461	13 671
	Subset	40 545	27 862	12 683
Total				
	All points	118 266	68 819	49 447
	All points %		58.19%	41.80%
	Subset	112 471	66 412	46 059
	Subset %		59.05%	40.95%

4.2 Ground Models

4.2.1 Implementation of RLWR-based Model

The model was implemented based on the description provided in the proposed article, see [9]. The model took a point cloud as input and returned points predicted to be ground.

The implementation was carried out as follows. First, the point cloud was divided into $x-z$ profiles with width dy and $y-z$ profiles with width dx. Thereafter, every such profile was fed into a RLWR-iterator algorithm consisting of a number of subtasks which were iterated. First, the z-values were shifted so that the minimum value was zero. Thereafter RLWR-regression was performed using a linear fit as polynomial fit. This was performed for a specified number of iterations (t) and a specified fraction (f). Thereafter followed 4 subtasks:

Task 1: Residuals $r_i = z_i - \hat{z}_i$ are calculated where $\hat{z}_i$ is the fit.

Task 2: The points are divided into those above and those under the fitted RLWR-line.

Task 3: Down-weighting was performed using the bisquare robust weight function in Equation 6 on the points above the fitted line. Weights were produced by assigning a weight of $w_i = 1$ for points under the line and $w_i = B(r_i^*)$ above the line where $r_i^* = \frac{r_i}{6 \cdot \text{median}(|\mathbf{r}|)}$ for points over the

line. Re-weighting the z-values was carried out by element-wise multiplying the weights with corresponding z_i and assigning these as the new z-values. Following re-weighting, low outliers were adjusted. This was done for a neighborhood of a point consisting of $k = \text{ceil}(f \cdot n)$ closest points. Here, a check was made whether the given point was lower than the lowest point in its neighborhood. If it was, its value was replaced by the lowest.

Task 4: The RMSE (specified in Equation (32)) was calculated between the values of the RLWR-regression and the down-weighted z-values. The new set of z-values was fed to the RLWR-regression to get the next fit and tasks 1-3 were repeated until the change in RMSE (dRMSE) was less than a specified threshold δ. The last RLWR fit was used as the final fit for determining the height of the ground along the given stripe. The final ground points were chosen as the initial $z - values$ (dependent variables) that were within a threshold (T) of this determined ground.

The common ground points were decided as the intersection of ground points for both overlapping $x - z$ and $y - z$ profiles, i.e. the points classified as ground points for both stripes. The threshold for determining ground points for stripes in either direction was allowed to vary due to varying spacial densities for these directions. With sufficiently narrow stripes, the thought was that variations along the complementary horizontal axis was limited, so that a good estimation of the ground level could be achieved. This is however a trade-off, since narrower stripes give fewer points and a less robust fit.

In total, this algorithm required 7 hyperparameters: dx, dy, f, t, δ, T_x and T_y

4.2.2 Implementation of Hybrid Regression Model

The implementation of the hybrid regression model was very similar to the description provided in the original paper [10]. The main difference was in the point cloud representation and the addition of DBSCAN. In the article, the considered datasets all consisted of a spinning LiDAR scanner, producing a point cloud with a 360° view. The data considered in this thesis consisted of a stationary LiDAR with a 120° view. Thus, the polar grid representation in the model implemented here consisted of a slice slightly larger than 120°. The model took a point cloud as input and returned predicted ground points and a ground function $z = f(x, y)$ as output.

The model was implemented in the following way. First, all points in a provided point cloud were filtered through DBSCAN with $\epsilon = 1.5$ and a minimum of 5 samples for a point to be a core point. This was to remove noisy points. All points within a radius of R_{max} were thereafter assigned into a grid pattern displayed in Figure 10. This was done for N circles, M segments and $N \times M$ *bin*:s. The bins were the intersections of segments and circles. The sizes and

numbers of these were specified by a radial length Δr and angle $\Delta\alpha$. The assignment was given by

$$seg(p_i) = \left\lceil \frac{\text{atan2}(x_i, y_i)}{\Delta\alpha} \right\rceil, \tag{33}$$

and

$$circ(p_i) = \left\lceil \frac{\sqrt{x_i + y_i}}{\Delta r} \right\rceil. \tag{34}$$

For each *bin*, the lowest height was selected to create a skeleton of the ground. Thereafter the set of these lowest points were arranged into segments (SL_m). Despite consisting of the lowest point in every *bin*, the points were not necessarily ground points and were therefore fed through an RLWR-regression and a gradient filter before being assumed to be ground.

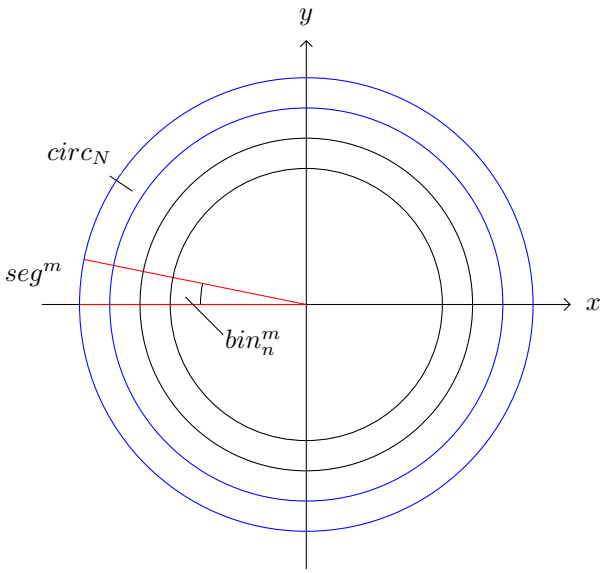


Figure 10: Polar grid map for projecting point cloud into grids. Segments are the divisions for different azimuthal angles (area between red lines) that go in the radial directions whereas the circles are the areas between given radius from the origin (area between blue circles). The bin:s are the intersections between segments and circles.

The points of every SL_m were fed into RLWR, using radius r as independent variable and z as dependent variable. This is done given t-iterations and a specified fraction f. The fitted heights were thereafter fed into a gradient filter. This filter calculated the gradient for every (r_i, z_i) pair. This was done, given second order accurate central differences of interior points and first or second

order at the sides using numpy's gradient function, see [30]. Any gradients with a slope larger than a specified β_{max} were replaced by the value of the nearest point for which the slope was less than this. The remaining set of points along a segment (SL'_m) were thereafter used as a seed skeleton for GPR along the angular direction α.

Before performing the GPR, the seeds were rearranged circle-wise SL_n. For every circle, the mean height value was calculated and subtracted from the seeds SL_n. The azimuthal angle α was calculated for all points. Thereafter, given the seeds of each circle, a Gaussian Process regression model was fitted for α_i, z_i pairs and the result of this was used to predict the heights of all points in the circle. The kernel used was the squared exponential kernel specified in Equation (14) with initial values $\theta = (l, \sigma_f^2, \sigma_n^2)$. The GPR implementation used was scikit-learn's *GaussianProcessRegressor* provided by their API. The fit was carried out using a Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimizer algorithm from *scipy.optimize.minimize*, see [31].

The predictive height for the model was the mean predicted height of the points in a bin according to the GPR. When filtering points, all points within a threshold value T of this height for a bin were considered part of the ground. A ground function $z = f(x, y)$ was constructed by returning a function that mapped every x, y value to a bin and returning the bin's average predicted height. If there was no such bin (i.e. the (x, y)-position was outside the point clouds area) it was set to return a $None$-value. For bins that had no points but were inside the point-cloud's area, the predicted height was used of a point with an angle α in the centre of the bin. This was done using a GPR fitted with the circle's seed points. Also given, was a grid for which the predicted mean height of every relevant bin along with x and y values for the centre of these.

In total, there were 10 hyperparameters for the implementation: the radial width of bins/circles Δr, the angular width of bins/segments $\Delta\alpha$, the maximum distance considered for the point cloud R_{max} the fraction of points taken into account in the RLWR f, the number of iterations in the RLWR t, the maximum slope by which points are filtered β_{max}, the initial values for the kernel in the GPR $\theta = (l, \sigma_f^2, \sigma_n^2)$ and the threshold for which points were filtered by T.

4.2.3 Implementation of Plane Model

The plane model was the simplest model used. It used the RANSAC algorithm provided by *Open3D*'s API through their *segment_plane()* function, see [21]. The model took a point cloud $\mathbf{P}$ and a threshold T. Using the RANSAC algorithm with error tolerance δ, fitted on n points, for t iterations, the best plane was fit to the point cloud. The heights $\hat{z}$ of all points in the cloud were predicted by using the plane equation in Equation (1), i.e.

$$\hat{z} = f(x, y) = \frac{-ax - by - d}{c}. \tag{35}$$

All points (x_i, y_i, z_i) for which z_i was within the threshold of the plane's height $|z_i - \hat{z}_i| < T$ were considered ground points by the model.

4.3 Test on Ground models

The three models were tested on the ground model dataset described in Section 4.1.3, divided into *Hilly* and *Flat* terrain. For each such subset of point clouds, every ground model was tested for a few different choices of hyperparameters. The models were tested on their ability to filter out ground points and relevant scores were calculated. Table 5 below, show the choices of hyperparameters for the different types of models evaluated on the two sets of point clouds. These remained the same for all runs.

Table 5: Hyperparameters of the different Ground models tested.

Hybrid Model					
Parameters:	f	t	β_{max}	$\theta = (l, \sigma_f^2, \sigma_n^2)$	T
	0.1	5	10	$(0.1935, 0.2415, 0.0396)$	1
RLWR-Based Model					
Parameters:	f	t	δ	T_x	T_y
	0.1	5	0.1	$2m$	$1m$
Plane Model					
Parameters	δ	n	t		
	0.001	3	1000		

All models except for one plane model were fit on a subset of points $\mathbf{P}'$. This subset constituted of all points with a distance between $R_{max} = 100m$ and $R_{min} = 2m$ of the LiDAR, i.e. $\mathbf{P}' \in [R_{min}, R_{max}]$. A subset of the points were chosen due to the properties of point clouds generated by stationary LiDAR. Generally, points within $2m$ were noise and therefore there was no greater purpose of fitting ground within this radius. These point clouds also decreased in point density at further distances due to trigonometry but also occlusion. In the data, points beyond $100m$ tended to be too occluded to provide any valuable information.

For the Hybrid and plane models the threshold was set to $T = 1m$ and for the RLWR based model the thresholds were $T_x = 2m$ and $T_y = 1m$. Due to two reasons, larger values of T_x and dy were assigned. Firstly, RLWR had a more complicated process filtering points. Ground points had to be chosen by RLWR filterings in both x and y directions. Secondly, due to the scan pattern of a stationary LiDAR being very dense along azimuthal scans, (stripes in y-direction) and a lot less dense in the orthagonal x direction. The choice of initial θ for the

Hybrid models was set according to what the authors set in the paper [10].

Table 6 shows how each type of model was varied to explore the impact of given hyperparameters. The Hybrid and RLWR-based models were varied in choices of grid sizes (dx, dy) and $(dr, d\alpha)$ respectively, whereas the plane model was varied in choices of points from point cloud. One plane model was fit using all points and the other was fit using the same points as the other models $\mathbf{P}' \in [R_{min}, R_{max}]$. The "Small grid" for the hybrid model was chosen according to the grid size used by the authors of [10].

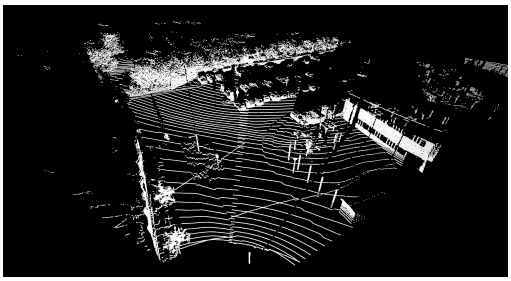
Table 6: Specifications of the different Ground models tested.

Model Type	**Model**		
Hybrid Model		dr	$d\alpha$
	Large grid:	$1.2m$	$12°$
	Medium grid:	$0.6m$	$6°$
	Small grid:	$0.2m$	$2°$
RLWR-Based Model		dx	dy
	Large grid:	$1.0m$	$2.0m$
	Medium grid:	$0.5m$	$1.0m$
	Small grid:	$0.25m$	$0.5m$
Plane Model		R_{max}	R_{min}
	fit on all points:	$1000m$	$0m$
	fit on subset of points:	$100m$	$2m$

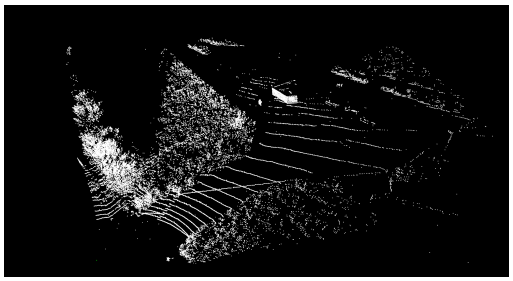
The set of grid sizes in Table 6 with overall best performing model, was thereafter used for every model-type to produce plots of filtered point clouds.

4.4 Evaluation of Ground model on point clouds

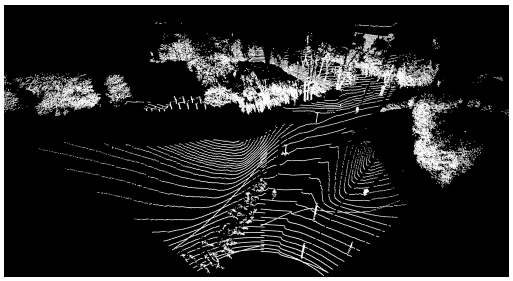
A hybrid model was visualised on three point clouds generated by the RP-LiDAR. The model chosen was that which gave the best overall performance from Section 4.3. The data consisted of three point clouds seen in Figure 11 from separate recordings and environments, all recorded using a trapezoidal scan pattern with the RP-LiDAR.



(a) Point cloud referred to as *Emdala*.



(b) Point cloud at Eslöv airport.



(c) Point cloud referred to as *Matteannexet*.

Figure 11: Three different point clouds that the ground models were tested on.

In Table 7 below, total number of points for the three point clouds are shown.

Table 7: Specifications of the point clouds studied.

Environment	Point Count
Emdala	57504
Eslöv airport	29136
Matteannexet	59451

4.5 RandLa-NET

4.5.1 Data preprocessing

Firstly, before using data as input to the network, it was pre-processed. This was done by gridsubsampling it according to cubes of side length 0.06. This was done to decrease the point density at certain locations. For the testing and validation sets, a projection file was saved containing the mapping from the subsampled to the original point clouds. Thereafter KD-trees for all files were created and saved, these KD-trees were calculated using the *scikit-learn* library [32].

4.5.2 Training of network

Different models of RandLa-NET were trained and tested with variable number of layers and KNN neighbours to predict how these values affect the network. The models where trained at 50 epochs and the number of input points to the network was set to 30 000 (25 000 with KNN as 24) due to GPU limitations. 50 epochs was chosen as the training seemed to converge after this amount of epochs. Each training was done with inital learning rate of 0.01 with a decay of 0.95 after each epoch. Batch size was set to 4 and validation/test batch size was set to 15. The batch sizes was chosen as they were the maximal sizes the network could be run without GPU memory overload.

Table 8: Specifications of the different Randla-NET models tested on the CARLA dataset.

Model name	No. classes	KNN	No. layers	Epochs	Input points
Normal	3	16	4	50	30 000
Normal-25K	3	16	4	50	25 000
2_layer-model	3	16	2	50	30 000
6_layer-model	3	16	6	50	30 000
Normal_knn_32	3	24	4	50	25 000
Normal_knn_8	3	8	4	50	30 000

The training of the network was carried through whilst computing the validation and training accuracy. This was done with the aim of detecting overfitting.

4.5.3 Testing of the network

The testing of the network was carried out on pointclouds simulated from a separated environment (a different town) see Table 2. The voting threshold ρ was chosen as 0.5. The testing was done by performing inference on the testing frames and computing a confusion matrix over the classes. The prediction on 5 frames of CARLA data was visualised. This was done to provide an understanding of how the semantic segmentation performs on the test dataset. Thereafter the network was applied on 5 different RP-LiDAR recordings (Grenden, Lomma beach, Matteannexet, Hörby and Eslöv airport) and the prediction was plotted. We do not have the ground truth as these are real LiDAR recordings therefore the only way to test the network is by visual inspection. The total network prediction time and batch inference time was saved. This was done to determine whether or not the network could be used in real time.

4.6 Improvement to the Background Density Filter

The Density filter in Section 3.4 was expanded on in order to incorporate semantic information with the purpose of improving its ability of filtering out vegetation. The filter required the existence of a model (neural network) that could perform semantic segmentation on relevant point clouds. The filter used this semantic information to strictly filter out points in areas considered vegetation, whilst simultaneously performing the same standard filtering based on a threshold.

During the training phase, the conventional count aggregation was, for a given fraction of frames f, accompanied with aggregation of semantic prediction of the points in these frames. After training, a voting scheme was carried out for all voxels that had received semantic information. The most common vote for every voxel was chosen as the voxel's type. Here, an exception was carried

out for all vegetation voxels where there were fewer than a specified number of semantic counts T_c. This was to limit the effect of too many voxels being missclassified as vegetation. After voting, an **expansion scheme** was carried out for every "vegetation voxel" to all 26 neighbouring voxels that shared a side or corner. This expansion scheme was iterated t-times.

Points were assigned voxels according to

$$\mathbf{v} = \left[l \cdot \left[\frac{r}{l} \right] \right]_n . \tag{36}$$

Here $[\cdot]$ denotes a rounding operator and $[\cdot]_n$ rounds to the n:th decimal. l is the side length, $\mathbf{r} = (x, y, z)$ is the coordinate of a point and $\mathbf{v}$ is the centre coordinate of the voxel that $\mathbf{r}$ belongs to. The size of n was chosen according to the side length so that n was greater than the number of significant digits in l.

Filtering was carried out by the same mechanism as before, but with the addition of voxels classified as vegetation filtering out all points mapped to the voxel regardless of the count. Thus, the filter heavily removed points in areas classified as vegetation by the neural network but worked as the standard Density Filter in all other spatial areas.

4.7 Test on Background Subtraction

The two different background subtraction models were tested in situations with wind, to see whether the improvements made any difference. Since the data was not annotated, this was limited to a select few frames and no advanced comparisons were made. More varied frames would have only made it more difficult to discern a difference since it would become difficult to pin point the cause of potential differences.

The *Improved Density Filter* and standard *Density Filter*, with side-length $l = 0.1m$ and $t = 1$ expansion, were tested on recordings of *Eslöv airport* and *Snowy Bus Station* with the RP-LiDAR. For *Eslöv airport*, both filters were trained on 840 consecutive frames and evaluated on 442 frames of two consecutive chunks of 250 and 192 frames each. For *Bus Station Snow*, the filters were trained on the initial 800 frames and evaluated on the final 95, for which wind appeared. Frames were chosen for mostly containing background. For every frame, points were filtered using $T = 10$ as threshold, and the remaining points were clustered using DBSCAN with $\epsilon = 0.4$ and minimum of 10 points in a neighbourhood for a point to be considered a core point. This was to emulate the behaviour of the filter in the actual pipeline.

Three metrics were calculated. The number of total points and clusters passing each filter were calculated. This was done to estimate both filters ability to

filter out vegetation in windy conditions.

In addition to this, visualisations of the *Improved Density Filter* was produced for further understanding of behaviour and intuition. Here, all voxels which would filter out points, where $T_c = 5$ and $T = 10$, were saved and plotted. This was done for *Eslöv airport* and a version of *Matteannexet*. In both cases, the filters were trained on 800 frames.

The recordings were produced with the standard settings described in Section 2.4.1.

5 Results

5.1 Ground Models

The following section presents results of the tests of the ground models described in Section 4.3.

Figures 12 and 13 show the F1-scores and accuracies for the hybrid models of different grid sizes on the two different sets of point clouds.

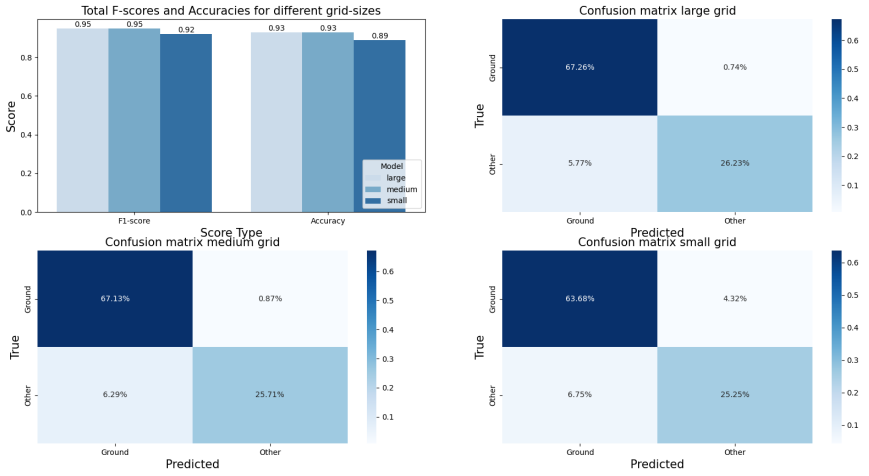


Figure 12: Results for Hybrid Model on the *Flat* point clouds for three different grid sizes. Total F-scores and Accuracies at top left and confusion matrices in other boxes for different grid sizes.

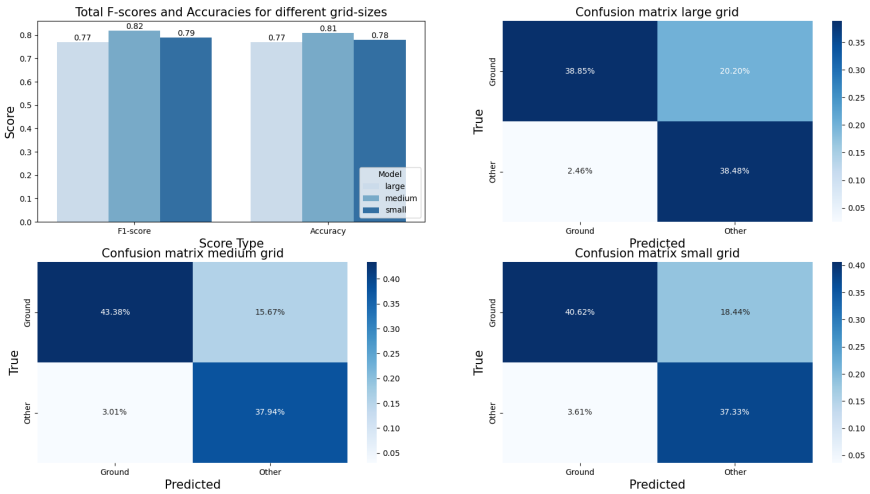


Figure 13: Results for Hybrid Model on the *Hilly* point clouds for three different grid sizes. Total F-scores and Accuracies at top left and confusion matrices in other boxes for different grid sizes.

Figures 14 and 15 show the F1-scores and accuracies for the RLWR-based models of different grid sizes on the two different sets of point clouds.

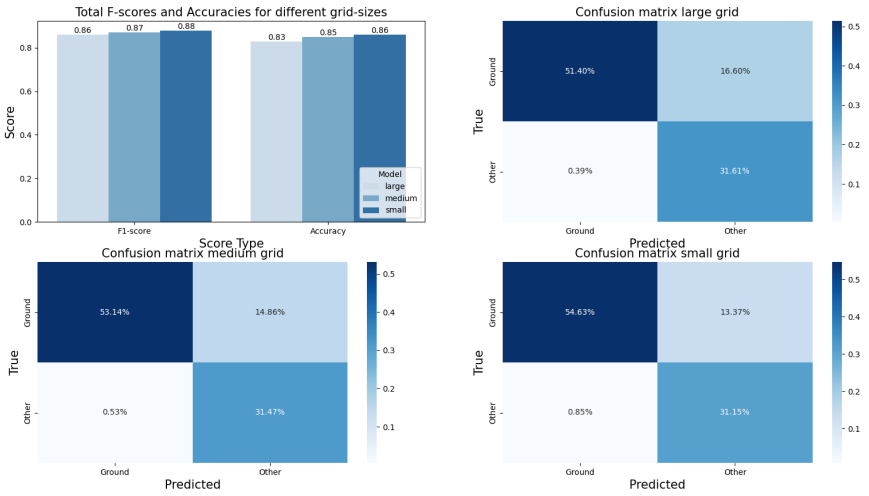


Figure 14: Results for RLWR-based model on the *Flat* point clouds for three different grid sizes. Total F-scores and Accuracies at top left and confusion matrices in other boxes for different grid sizes.

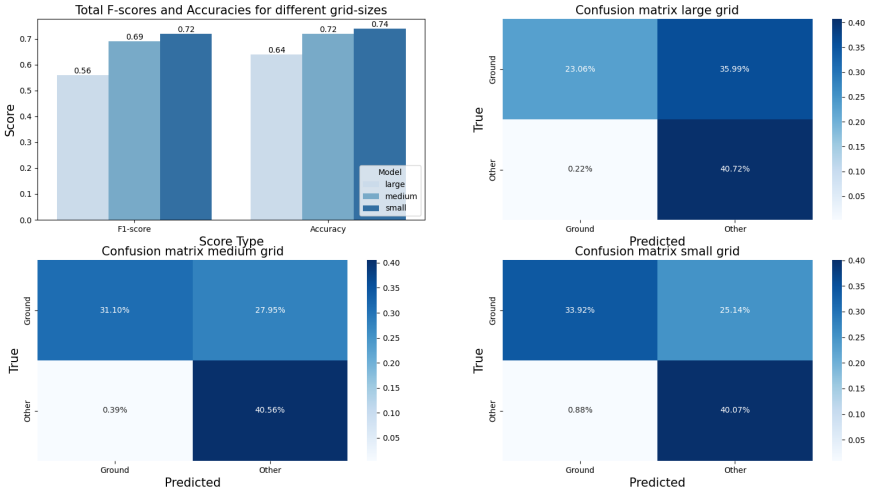


Figure 15: Results for RLWR-based model on the *Hilly* point clouds for three different grid sizes. Total F-scores and Accuracies at top left and confusion matrices in other boxes for different grid sizes.

Figures 16 and 17 show the F1-scores and accuracies for the plane model for different fitted with all points or with a subset of points on the two different sets of point clouds.

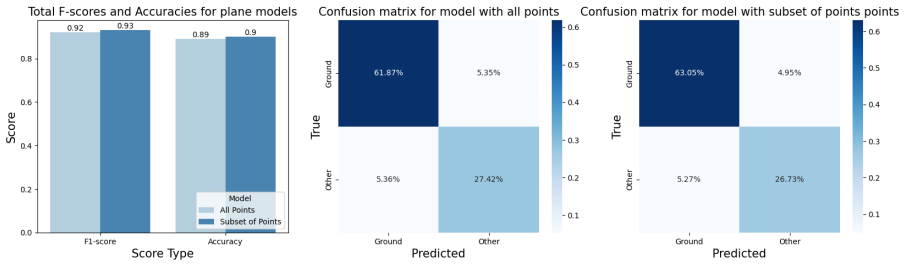


Figure 16: Results for plane-model on the *Flat* point clouds for either all points in point cloud or subset of point cloud. Total F-scores and Accuracies to left and confusion matrices in other boxes.

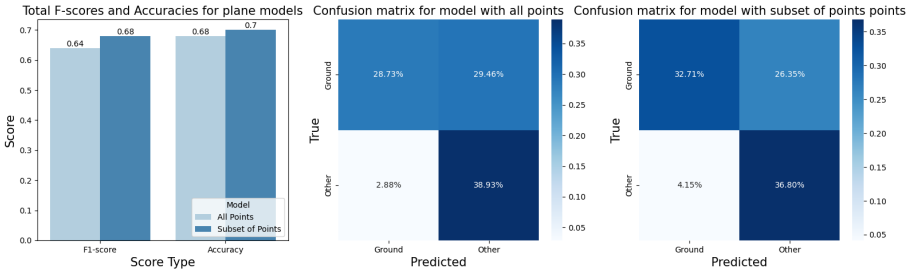
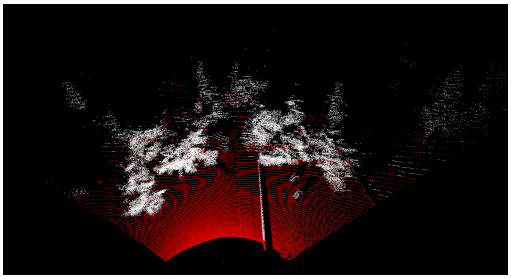
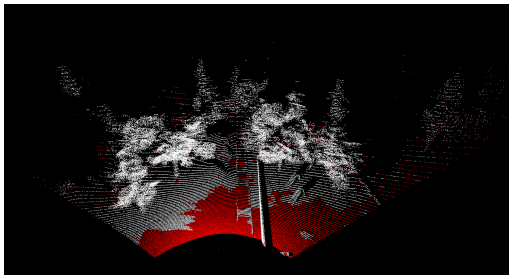


Figure 17: Results for plane-model on the *Hilly* point clouds for either all points in point cloud or subset of point cloud. Total F-scores and Accuracies to left and confusion matrices in other boxes.

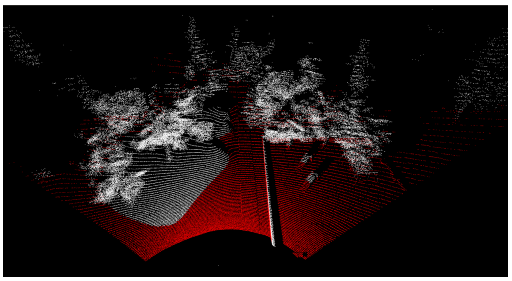
In Figures 18 and 19, examples of how the different type of models filter out ground points is displayed. The choice of grid-sizes was set according the models that produced best results for every model type. Note, these were separate runs from those that produced the results above. They used the same hyperparameters and are of the same point clouds. The models used were the Hybrid model with medium grid size, RLWR-based model using the small grid size and the Plane model using the subset of points in the point cloud.



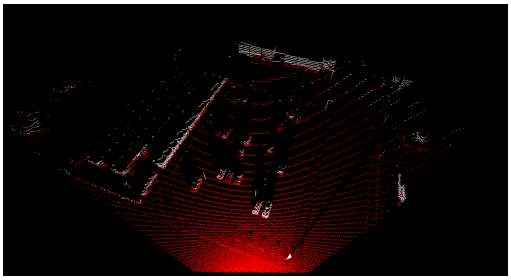
(a) Hybrid model on playground



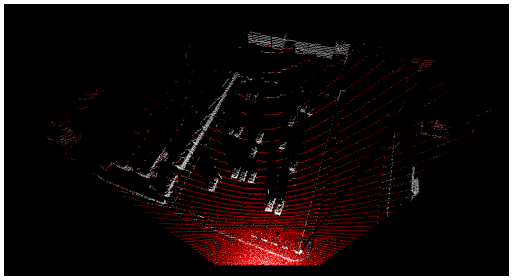
(b) RLWR model on playground



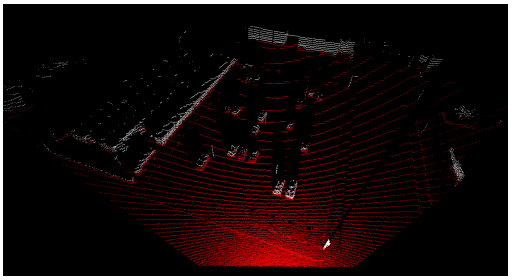
(c) Plane model on playground



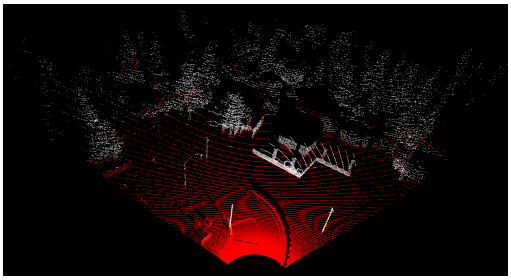
(d) Hybrid model on parking lot



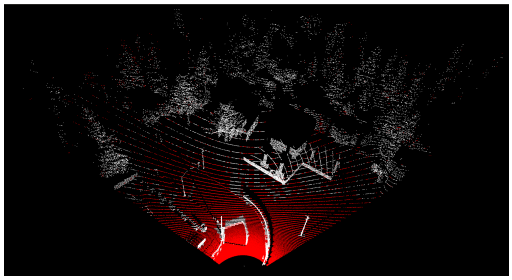
(e) RLWR model on parking lot



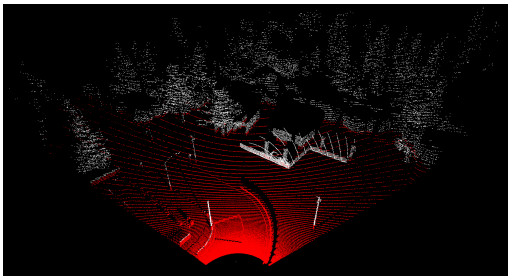
(f) Plane model on parking lot



(g) Hybrid model on petrol station.

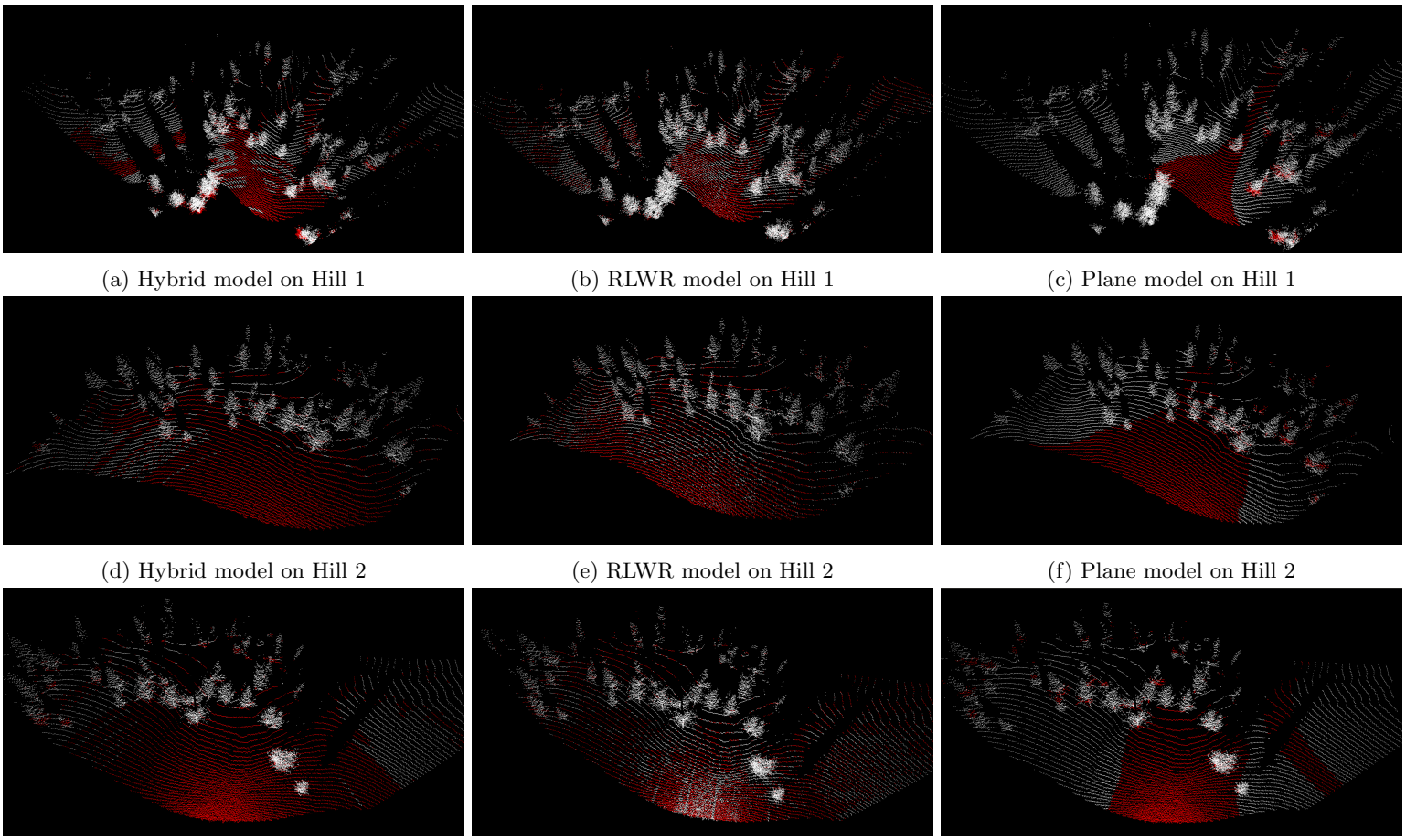


(h) RLWR model on petrol station



(i) Plane model on petrol station

Figure 18: Examples of ground model filtration on *Flat dataset* for the Hybrid model with medium grid, RLWR-based with small grid size and plane model fit on the same subset of points as the other two models. All hyperparamers are the same as previously. Points predicted as ground are red and all other points are white.



(a) Hybrid model on Hill 1 (b) RLWR model on Hill 1 (c) Plane model on Hill 1

(d) Hybrid model on Hill 2 (e) RLWR model on Hill 2 (f) Plane model on Hill 2

(g) Hybrid model on Hill 3 (h) RLWR model on Hill 3 (i) Plane model on Hill 3

Figure 19: Examples of ground model filtration on *Hilly dataset* for the Hybrid model with medium grid, RLWR-based with small grid size and plane model fit on the same subset of points as the other two models. All hyperparamers are the same as previously. Points predicted as ground are red and all other points are white.

The Hybrid model had the best performance of the three models in terms of accuracy and F_1-score. The best performing Hybrid model used the medium grid size ($dr = 0.6m, d\alpha = 6^\circ$) and achieved higher scores than the best RLWR-based model, which used the small grid size of ($dx = 0.25m, dy = 0.5m$). The Plane model performed very well on the *Flat dataset*, but did not achieve scores as high as the Hybrid model. The Plane model, however, achieved low scores on the *Hilly dataset*.

In Table 9, the worst specificity of every model type is shown on each dataset.

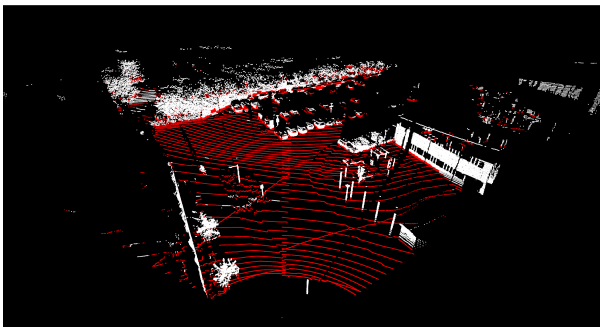
Table 9: Worst specificity of every model type on each dataset.

Model	Specificity *Flat dataset*	Specificity *Hilly dataset*
Worst Hybrid model	78.91%	91.18%
Worst RLWR model	97.34%	97.85%
Worst Plane model	83.53%	89.89%

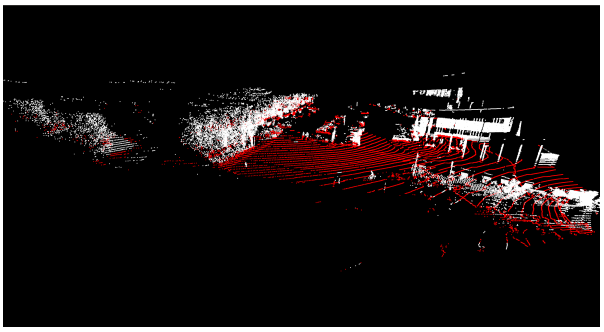
The RLWR-based model generally had very high specificity compared to the others. Also, specificity tended to be higher on the more complicated *Hilly dataset*.

5.1.1 Hybrid Model on RP-LiDAR point clouds

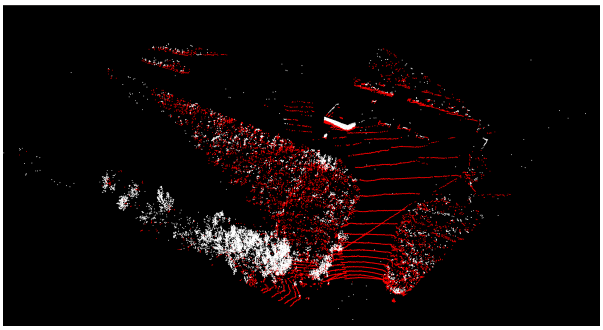
In Figure 20 below are predictions of ground points according to the description in Section 4.4. The Hybrid model with medium grid size was used together with the hyperparameters in Table 5 with the exception that the threshold was decreased to $T = 0.5m$. This means that the filtered red points in the images are within 0.5 meters of the predicted ground. The other parameters were $dr = 0.6m$, $d\alpha = 6°$, $f = 0.1$, $t = 5$, $\beta_{max} = 10$ and $\theta = (0.1935, 0.2415, 0.0396)$.



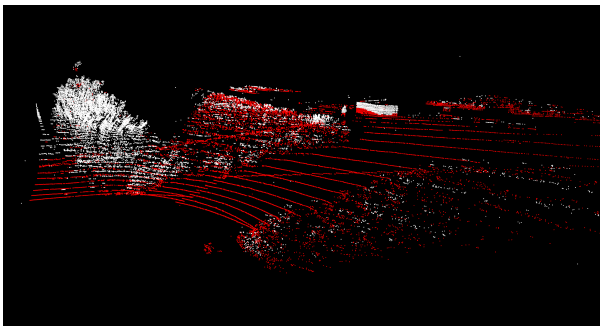
(a) Emdala seen from top view



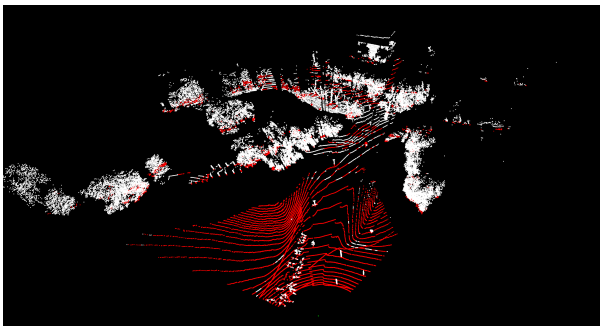
(b) Emdala seen from the side



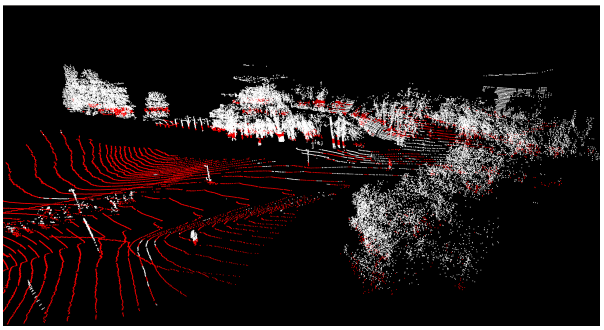
(c) Eslöv airport seen from top view



(d) Eslöv airport seen from the side



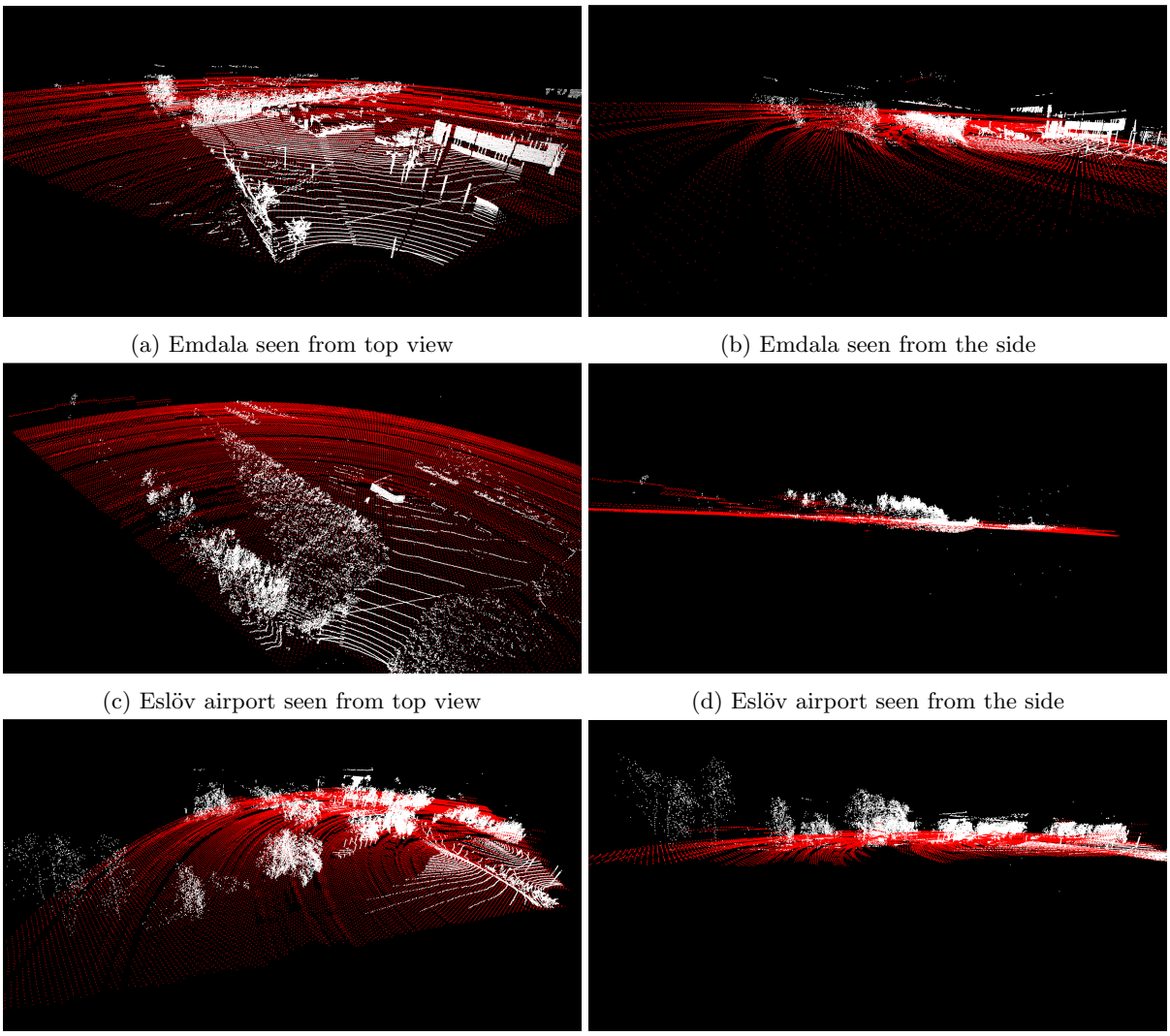
(e) Matteannexet seen from top view



(f) Matteannexet seen from the side

Figure 20: Examples of ground model filtration with the Hybrid model on three different point clouds recorded by the RP-LiDAR. Points classified as ground are red and remaining points are white.

In Figure 21, a grid of points representing the predicted height of the ground model (in red) are plotted alongside the point cloud for which it was fit. These are viewed in two directions for every point cloud.



(a) Emdala seen from top view

(b) Emdala seen from the side

(c) Eslöv airport seen from top view

(d) Eslöv airport seen from the side

(e) Matteannexet seen from top view

(f) Matteannexet seen from the side

Figure 21: Examples of ground model points for the Hybrid model on three different point clouds recorded by the RP-LiDAR. Points classified as ground are red and remaining points are white.

5.2 RandLa-NET

5.2.1 Training

In Figures 22, 23, 24, 25, and 26 shows training and validation accuracy along with loss and confusion matrices for variations of RandLA-NET specified in Table 8. Table 10 shows Mean-IoU and Accuracy for these models on the

test set. In Figure 27 the same is shown for the normal model trained on the Semantic-KITTI dataset.

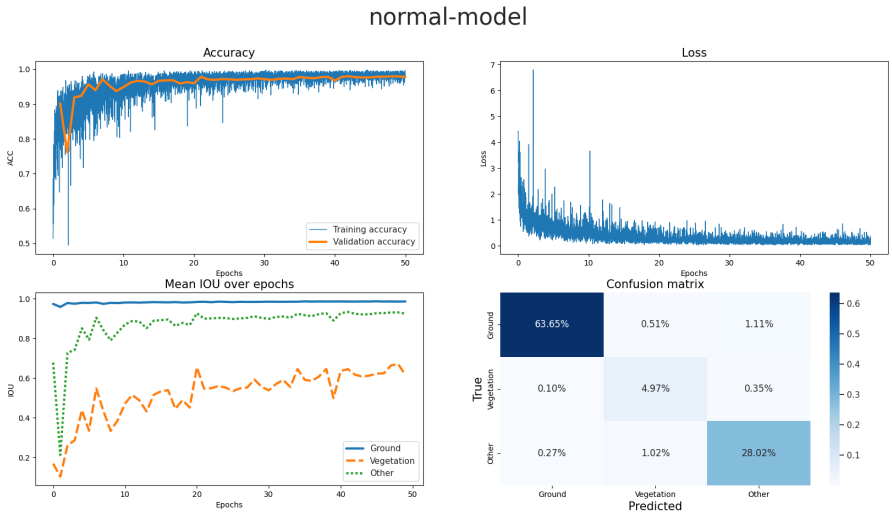


Figure 22: Training/validation accuracy and loss during training. Mean IOU score over different classes on validation set during training and confusion matrix of the normal model for test set.

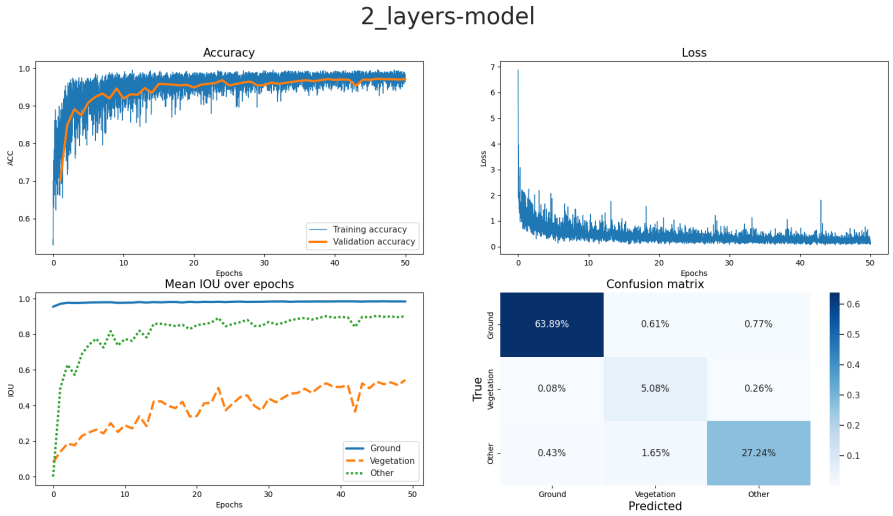


Figure 23: Training/validation accuracy and loss during training. Mean IOU score over different classes on validation set during training and confusion matrix of the 2 layer model for test set.

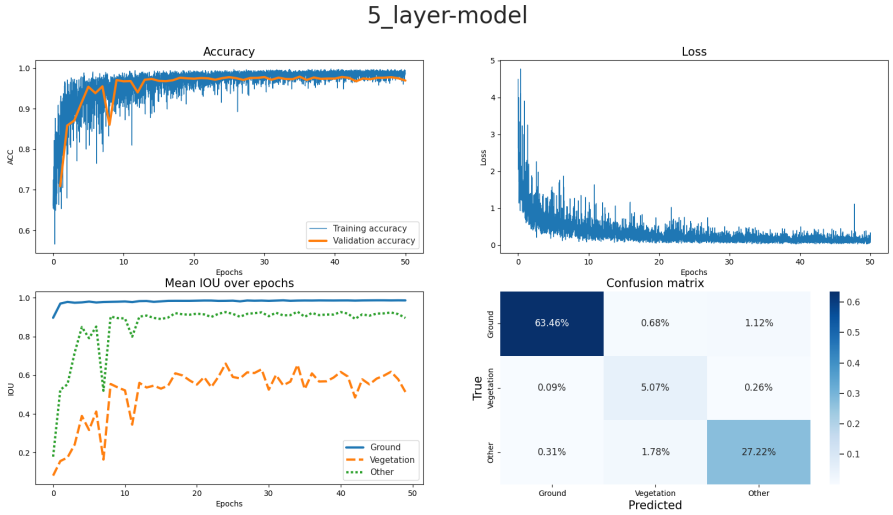


Figure 24: Training/validation accuracy and loss during training. Mean IOU score over different classes on validation set during training and confusion matrix of the 5 layer model for test set.

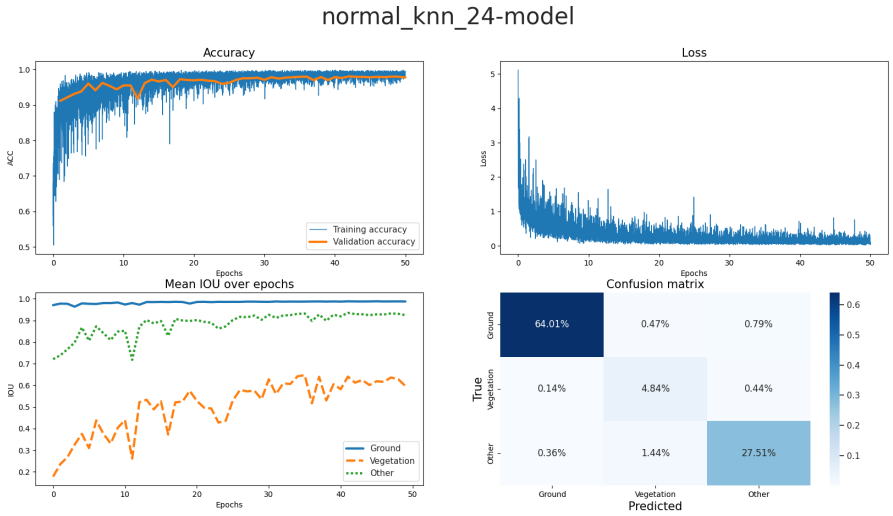


Figure 25: Training/validation accuracy and loss during training. Mean IOU score over different classes on validation set during training and confusion matrix of the KNN 24 model for test set.

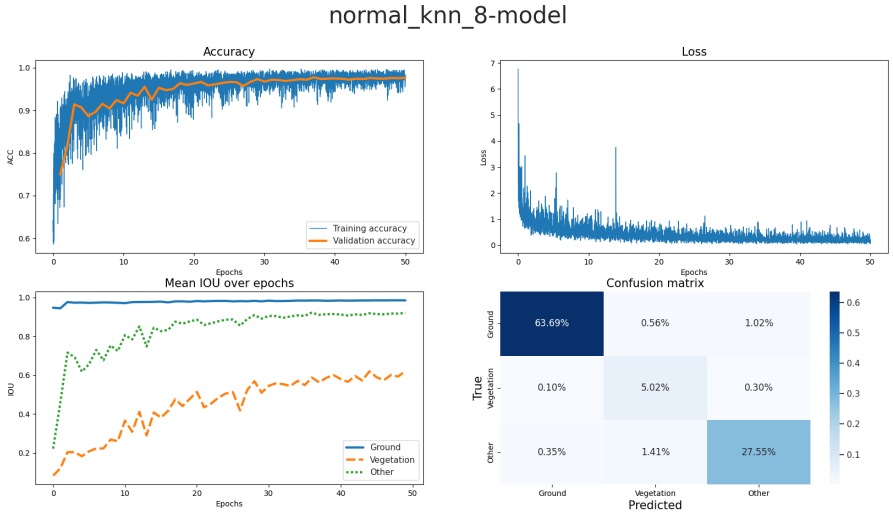


Figure 26: Training/validation accuracy and loss during training. Mean IOU score over different classes on validation set during training and confusion matrix of the KNN 8 model for test set.

Table 10: Mean IoU and accuracy over test set for different models.

	IoU [Ground vegetation other] [%]	mean IoU [%]	Accuracy [%]
Normal-model	[97 69 91]	85	96
Normal-model-25K	96 45 84	55	93
2-layer model	97 68 90	85	96
5-layer model	97 64 89	83	96
KNN-24 model	97 66 90	84	96
KNN-8 model	97 68 90	85	96

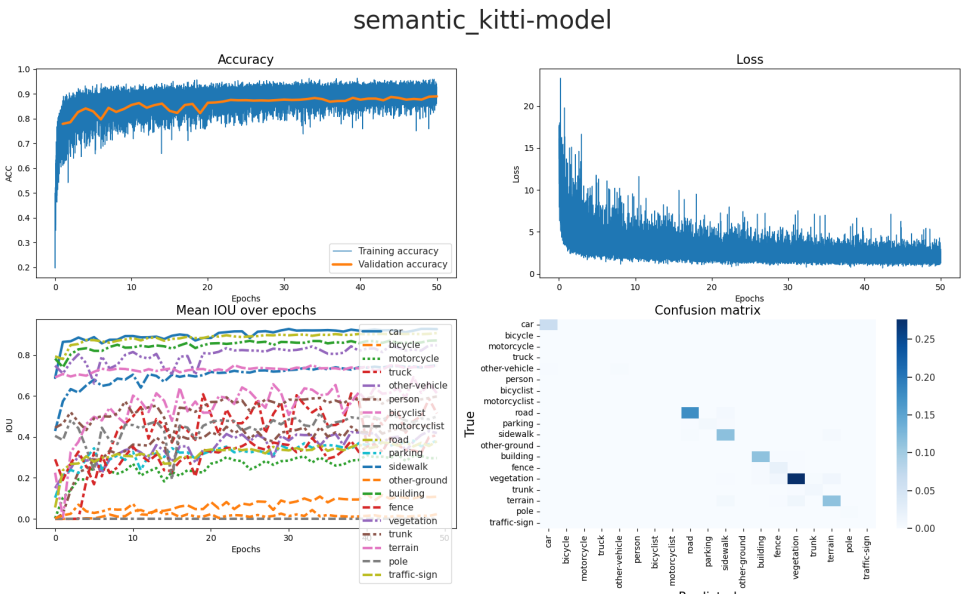
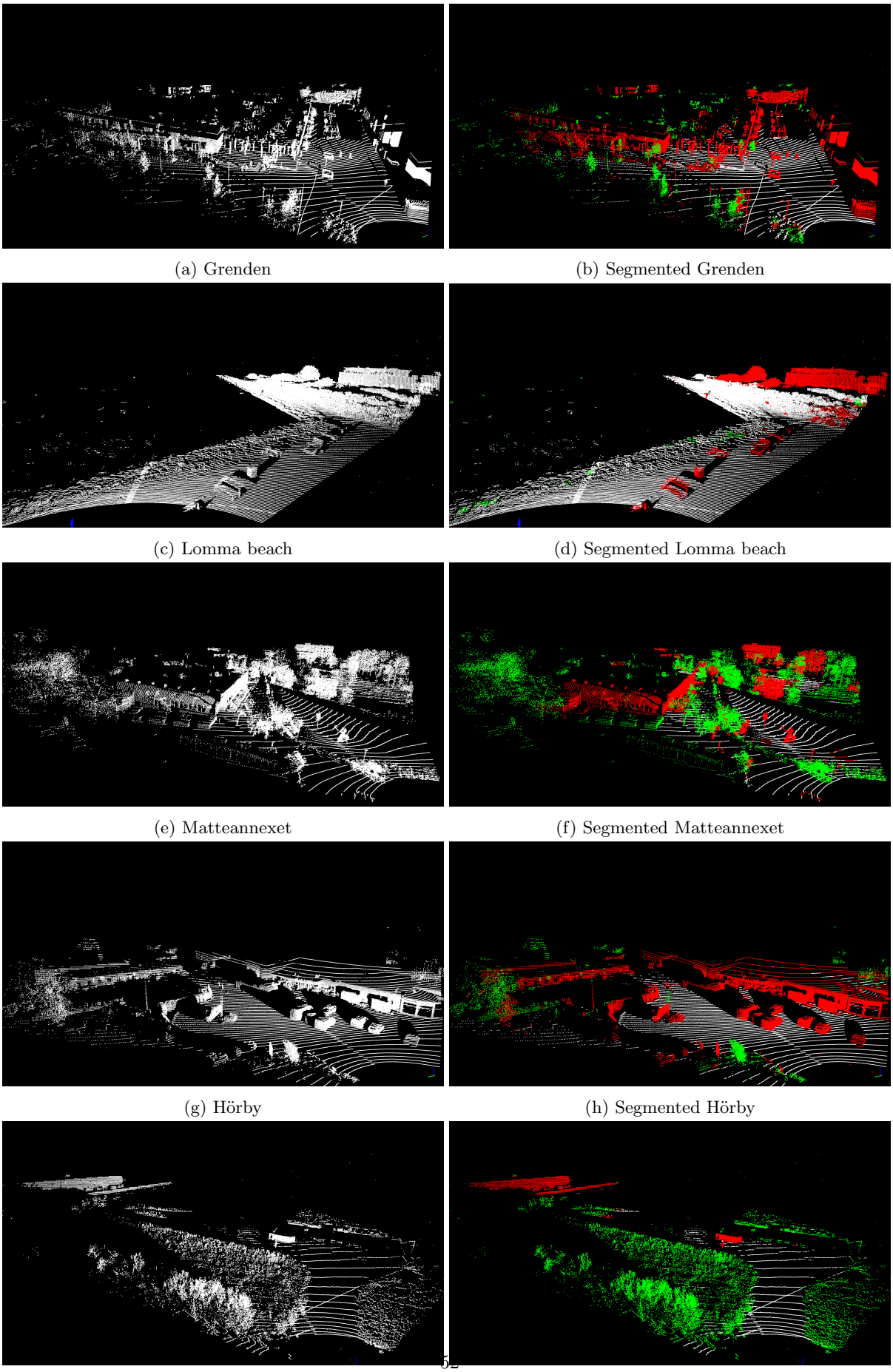


Figure 27: Training/validation accuracy and loss during training. Mean IOU score over different classes on validation set during training and confusion matrix for Semantic-KITTI model on Semantic KITTI dataset.

In Table 11 inference time along with relevant information of size for different point clouds from different recordings are shown. In Figure 28, the results of semantic segmentation on these point clouds is shown and in Figure 29 the same is done on scenes from CARLA with the *Graphic LiDAR*.

Table 11: Inference time for the normal-model on different RP-LiDAR datasets.

	Total time [s]	Mean batch time [s]	Batch size	Av pc size	Num points
Grenden	8.89	0.488	15	56402	30000
Lomma beach	9.64	0.528	15	83742	30000
Matteannexet	9.14	0.503	15	67919	30000
Lomma beach	7.9988	0.4407	15	83742	30000
Hörbylantmän	9.279	0.4931	15	48738	30000
Eslövflygplats	6.02927	0.5155	15	29956	25000



(a) Grenden

(b) Segmented Grenden

(c) Lomma beach

(d) Segmented Lomma beach

(e) Matteannexet

(f) Segmented Matteannexet

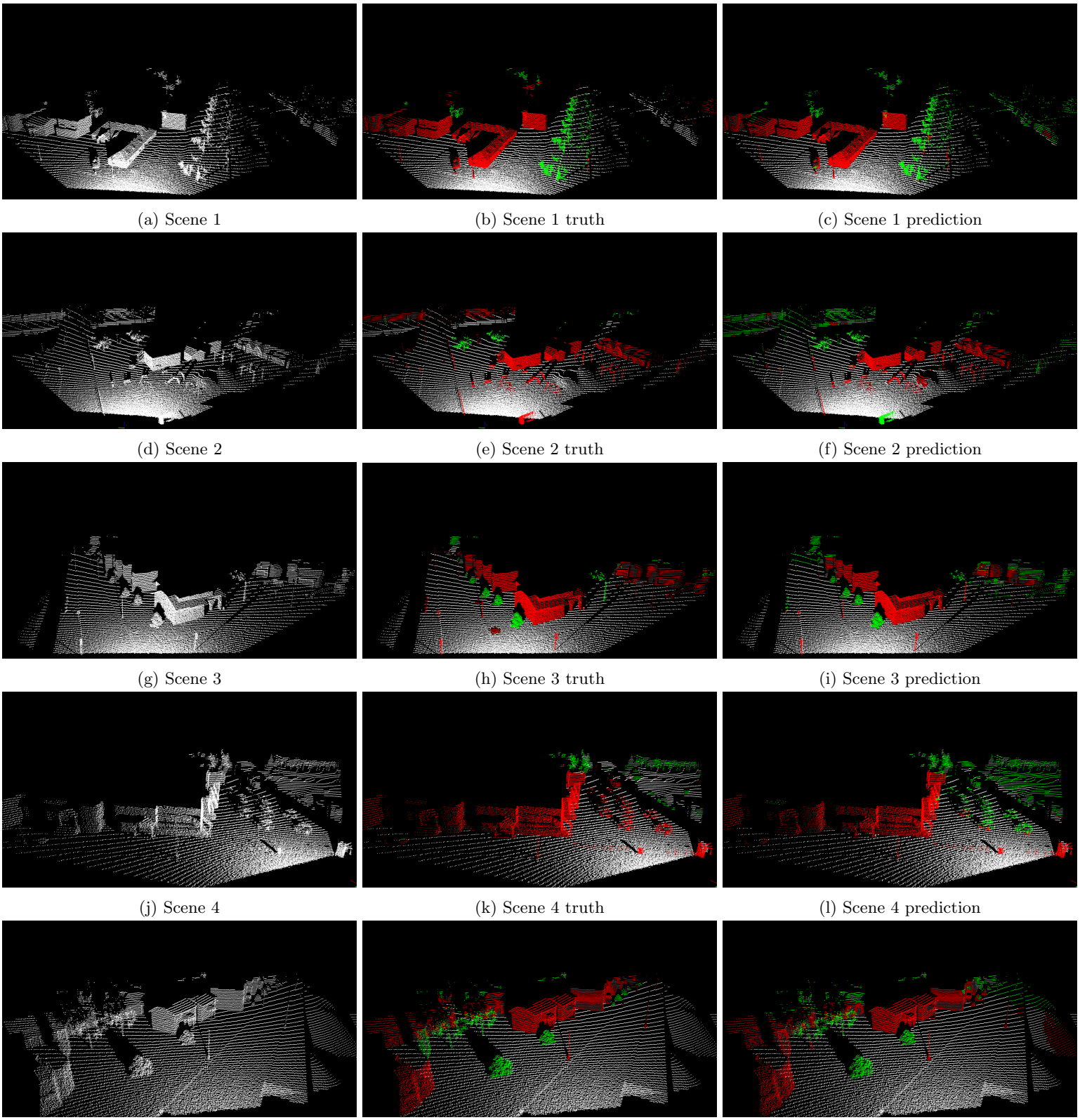
(g) Hörby

(h) Segmented Hörby

(i) Eslöv airport

(j) Segmented Eslöv airport

Figure 28: Results of the semantic segmentation on recordings by the RP-LiDAR. To the left are the original point clouds and to the right are the predictions. White points are classified as *ground*, green points are classified as *vegetation* and red points are classified as *other*.



(a) Scene 1 (b) Scene 1 truth (c) Scene 1 prediction

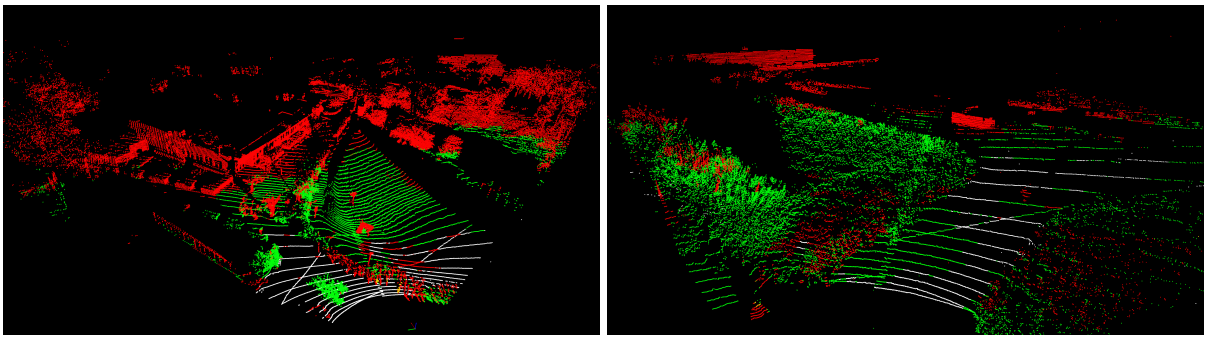
(d) Scene 2 (e) Scene 2 truth (f) Scene 2 prediction

(g) Scene 3 (h) Scene 3 truth (i) Scene 3 prediction

(j) Scene 4 (k) Scene 4 truth (l) Scene 4 prediction

(m) Scene 5 (n) Scene 5 truth (o) Scene 5 prediction

Figure 29: Comparing the semantic segmentation versus ground truth on frames from the CARLA test dataset.



(a) Matteannexet (b) Eslöv airport

Figure 30: Model trained from semantic-kitti data predicting on frames recorded from RP-LiDAR.

5.3 Background Filter

The number of points passing the filter, accumulated clusters and number of points in these clusters can be seen for *Eslöv airport* in Table 12 and for *Snowy Bus Station* in Table 13.

Table 12: Number of clusters, accumulated Point count and accumulated Point count within clusters for density filter and improved density filter for *Eslöv airport*.

Model:	Density Filter	Improved Density Filter
Number of clusters	2406	345
Accumulated Point count	629230	110436
Accumulated Point count (clusters)	70906	17080

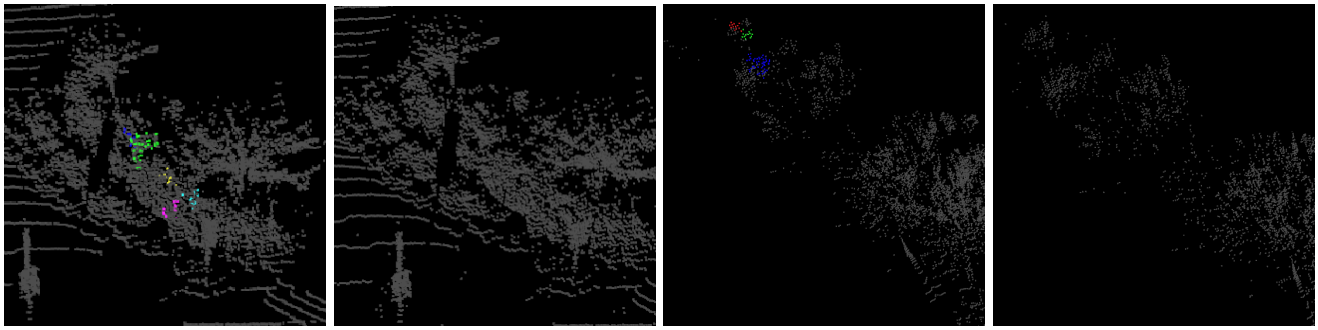
Table 13: Number of clusters, accumulated Point count and accumulated Point count within clusters for density filter and improved density filter for *Snowy Bus Station*.

Model:	Density Filter	Improved Density Filter
Number of clusters	390	227
Accumulated Point count	158287	88880
Accumulated Point count (clusters)	5964	2946

Of the registered clusters in *Eslöv airport*, 194 of them were of people moving throughout the scene. None of the filters filtered out a single instance of a person in a frame. Thus the count of clusters from wind in vegetation was 151 for the *Improved Density Filter* and 2212 for the standard *Density Filter*. In *Snowy Bus Station* a lot of clusters originated from a part of the ground that had been

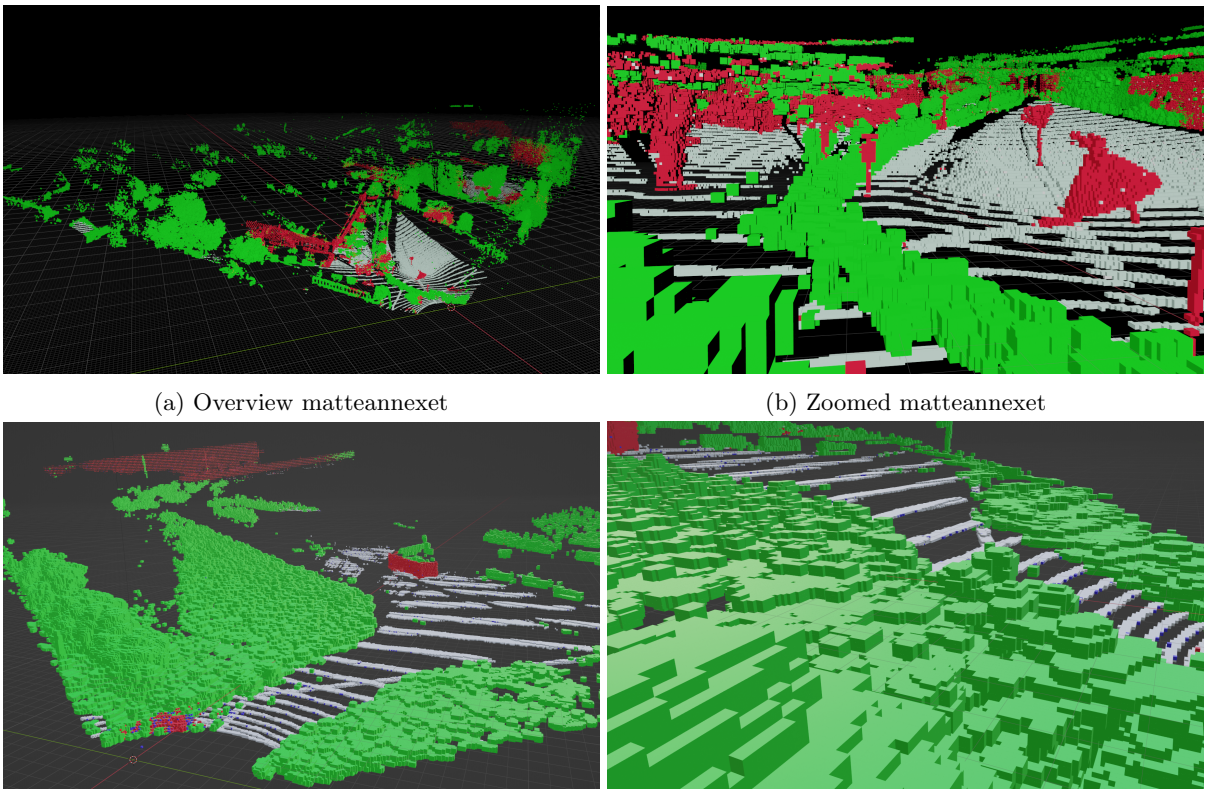
occluded during training due to snow on the LiDAR-sensor. From this, 130 clusters were added. Subtracting this gives 97 for the *Improved Density Filter* and 260 for the standard *Density Filter*. In addition to this, there were also many buses and cars that were clustered. In fact, an overwhelming majority of cluster registrations in the *Improved Density Filter* were from these. These were passed through both filters to the same extent. This means that the number of clusters in vegetation were much lower and by the same amount.

In Figure 31 examples can be seen of the same frame for each filter. The coloured points were the clustered points and the colour represents individual clusters. In Figure 32, visualisations of the *Improved Density Filter* can be seen on *Eslöv airport* and *Matteanexet* for $T_c = 5$ and $T = 10$. Voxels are coloured according to their voted category and the voxels shown are voxels.



(a) Density filter (b) Improved density filter (c) Density filter (d) Improved density filter

Figure 31: Comparison of density filter and our improved density filter over vegetation for a recording on *Snowy Bus Station*



(a) Overview matteannexet

(b) Zoomed matteannexet

(c) Overview eslöv airport

(d) Zoomed eslöv airport

Figure 32: Improved density filter voxels visualized. Green voxels: vegetation, white voxels: ground and red voxels: other.

6 Discussion

6.1 Ground Models

In terms of accuracy and F_1-scores, it was quite expected that the Plane model performed well on the *Flat dataset* but not well on the *Hilly dataset*. Still, this shows the strength of simpler models. The RLWR-based model, which was much more complicated and computationally heavy, did not manage to match the *Plane model*'s scores on the *Flat dataset*.

The datasets were somewhat skewed. In both datasets, there were more ground points than non-ground, however, ground points never made up more than 2/3 of all points. Also, due to the *Graphic LiDAR* produced "Parking lot" point cloud making up fewer points than the others, this scene was weighted less in the final scoring on the *Flat terrain* data. This effect makes F_1-scores increase somewhat and it is good to keep this in mind when comparing models. Despite this, a combination of accuracy and F_1-scores were strong factors in reaching a conclusion regarding a preferred model.

Data from two different types of simulated LiDARs was used to study the Ground Models in addition to real recorded data from the RP-LiDAR. The reason for this was to increase understanding of the performance of these models in as many settings and domains as possible.

6.1.1 Performance of Plane model

The limitations of the Plane model are also apparent in the confusion matrix in Figure 17, where a lot of ground is classified as other. For the best model, *True Positives* account for $TP = 32.71\%$ and *False Negatives* account for 26.35% which is very close. This means that it manages to correctly predict only 55.38% (recall) of ground points. This is due to the inflexibility of the model. Examples of this can be seen in Figures 19c, 19f and 19i where clear continuous parts of the ground are correctly classified.

6.1.2 Performance of RLWR-based model

The RLWR-based model produced quite underwhelming results. Compared to accuracies between 97.45% and 97.84% in the original article [9], the results in this thesis do not come close. In Figures 18b, 18e, 18h, 19b, 19e and 19h, it can be seen that the method is sporadic in classifying points. This is due to the way the method has to classify ground points in both x and y directions and then only classifies a point as ground if there is consensus, i.e. it is classified in both. Compare this to the other methods, were predictions of ground heights are made on continuous areas, such as a plane in the Plane model, and bins in the Hybrid model. The fact that the point clouds studied in this thesis are a lot less dense than those studied by the authors of [9], is likely a strong factor

behind the decreased performance. This means that height variations, caused by objects not belonging to ground, are a lot more difficult to discern. This is analogous to a low sample rate. A way of making every slice more dense, is making them broader, but here is a trade-off. Making slices broader decreases their "resolution". Points that are a slice-width from each other can be projected as beside each other. This makes it more difficult for the RLWR-iteration to discern the actual ground level if there is to much variation in a slice's depth.

In [10], the authors achieve accuracies between 73.18% and 95.23% on scenes from the KITTI dataset for their RLWR-based model. Since these are different environments, no definitive conclusions can be drawn, but since the KITTI dataset is produced by a LiDAR (Velodyne) with point densities that are roughly the same, this might suggest that the model in this thesis could have been implemented in an inferior way. However, it should be clearly pointed out, that the authors made a few modifications to the RLWR-implementation for their experiments. First, the x and y slices were exchanged for a polar grid map, using segments, circles and bins, the same as for the Hybrid model. The authors of [10] claim that this representation adapts to the point cloud's distribution better. Also, due to the size of the point clouds in the KITTI dataset, only the lowest points of every bin are used and points are filtered by the average height of the bin and a threshold parameter T_g. The discrepancy in results from the polar grid map is difficult discern. This likely makes point counts between slices in the same direction vary less. However, even though the authors claim that the choice of only picking lowest values in bins was motivated by time-performance reasons, it likely also had an impact on classification performance and behaviour. The sporadic and conservative nature of point classification is likely caused by the consensus method of filtering points. Changing to a threshold based method over a continuous area, changes this dynamic and causes all points within a threshold in the area to receive the same classification. This will cause more non-ground points to be miss-classified but likely more ground points to be correctly classified. For further discussion on sources of discrepancies, see the appendix, Section D.2.

There seems to be a lot of possible improvements that can be made for the RLWR-based model, but these would likely not improve the method beyond the Hybrid model. For one, the results in the article that proposed the Hybrid model [10], consistently showed their model outperforming their RLWR-based model. Secondly, RLWR-regression fits points in a scatterplot and does not seem to lend itself to height prediction in areas of a given grid without points. When searching for better models, potential improvements to the Hybrid model or newer models are suggested instead of this model.

6.1.3 Performance of Hybrid model

By measures of accuracy and F_1-score, the Hybrid model achieved the best results. This was always the case unless the ground was completely flat, as seen in

Figure 18i, where the Plane model performed the best. It was more adaptable to both ground undulation occlusion. There are however, very clear limits to how much it could handle. Even though the Hybrid model achieved best scores on the *hilly dataset* it is quite evident from Figures 19a, 19d and 19g that the model was quite bad at adapting to ground that varies at that scale. In Figures 18a and 18g it is possible to see that the model also struggled when the height of the ground had abrupt and steep variation. In the *playground* point cloud, there is such a hill just to the left of the central path and in the *petrol station* point cloud the same can be seen behind the petrol station. In both these cases, the problem did not arise from occlusion.

On the "real" points clouds in Figure 20, generated by the RP-LiDAR, the Hybrid model showed good performance on *Emdala* and *Eslöv airport*. There were height variations in these point clouds, but they were more gentle than those seen in the simulated clouds. In *Emdala* it is possible to see that the model adapted to the slightly twisted shape of the ground undulation. The little hill in the far back (seen more clearly in Figure 20b) was missed by the model however. Here, the model was drawn up to the height of the bushes as evident in Figure 21a and 21b. In *Eslöv airport*, the ditch to the left, close to the LiDAR was captured in the model and the lowest point of the smaller bushes were also included. It is worth pointing out here that, despite not seeing the ground, the model made an accurate prediction of what the height actually was. All of this is confirmed by plots of the actual model in Figure 21. From Figures 20e and 20f it is clear that, once again, the Hybrid model had problems in handling too much ground undulation and occlusion. The model quite accurately managed to capture the change of height of the hills close to the LiDAR. Further away however, where these hills occluded large parts of the point cloud, especially around the completely occluded pond with steep edges surrounding it, the model was barely able to find any ground. Looking at the actual model in Figures 21e and 21f this is clear. The model was very modest in following any steep inclines. This was likely due to a combination of the gradient filter and the GPR. The gradient filter in the RLWR-step changes predicted height for points along segments with slopes larger than 10°. The GPR has a noise parameter in the kernel σ_N^2 that avoids outliers and since it is fit only on seed points along a circle, the angular resolution might have been too large to treat these sudden changes in height as non-outliers.

6.1.4 Further analysis of Hybrid Model

From all of this, a few a clear patterns emerge. The first is that the model could not handle changes in height that were too sudden and dramatic as in the *Hilly dataset*, the sharp hills in *Playground* and *Petrol Station* and the steep edges in *Matteannexet*. This was likely caused by a combination of the gradient filter and the smoothing of the GPR using the exponential kernel.

The second pattern is that the model seemed to have been quite negatively

impacted by sparsity of clusters and large empty gaps in a given circle. These issues were usually caused by occlusion. The resulting incorrect fit was likely driven by two factors. The first was that when the mean value of a circle became based on few points with large gaps and since these were often affected by occlusion, they would usually not belong to the ground. Thus, bias was introduced and the mean height for these circles became biased to become higher than the mean height of the actual ground for that circle. The second factor, was that the GPR likely became more conservative with less inputs. Further discussion of this factor and examples of this in the point clouds can be seen in Section D.3, in the appendix. Generally, where the model was smooth, there seemed to be an abundance of regularly spaced points and where the heights of the circles begin to vary in a non-smooth way, the point clouds started to become affected by occlusion.

It is difficult to avoid the problem of strong height variations unless the actual scene is changed. In a case where it is critical that the ground model is accurate, it might be motivated to actually change the environment. Obscuring objects such vegetation or man made objects, might be removed. If it is possible to convert the scene of interest into something like the front part of *Emdala*, the model would likely consistently perform well. Lastly, there are a few ways in which the model itself might be improved to prevents issues from heavy occlusion. One source of error seems to have arisen from calculating the mean height of circles. This was done by calculating the mean of the seeds for a given circle and did not involve comparing them to that of any neighbouring circles. Although, the seeds are affected by neighbours by the RLWR-iteration for the segments, there was no sanity check on whether there was a large jump between circles. This could be done by assuming that the mean height of circles only can change with a given slope angle. Doing this, sudden jumps in mean height may be suppressed and also, the more systematical increase in height at further distances, caused by only seeing the top of objects, might be reduced. Another potential improvement would be to allow increased sizes of bins at further distances. Due to lower point densities, this could potentially increase the probability of including actual ground points, decreasing height bias and making the GPR fit less sparse.

Another interesting aspect is what data to use when producing the ground model. In these experiments, only single frames were used. In an actual setting, together with other background models that require multiple frames to be trained, the model might take advantage of aggregated frames to reduce sparsity. One way of doing this might be by evaluating the model on the background filter and perhaps voxel-centres of appropriate kinds from the improved density filter. Another possible solution would be to hand-scan the surroundings beforehand and constructing the ground model from this data. This would also reduce the impact of occlusions.

As mentioned before, the Hybrid model had the worst specificity on the *flat dataset*, meaning that it miss-classified the greatest amount of non-ground points

as ground. If a model is sought where this is very important to avoid, another model might be of more interest. The results of ground model filtering in this thesis seems to suggest that the consensus approach to filtering is better at avoiding this than categorising all points within a threshold of a continuous model. This might also just be the case of the RLWR-model being overly cautious.

6.1.5 Conclusions

The great performance of the Plane model on simple flat terrain, clearly demonstrates the strength of simple models. However, once the terrain becomes more complicated, with ground undulation and occluded areas, more complicated models are desired. Of these, the Hybrid model clearly outperformed the RLWR-based model in terms of accuracy and F_1 score and shows greater promise in adapting to occlusion. Still, even this model breaks down with too much height variation and occlusion. This means that it is necessary to be mindfull of placement of the LiDAR and the height variations of a scene when considering deployment of the model.

6.2 RandLA-NET

Looking at the different RandLA-NET models in Figures 22, 23, 24, 25 and 26 it can be seen that all models achieve the lowest IoU-score for the vegetation class. This is likely due to the *vegetation* class having the most complex spatial structure. This complexity partly comes from the fact that vegetation has many different forms, from a single bush to a field of crops to a tree. To improve future predictions it could be wise to provide a dataset with separate labels for small shrubbery and trees. Since CARLA did not allow for such differentiation this could not be tested in this thesis.

The IoU score for the ground class was already after one epoch over 90 % for all models. This means that all models learnt to detect this incredibly fast and is likely due to ground points generally sharing a simple pattern: evenly distanced and following straight lines. This seems to have been easy for the network to learn.

In all models, the training appeared to converge after approximately 25 epochs as neither the training accuracy or loss changed considerably. This indicates that training after 25 epochs on the data from CARLA could lead to overfitting. One can actually see a small a hint of overfitting as the validation accuracy slightly falls short of the training accuracy during the last epochs. However, as the overfitting is very small it does not seem to have been an issue. The low amount of overfitting could imply that the training set is well varied with many different environments.

It is somewhat strange that most models share similar scores on their confusion matrices even though they have large differences in the amount of layers, point spatial resolution and network parameters. A common strategy in machine learning is to rely on the simplest model (Occham's razor) as larger models tend to induce overfitting, require more memory and are slower. This implies that the larger models might be unnecessarily large for the CARLA point clouds and it might therefore be better to use the smaller models to save computer resources. In the instance of domain transfer to "real" point clouds, if models are overfit, they are more likely to achieve worse performance and thus one of the smaller models is recommended for use in such a pipeline.

The performance of RandLA-NET on RP-LiDAR data was good for many different recordings. From Figures 28b, 28d, 28f, 28h and 28j one can see that in many cases it correctly segmented most points of class *ground*. This agrees with the aforementioned reasoning that *ground* had a simpler spatial relationship. The class *other* was usually correctly predicted on buildings and poles. The class *vegetation* was generally predicted correct with some exceptions where it was confused with *other* as can be seen in Figure 28f, where a tree is confused with the class *other*. Although the performance was somewhat lacking in certain instances, it was generally quite good.

One aspect in which further improvements could be achieved is through using better and more varied data for training. In this thesis, data from six CARLA-environments were used. Despite being from different "maps", a lot of the environments were quite similar and most of them were mostly flat. More varied environments, with more ground undulation and more assets would help in increasing this variability. Being able to vary the scanning pattern to a trapezoidal pattern and more opportunities to vary semantic tags for assets would also help in making models that are more adaptable. These would likely be better at domain transfer and therefore also better at semantic segmentation of the RP-LiDAR data.

Looking at the predicted point clouds from the *Graphic LiDAR* plugin, in Figure 28, one can visually confirm that the point clouds were mostly correctly predicted concerning the three classes. However, it seems that there occured some miss-classifications on the edges, where the point cloud was especially sparse. This was especially the case at far distances from the LiDAR, but also on the sides. Here, the ground truth was often *other* or *ground* whereas points were often predicted as *vegetation*. The likely reason behind this is that these locations are sparse in such a way that point formations closely resemble that of vegetation.

It was not necessary to preprocess the point cloud using grid subsampling, however, it likely improved performance as it increases the receptive field for the points in the dense areas, see [28]. During testing, the skipped points were mapped to the voxel center point and therefore assigned the same prediction

value.

Looking at Table 10 most models had equal performance on the test set with the exception of the normal model with $N = 25K$ input points per inference pass, which achieved considerably worse performance on the vegetation class. This is reasonable, since smaller input clouds would provide a worse overview of the setting. Interestingly though, the KNN-24 model which also had 25K input points, achieved a much better IoU-score. Therefore the larger receptive field, provided by using more nearest neighbours, seemed to have counteracted the shorter point reach by having less input points.

Concerning results on Semantic KITTI, our maximum mean-IoU score, achieved by any model, was 51.4 %. In the original report the authors received a mean-IoU score on the dataset of 55.9 % [28]. The scores in the original report were however collected from *Semantic KITTI*'s official test dataset for which access was not available to us. Our score was computed from the validation set. The reason for reaching a lower mean-IoU score was likely due to a lack of computer resources. The authors used an Nvidia RTX2080Ti GPU with 12 GB of memory, whereas we used an Nvidia RTX2080 with 8 GB of memory. The difference in GPU memory proved to be a substantial problem during training as we often received GPU resource exhaustion when running on standard settings. To compensate for low GPU memory, parameters such as number of input points to the point cloud and batch size, had to be scaled down. A small input point cloud likely minimizes the information scope that the network has and likely worsens the performance. A small batch size would likely worsen the updates of the weights as the gradient updates contains less data samples and is therefore more noisy.

In Figure 30 we can see that it is not wise to use a model trained on the Semantic KITTI dataset for inference on the RP-LiDAR recordings. In Figure 30a and 30b *ground* and *vegetation* is mixed up in a lot of cases and many trees were labeled as *other*. The reason for this is likely due to large differences in point density, scanning pattern and recorded angles.

The prediction currently only utilizes three classes. More applications of the semantic segmentation would be enabled if the models were trained on more classes. Examples could be *car*, *ground*, *grass*, *tree*, *bushes*, *buildings*, *poles*, etc. An interesting improvement of the network could also be to introduce a time series module. This could be done by connecting points from previous point clouds to the local spatial encoder to make features of the relative point positions to the previous point clouds. This would likely improve prediction on dynamical objects that move or sway. However, this would also greatly enlarge the size of the network and would therefore require a very strong GPU. It would also be wise to train the model with data that has reflectivity as this would likely improve predictions.

As can be seen in Table 11, RandLA-NET is a very fast network. The total inference time including multiple inferences, voting and remapping of subsampled point clouds was approximately 0.6 seconds per point cloud. The true inference time was lower: 0.032 seconds per sub-point cloud with N points. This suggests that RandLA-NET is suitable for real time point cloud processing if there is a GPU available. The authors of RandLA-NET received an inference time of 0.04 seconds which was similar to our inference time.

6.2.1 Conclusion

RandLA-NET showed great promise of being both effective and precise in performing semantic segmentation on point clouds of the same type that it was trained on. On those generated by RP-LiDAR, the results were still good, although sometimes, visibly worse. Besides being accurate, it was also fast, showing promise for real time segmentation in deployment, although it required a GPU. To reduce the workload it is suggested that the amount of layers and KNN points are decreased as the network showed similar performance. Furthermore, it is recommended that more varied data is produced for training better models. Also, more classes are recommended as this would increase the possible applications of the network.

6.3 Background Filtering

The results strongly indicate that there were added benefits of including semantic information in the background model. However, it is essential to mention that the tests were only in scenes that contained wind. In a lot available recordings this was not the case, and here, the standard *Density filter* showed equal performance. This is reasonable since the additions of voxel expansion and strict filtering of vegetation voxels only adds utility in these cases. When the background is still, no additional point subtraction is made. In this case, this will actually only worsen the performance of the pipeline, since the background model will filter objects such as animals and people that are close to the vegetation. This effect is also present when there is wind, but the added utility of being able to remove more points generally outweighed these effects. With larger side-length however, this effect becomes more pronounced and thus might start to become an issue.

Another noteworthy aspect of the results was the impact the clustering stage had in filtering out stray and sporadic points. In the point cloud stream, a fair amount of noise, generally in the sky or at far distances appear as sporadic points. These points are difficult for the filter to handle due to shifting positions and low density nature. This shows why the subsequent clustering step is essential in the filtering procedure. DBSCAN is very effective in determining

whether points are stray or part of a cluster.

A strong limiting factor in comparing the models was the lack of annotated data. This meant that it was difficult to numerically compare the models. Thus, only certain selected frames were used, where visual and numerical comparisons could be made and where confounding effects could be avoided. One way of improving the comparison would have been to simulate scenes and thus have a ground truth to compare with. Another limiting factor was that the semantic segmentation was required to achieve a certain level of performance on a given environment for the filter to work.

From visual comparisons of the two models, the *Improved Density Filter* seemed to suppress dynamic background to a much greater extent, even with lower levels of wind. However, this would have been difficult to numerically compare in a lot of cases, where the majority of clusters arose from non-background dynamic objects. Discrepancies could have arisen from many different reasons. It is difficult to formalize this as a result, but it is worth mentioning.

Given all this, the *Improved Density Filter* is recommended in general cases when vegetation occurs. Depending on the use-case however, the number of expansions of the model might be decreased to $t = 0$ if higher value is placed on avoiding miss-classification of non-background as background.

6.3.1 Conclusion

It is difficult to draw any strong conclusions from the limited experiments on the different filters. However, in situations with strong wind it was shown that the *Improved Density Filter* made a big difference in suppressing dynamic background. In addition, the semantic background filter is only as good as the semantic information it is provided, and thus it is essential that a semantic segmentation model is able to handle the given environment.

7 Conclusion

It has been shown that the Hybrid ground model is able to accurately describe ground that is sufficiently flat and non-occluded. The plane model failed to capture ground undulation and the RLWR-based model was too conservative in predicting ground points. For both Hybrid and RLWR-based models, suggestions of improvement were made, however, the Hybrid model was deemed to show more promise. It has also been shown that it is possible to perform semantic segmentation on point clouds recorded by a LiDAR well by training a neural network on simulated point clouds. The segmentation is domain transferable as the results are sufficiently adequate even though the scanning patterns vary between simulated and recorded data. Changes to the architecture of the neural network did not seem to have a large effect. Finally it has been shown that

semantic information could be included in an expanded background filter to successfully suppress dynamic background objects. This experiment was however limited in scope and further study is encouraged for definitive conclusions regarding the model's performance. The background filter is also only as good as the semantic model and so improvement of this is essential for the background model to become fully reliable.

8 Future Work

There is a lot further work to be done. In the case of ground models, the potential improvements mentioned in Section 6.1.4 could be explored. Also, better ways of visualizing the actual model could be studied with focus on creating meshes or wireframes from the ground function $f(x, y)$. An interesting addition to the tests performed, would be to study distributions of distances between the model's prediction of ground to correctly and incorrectly classified points in histograms. In [11], the authors take a Deep Learning approach by developing a neural network that takes a point cloud as input and as output (1) produces a grid of the ground and (2) performs segmentation of ground points. Such a network would however require a lot of data. Thus, the success of this depends on whether the environments in CARLA are varied enough to achieve results that are good enough to translate to RP-LiDAR data.

Whereas all semantic segmentation models were evaluated on a test set of simulated data. It would be interesting to further explore how the different models would have handled domain transfer to the data produced by RP-LiDAR. As of now, this was only explored for the *Normal* model with $N = 30K$. Even though the model could handle many environments surprisingly well, there is still a lot of room for improvement. The semantic segmentation was concluded to very likely have been limited by the quality of the training data and thus, this is an area in which further work is strongly suggested. One way of improving the quality of the data would be to switch over to an Unreal Engine 5 based simulation environment and use the LiDAR in [33]. Trapezoidal scanning patterns of the training data could also greatly have benefited the domain transfer along with more varied terrain than CARLA's generally flat ones. There is also much more experimentation to be done in choice of classification categories and the impact of these. For example, being able to successfully distinguish between different types of vegetation could potentially synergize with the ground model if the ground model was trained on only points classified as "ground", "short bushes" and other types of objects types that are generally close to the ground. Another interesting comparison would be to test other models such as Voxnet and KpConv and see how well different models handle domain transfer.

A natural next step would be to study the performance of the background filter on simulated data. Doing so, a more data driven conclusion regarding its effectiveness could have been achieved. Also, it would be of interest expand another type of model to include semantic information, such as the *Maximum Distance Filter*. Such an implementation could be carried out by a similar method of accumulating semantic information for every angle represented by a matrix and saving the most common for every angle. Thereafter, depending on the background type, different thresholds could be applied as well as a similar expansion scheme.

In this thesis, potential uses of ground model and semantic information of the

background were discussed, but not explored to a greater extent. A natural further step would be to do this. Examples of this is to use centroid height as a feature in tracking and classification, to study impact of using a ground model as a filter for ground points and/or clusters outside heights of interest, to use semantic information to fill out parts of a point cloud for visualisation purposes etc.

References

[1] Berntsson J. Winberg W. "Pedestrian detection and tracking in 3D point cloud data on limited systems". Master Thesis in Computer Science. Lund University, 2021.

[2] Bernståhle R. Lind H. "Segmentation, Classification and Tracking of Objects in LiDAR Point Cloud Data Using Deep Learning". Master Thesis in Mathematical Sciences. Lund University, Jan. 2022.

[3] J. Wu, H. Xu, and J. Zheng. "Automatic background filtering and lane identification with roadside lidar data". In: *2017 IEEE 20th International Conference in Intelligent Transportation Systems (ITSC)* (2017), pp. 1–6.

[4] J. Wu et al. "Automatic Background Filtering Method for Roadside LiDAR Data". In: *Transportation Research Record* 2672 (2018), pp. 106–114.

[5] J. Zhao et al. "Detection and tracking of pedestrians and vehicles using roadside LiDAR sensors". In: *Transportation Research Part C: Emerging Technologies* 100 (2019), pp. 68–87.

[6] Wen Xiao et al. "SIMULTANEOUS DETECTION AND TRACKING OF PEDESTRIAN FROM PANORAMIC LASER SCANNING DATA". In: *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences* III-3 (June 2016), pp. 295–302. DOI: 10.5194/isprs-annals-III-3-295-2016.

[7] Y. Song H. Zhang and Y. Liu J. Liu H. Zhang X. Song. "Background Filtering and Object Detection With a Stationary LiDAR Using a Layer-Based Method". In: *IEEE Access* 8 (2020), pp. 184426–184436.

[8] Jianying Zheng et al. "Background Noise Filtering and Clustering With 3D LiDAR Deployed in Roadside of Urban Environments". In: *IEEE Sensors Journal* 21.18 (2021), pp. 20629–20639. DOI: 10.1109/JSEN.2021.3098458.

[9] D. Belton A. Nurunnabi G. West. "Robust Locally Weighted Regression Techniques for Ground Surface Points Filtering in Mobile Laser Scanning Three Dimensional Point Cloud Data". In: *IEEE Transactions on Geoscience and Remote Sensing* 54.4 (2016), pp. 2181–2193.

[10] Kaiqi Liu et al. "Ground Surface Filtering of 3D Point Clouds Based on Hybrid Regression Technique". In: *IEEE Access* 7 (2019), pp. 23270–23284. DOI: 10.1109/ACCESS.2019.2899674.

[11] Anshul Paigwar et al. “GndNet: Fast Ground Plane Estimation and Point Cloud Segmentation for Autonomous Vehicles”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 2150–2156. DOI: 10.1109/IROS45743.2020.9340979.

[12] Jie Cheng, Dong He, and Changhee Lee. “A simple ground segmentation method for LiDAR 3D point clouds”. In: *2020 2nd International Conference on Advances in Computer Technology, Information Science and Communications (CTISC)*. 2020, pp. 171–175. DOI: 10.1109/CTISC49998.2020.00034.

[13] Charles R Qi et al. “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space”. In: *arXiv preprint arXiv:1706.02413* (2017).

[14] Daniel Maturana and Sebastian Scherer. “VoxNet: A 3D Convolutional Neural Network for real-time object recognition”. In: *Ieee/rsj International Conference on Intelligent Robots and Systems*. 2015, pp. 922–928.

[15] Hugues Thomas et al. “KPConv: Flexible and Deformable Convolution for Point Clouds”. In: *Proceedings of the IEEE International Conference on Computer Vision* (2019).

[16] “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Communications of the ACM* 24 (1981), pp. 381–395.

[17] Matt Weed. *Sensor(y) Overload: Making Sense of Lidar @ONLINE*. 2022. URL: https://www.luminartech.com/sensory-overload-making-sense-of-lidar/.

[18] Alexey Dosovitskiy et al. “CARLA: An Open Urban Driving Simulator”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[19] J. Behley et al. “SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences”. In: *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*. 2019.

[20] CARLA Team. *CARLA: Open-source simulator for autonomous driving research @ONLINE*. 2022. URL: https://carla.org/.

[21] Open3D. *Point Cloud @ONLINE*. 2022. URL: http://www.open3d.org/docs/latest/tutorial/Basic/pointcloud.html.

[22] W. S. Cleveland. “Robust Locally Weighted Regression and Smoothing Scatterplot”. In: *Journal of the American Statistical Association* 74.368 (1979), pp. 829–836.

[23] C. E. Rasmussen C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006. ISBN: 0-262-18253-X.

[24] “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: *AAAI* (1996), pp. 226–231.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[26] Yaoshiang Ho and Samuel Wookey. "The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling". In: *IEEE Access* PP (Dec. 2019), pp. 1–1. DOI: 10.1109/ACCESS.2019.2962617.

[27] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2015).

[28] Qingyong Hu et al. "Learning Semantic Segmentation of Large-Scale Point Clouds with Random Sampling". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2021).

[29] David Powers. "Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness Correlation". In: *Mach. Learn. Technol.* 2 (Jan. 2008).

[30] NumPy. *numpy.gradient @ONLINE*. 2022. URL: https://numpy.org/doc/stable/reference/generated/numpy.gradient.html.

[31] scikit learn. *sklearn.gaussian_process.GaussianProcessRegressor @ONLINE*. 2022. URL: https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessRegressor.html.

[32] scikit learn. *sklearn.neighbors.KDTree @ONLINE*. 2022. URL: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html.

[33] Unreal Engine. *LiDAR Point Cloud Plugin Overview @ONLINE*. 2022. URL: https://docs.unrealengine.com/5.0/en-US/lidar-point-cloud-plugin-overview-in-unreal-engine/.

9 Appendix

A F_1-scores and Skewed Datasets

The F_1-score does not contain information regarding negative cases which is evident in how True Negatives are not included in the metric. Thus, a model predicting outcomes on a dataset containing skewed distribution with very few negative outcomes may not receive a bad F_1-score even though it does not correctly predict a single negative outcome. For example, if the dataset contains 1000 positive outcomes and 10 negative outcome and the model predicts all instances as positive as seen in Table 14, the recall will be 1 (recall = 1), the precision will be (precision = $\frac{1000}{1010} \approx 0.990$) and the F_1-score will be $F_1 = \frac{1000}{1005} \approx 0.995$ even though it has not correctly classified a single negative outcome. Thus, it is necessary to be critical of the data and not rely blindly on the scores.

Table 14: Contingency table for a case where a "bad" model that predicts all instances as positive, produces a good F_1-score.

Actual \ Assigned	Test outcome positive	Test outcome negative	Total for condition
Condition positive	1000	0	1000
Condition negative	10	0	10

B Gaussion Process Regression

All content in this section is retrieved from [23].

B.1 Including noise

It is good to note that joint posterior in Equation 13 holds for the case when there is no noise, i.e $y = f(\mathbf{x})$. Here, the posterior perfectly incorporates the conditioned observations with zero variance at the observations. Adding noise however, adds uncertainty for these observations which is more in line with most situations, where there is only access to noisy measurements of function values. In these cases, observations are modeled as

$$y = f(\mathbf{x}) + \epsilon. \tag{37}$$

Here, ϵ is assumed to be independent identically distributed Gaussian noise with variance σ_n. This changes the covariance structure between observations from consisting of solely the covariance kernel $cov(y, y') = k(\mathbf{x}, \mathbf{x}')$ to include a noise term as seen in Equations (38).

$$cov(y, y') = k(\mathbf{x}, \mathbf{x}') + \sigma_n^2 \delta \tag{38a}$$

$$cov(\mathbf{y}) = K(X, X) + \sigma_n^2 I \tag{38b}$$

In these equations δ represents the Kronecker delta, I is the identity matrix and $\mathbf{y} \in \mathbb{R}^{n\times 1}$ is column matrix of observations with inputs $X \in \mathbb{R}^{n\times d}$. With this added noise term, the joint distribution of the observed values under the prior becomes

$$\begin{bmatrix}\mathbf{y}\\ \mathbf{f}_*\end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix}K(X,X)+\sigma_n^2 I & K(X,X_*)\\ K(X_*,X) & K(X_*,X_*)\end{bmatrix}\right). \tag{39}$$

This produces a joint distribution, seen in Equations (40), which is quite similar to the case without noise [23]:

$$\mathbf{f}_*|X_*, X, \mathbf{y} \sim \mathcal{N}(\bar{\mathbf{f}}_*, cov(\mathbf{f}_*)) \tag{40a}$$

$$\bar{\mathbf{f}}_* = \mathbb{E}[\mathbf{f}_*|X_*, X, \mathbf{y}] = K(X_*,X)[K(X,X)+\sigma_n^2 I]^{-1}\mathbf{y} \tag{40b}$$

$$cov(\mathbf{f}_*) = K(X_*,X_*) - K(X_*,X)[K(X,X)+\sigma_n^2 I]^{-1}K(X,X_*) \tag{40c}$$

Such a process can be seen in Figure 33. Here, the impact of noisy measurements are clearly seen as even though the variance shrinks close to the observations, it is not zero. Also, the mean function is not required to pass through the observations, decreasing the impact of outliers.

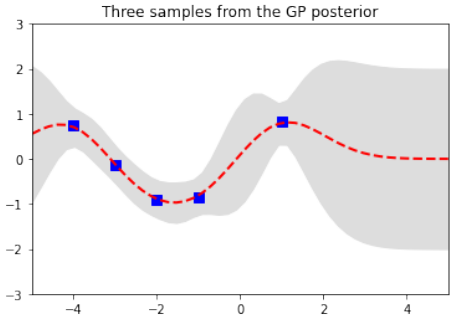


Figure 33: Posterior for a Gaussian process with squared exponential covariance kernel with $l = \sigma = 1$ and $\sigma_n^2 = 0.05$ given $X_{train} = [-4, -3, -2, -1, 1]$ and $y_{train} = sin(X_{train})$. The red line is the mean value for the posterior and the gray area marks $\pm$ 2 standard deviations.

C Fitting a Gaussian Process Regression Model

A Gaussian process regression model is fit by optimizing the hyperparameters of the kernel for a given set of values for the independent values X_{train} (observations X in Equations (13), (40b) and (40c)) and corresponding dependent values $\mathbf{y}$. Optimization is done by maximising the log marginal likelihood $log\, p(y|X, \theta)$. The marginal likelihood can be described as the probability of receiving the outcome $\mathbf{y}$ given X assuming a Gaussian process with covariance

function $k(x_i, x_j)$ with hyperparameters θ. For noisy targets $\mathbf{y}$ given by Equation (37), where $K_y = K_f + \sigma_n^2 I$ is the covariance matrix for these noisy targets, the log marginal likelihood is [23]

$$\log p(\mathbf{y}|X, \theta) = -\frac{1}{2}\mathbf{y}^T K_y^{-1}\mathbf{y} - \frac{1}{2}\log|K_y| - \frac{n}{2}\log(2\pi). \tag{41}$$

Note that assuming no noise in targets but including the noise in the kernel function instead (as in Equation (14)), the same result is achieved. The partial derivatives w.r.t the hyperparameters of the function above becomes [23]:

$$\frac{\partial}{\partial\theta_j}\log p(\mathbf{y}|X, \theta) = \frac{1}{2}\mathbf{y}^T K^{-1}\frac{\partial K}{\partial\theta_j}K^{-1}\mathbf{y} - \frac{1}{2}tr(K^{-1}\frac{\partial K}{\partial\theta_j}). \tag{42}$$

These partial derivatives are of the matrix K, but can be expressed, for simplicity, for the kernel element-wise instead as seen in Equations (43) [10].

$$\frac{\partial k(x_j, x_i)}{\partial l} = \sigma_f^2 \exp\left(-\frac{(x_i - x_j)^2}{2l^2}\right) \cdot \frac{(x_i - x_j)^2}{l^3} \tag{43a}$$

$$\frac{\partial k(x_j, x_i)}{\partial \sigma_f} = 2\sigma_f \exp\left(-\frac{(x_i - x_j)^2}{2l^2}\right) \tag{43b}$$

$$\frac{\partial k(x_j, x_i)}{\partial \sigma_n} = 2\sigma_n \delta_{ij} \tag{43c}$$

C.1 Accounting for noise in Kernel

The squared exponential kernel in Equation 14 includes measurement noise in the kernel directly instead of in the model. Thus, there are different methods of incorporating and interpreting the noise. One way is to regard the kernel in Equation 14 as the covariance for the measured outputs as in [23], i.e as $cov(y, y') = k(y, y')$. In this case, all the results for Gaussian process regression with noise hold, except for the noise is incorporated in the covariance matrices/functions in Equation 39 and 40c. However, in [10], this kernel is used directly in the equations for noiseless GPR as a way of including noise for these points. The linear predictor in Equation 13 will still be the same as in the case with noise (Equation 40b). The same does not hold for the variance since the $K(X_*, X_*)$ term in Equation 13 will include the noise $\sigma_n^2 I$ which is not included in Equation 40c. Whereas these discrepancies produce theoretically different distributions, the predicted function will be the same for the same set of hyperparameters $\theta = (l, \sigma_f^2, \sigma_n^2)$.

The GPR with noise and GPR without noise (but with noise included in the kernel) form the same log marginal likelihood and therefore also partial derivatives in Equation 41, 42 and 42. Thus, the optimization will converge to the same solution. Since they also give the same linear predictor, both methods give equivalent results in the context of regression.

D Further Discussion of Ground models

D.1 Impact of specificity

The proportion of *False Positives* were generally quite low, however this was likely due to the datasets being skewed to containing more ground points. Thus, the specificity, or true negative rate (see Equation (29)) is a better measure. That the RLWR-based model generally had very high specificity indicates that it was quite conservative and did not miss-classify non-ground to a greater extent. In contrast, the worst Hybrid model and Plane model performed a lot worse in this regard. It is good too keep these numbers in mind when comparing the models. Depending on the use case, miss-classification of non-ground as ground might be detrimental and in this case the RLWR-based model may be of greater interest. These effects are less visible in *accuracy* and F_1-scores due to the datasets being slightly skewed. However, it should be pointed out that the RLWR-based model in this thesis was solely designed for the use of point filtering. Should the model have been adapted for the use of predicting ground height, the consensus method of comparing points stripe-wise may have been replaced by another method.

D.2 Further Discussion of RLWR-based model's poor performance

There are other potential causes of the discrepancies between the results in [9] and [10] compared to this thesis for the RLWR-based model. They might have been caused by errors in the implementations. The authors in [9] were quite unspecific by what they ment by "Down weighting" and "Reweighting" on page 2185 where the implementation of the model was described. When experimenting with choices of hyperparameters for this model, δs much smaller than $\delta = 0.1$ resulted in very few points being classified as ground. It amplified the sporadic nature of the classification. In the original paper, the authors used a change threshold of $\delta = 0.005$. This discrepancy is likely either due to differences in implementation or test environments.

D.3 How impact of sparse circles might have impacted GPR fit to be more conservative

The model attempted to fit the seed points from every cluster but seemed to struggle with all the empty bins in between. In two different cases, this could have had different effects.

1: For more evenly sparse circles, the regression seemed to become less variable and worse at adapting to points far below the "incorrect mean". These would incorrectly be seen as outliers. Such points would be predicted as closer to the mean value of the circle and therefore appear elevated. Examples of this can be seen clearly in *Matteanexet* where the back part was greatly occluded and contained mostly sporadic clusters of points. As seen in 21e and 21f the model has little variability in each circle.

2: In the case one or maybe two large gaps, there could also be the aspect of the kernel parameters being fit mainly for the part of the circle with most points. In these cases the mean would likely not be affected to the same extent. Since the regression would fit the kernel mainly based on the more continuous distribution of points, the model would predict a slow increase in height in the gap that joins the two separated sections. Such an affect seems to be visible in Figure 21, especially in the side views, as strings of points in the air. This is possible to see for the incorrectly classified hill at the back of *Emdala* and around the bushes in the back left of *Eslöv airport*. This behaviour can also be seen in the same figure for circles where the heights suddenly seem to elevate unexpectedly.

Master's Theses in Mathematical Sciences 2022:E23
ISSN 1404-6342

LUTFMA-3472-2022

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/