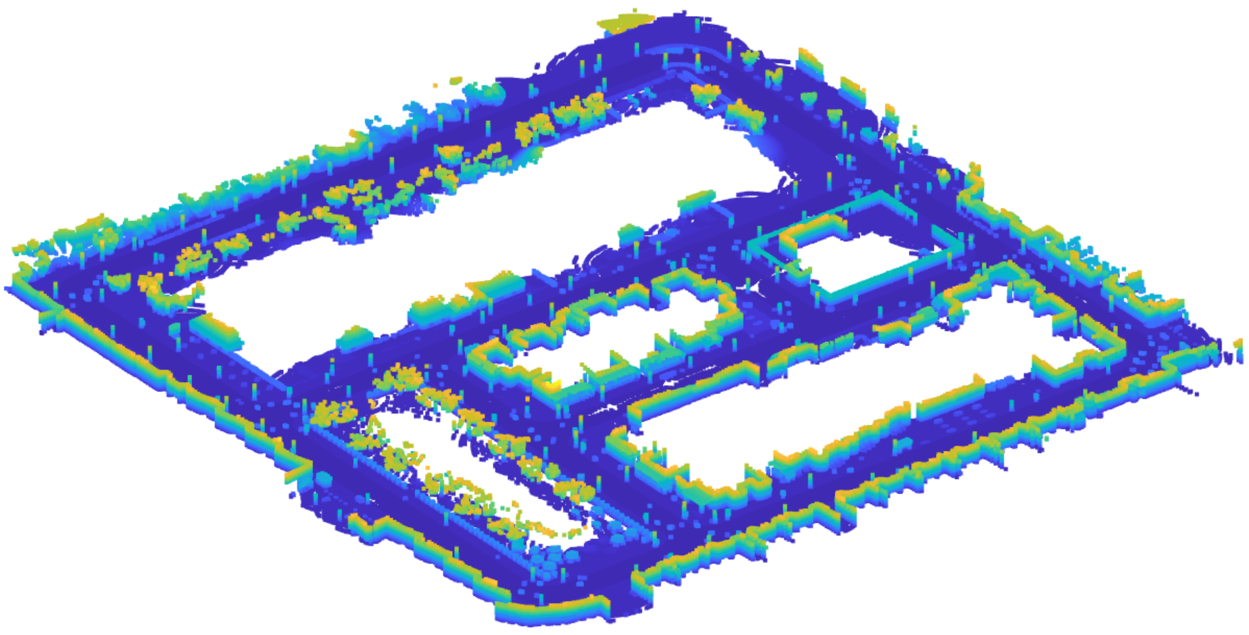
CHALMERS
UNIVERSITY OF TECHNOLOGY



# Map-based Localization Using LiDAR and Deep Neural Networks

Using regression to find rigid transformations between LiDAR scans and an a priori known map

Master's thesis in Systems, Control and Mechatronics

SABINA LINDEROTH
ANNIKA LUNDQVIST

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

Master's thesis 2019

# Map-based Localization Using LiDAR and Deep Neural Networks

Using regression to find rigid transformations between LiDAR scans and an a priori known map

SABINA LINDEROTH
ANNIKA LUNDQVIST

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2019

Map-based Localization Using LiDAR and Deep Neural Networks
Using regression to find rigid transformations between LiDAR scans and an a priori known map
SABINA LINDEROTH
ANNIKA LUNDQVIST

Supervisor and examiner: Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2019
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: LiDAR point cloud map for Town 2 in CARLA: Open-source simulator for autonomous driving research. The point cloud is visualized using MATLAB. The color coding represents height.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2019

Map-based localization using LiDAR and deep neural networks
SABINA LINDEROTH
ANNIKA LUNDQVIST
Department of Electrical Engineering
Chalmers University of Technology

## Abstract

The research on autonomous driving is expanding, and self-driving technology has the potential of transforming not only the transportation system, but our whole society. The ability to localize a vehicle in a map is an important piece of the puzzle in the development of self-driving vehicles. Different kinds of sensors can be used to detect the world around the vehicle. A suitable sensor in this case can be a LiDAR, which measures distances to objects using lasers. With this information we can create a map of the vehicle's surroundings that can be compared to a known map to find the vehicle's position.

This thesis investigates methods of localizing a vehicle in an a priori known map using a LiDAR sensor and neural networks. The LiDAR sensor yields a point cloud representation of the vehicle's surroundings and the data used in this project is collected from the simulation environment CARLA. The neural networks use the point clouds to predict how much we need to adjust an initial guess of the vehicle's position in the map to get closer to the true position. The thesis investigates two different network approaches using regression; one type of network that uses 2D LiDAR Bird's Eye View images as input, and another type of network that uses LiDAR point clouds as input.

The results show that pure translation can be found using regression by both types of neural networks, while rotation errors are more difficult to predict correctly. While none of the networks can compete with state-of-the-art methods, this thesis shows that neural network regression might have the potential to solve the localization task on its own.

Keywords: Localization, Map-Based Localization, Deep Learning, LiDAR, CARLA

## Acknowledgements

We want to express our gratitude to our supervisor and examiner Lennart Svensson, professor in Electrical Engineering, who has helped us throughout this project and provided us with valuable insights. We would also like to thank PhD student Anders Karlsson for his support and interesting discussions. Lastly, thanks to our course mates Simon Bastås and Robert Brenick for helping us with collecting data in CARLA when our software malfunctioned.

Sabina Linderoth, Gothenburg, June 2019
Annika Lundqvist, Gothenburg, June 2019

# Contents

# List of Figures

# List of Tables

# 1 Introduction

The research field of autonomous driving, ranging from computer vision to control theory, is of significant interest worldwide. Apart from the advantage of spending time in the car doing something else than driving, traffic accidents would most likely decrease dramatically if vehicles were automated. According to National Highway Traffic Safety Administration, 94% of all serious motor vehicle crashes are due to human errors [2]. It is a long leap from a standard vehicle driven and controlled by a human, to a fully autonomous vehicle that does not need any human interaction. The Society of Automotive Engineers International (SAE) describes six levels of automation that are widely used today to describe the level of automation of on-road vehicles [3]. Localization is an important piece of the puzzle, in order to reach the sixth and last level of automation.

## 1.1 Background

A fundamental problem in autonomous driving is the ability to localize the ego vehicle relative to its surroundings in real time. This is a key factor for autonomous driving systems such as Advanced Driver-Assistance Systems (ADAS), Advanced Emergency Braking System (AEBS) and Adaptive Cruise Control (ACC), which all rely on information of the ego vehicle's position relatively to other objects in order to give the ego vehicle correct control signals. Localization is therefore a field of active research where improvements lead directly to better self-driving vehicles.

Computer vision approaches are often used to perform localization. Great advances in image processing with help of deep learning have been a foundation for the perception systems where RGB-images can be used for object detection, ego motion estimation and localization. These system can however be vulnerable to different weather and lightning conditions, since an image of a scenery changes dramatically between day and night, at least if interpreted by a computer. Other sensors, such as Light Detection And Ranging (LiDAR) sensors, can be used as a complementary sensor, or on its own, to perform the same tasks. A LiDAR sensor measures distances to objects in the surrounding using lasers, with little sensitivity to weather or light conditions [4]. LiDARs can be used to construct high-resolution environment maps with which localization can be performed, for example by using filtering techniques as in [5] or by iterative methods of aligning point clouds as in [6]. With the introduction of deep neural networks, localization can now be performed using deep learning, as will be the case in this thesis.

This thesis proposes a method of localizing a vehicle using a LiDAR sensor, which creates a point cloud representation of the vehicle's surroundings. This point cloud is used as an input to a Deep Neural Network (DNN) with an ultimate goal of localizing the ego vehicle in an a priori known map. This kind of method using an a priori known map reduces the complexity of the localization task, from a perception problem (extracting lane markings etc.) to a localization problem matching current sensor data with a map. An additional usage of this kind of localization is to detect loop closure, which could further improve existing Simultaneous Localization And Mapping (SLAM) algorithms.

## 1.2 Related work

Despite decades of active research, localization still remains an area where there is room for improvement which is crucial for the development of autonomous driving. For a vehicle to get from point A to point B, precision of a few meters given by a GPS can be sufficient, but for safe motion planning in complicated environments, centimetre precision is desired. Geometric methods are often utilized to achieve high-precision localization, such as Iterative Closest Point (ICP) which can be used for point set registration [5]. These methods are vulnerable to repetitive environments, such as bridges and tunnels, if the environment is geometrically non-distinctive. Image-based methods can also be used for localization [7], but are often less accurate in outdoor environments compared to the geometric methods and require that season changes and daylight conditions are taken into consideration. Simultaneous Localization and Mapping (SLAM) is another method that is widely used for navigation of autonomous vehicles. The method serves, as the name indicate, to build a map of an unknown surrounding of a vehicle or update a current map, and at the same time localize the vehicle in the map. To succeed with this task SLAM uses Bayesian statistics, often with Kalman filters and in combination with at least one LiDAR sensor [8].

LiDAR point clouds can be used as input to neural networks for many purposes, ranging from object detection, semantic segmentation to localization. One way of localizing using LiDAR data is to create an intensity map in a grid-like structure, where each pixel represents the intensity in the corresponding area in the world as Barsan et al. suggest [9]. In this paper, they process the pre-built map with a neural network into an online embedding, and compare it in real time with LiDAR sweeps that are processed in the same way. Among the networks considered in [9], the neural network that performs best is inspired by LinkNet [1]. The comparison is done by computing the cross-correlation between the map and the sweep to find the position in the map that yields the highest activation. Rotation is handled by rotating the sweep and performing cross-correlation for different rotation angles. The method is fast and accurate, but requires a good initial guess of the vehicle's position in the map, and the output can not achieve a higher precision than the resolution of the grids or the rotation angles that are investigated in the cross-correlations.

When creating 2D images of LiDAR data, information like height is lost. Another way to handle LiDAR data without altering the detections is to use the point cloud coordinates directly as an input to a neural network. This also avoids rendering unnecessary voluminous data where grids in the 2D-picture would contain no detections, which is often the case with LiDAR data. The pioneers in the area of processing un-ordered sets of point clouds with neural networks are Qi et al. which introduced the network PointNet [10]. Their network succeeds in performing classification, part segmentation and semantic segmentation, but has not been used to perform localization. Zhou et al. applied this idea on LiDAR data with VoxelNet [11] to perform 3D-detection. The input point cloud is divided into voxels where each voxel is processed separately by stacked voxel feature encoding (VFE) layers to later be used in a region proposal network. The performance is strong, but the network is too slow to deploy in real time. A continuation on VoxelNet is provided by Lang et al. with their network PointPillars [12]. The goal is still to perform 3D object detection, but by using pillars instead of voxels, time-consuming 3D-convolutions can be replaced by 2D-convolutions which makes the network faster. PointPillars outperformed all state-of-the-art 3D detection networks when it was published, on KITTI benchmark data sets.

There are other types of neural networks developed whose purpose are not localization using LiDAR, but are still closely related to our problem formulation in this thesis. Flownet [13] has been successful in using 2D convolutions to estimate optical flow in image pairs. This indicates that neural networks have the capability to compare two images and estimate movement between them. Other networks, such as DeMoN [14], prove that camera motion can also be estimated from an image pair. Even more interestingly, iteration of the network has been shown to further improve these estimations. Although convolution neural networks (CNN) are very successful in a variety of computer vision tasks, there are some seemingly easy tasks that are surprisingly difficult for the CNN to solve. Liu et al. [15] show that CNNs perform poorly when ask to render an image and mark a specific pixel, given the pixel's coordinate as input. The problem is that convolutions are equivariant, indicating that the convolutional kernels don't know where in the image they are performing their operations. The solution is as simple as the task - introduce extra information in the image in the form of layers with pixel coordinates. This idea could also be used in this thesis, for the network to identify the same structure in two images and find the movement between them.

While localization using LiDAR data is not a new idea, many methods mentioned so far suggest transforming the data into a 2D representation, thus loosing information about height and limiting the accuracy to the resolution of the grids. We propose networks that instead utilize regression to estimate a rigid transformation, with the potential of yielding a more accurate result independently of the magnitude of the rigid transformation. Further more, we plan on using the main ideas of PointPillars to directly use a LiDAR point cloud as input, thus not loosing valuable information in the point cloud coordinates. The learned features of the pillars can then be processed with 2D convolutions and again use regression to estimate the final rigid transformation. We plan on getting inspiration for network architectures from the mentioned network that estimate camera motion and optical flow, as well as the networks that successfully performs 3D object detection. If the task proves to be difficult for the networks, we will implement inputs with extra pixel coordinate layers as suggested by Liu et al. [15].

## 1.3 Problem formulation

The localization task is performed by a DNN where the input consists of the current surroundings of the ego vehicle and a cut-out from the map at an initial guess of the vehicle's position and heading. The output of the DNN is a vector containing the rigid transformation which corrects the initial guess of the vehicle's position and heading in the map. A DNN of this kind can be designed using many different kinds of inputs. In this thesis project, we have focused on using either the point cloud generated by the LiDAR directly as input, or projecting the point cloud onto a surface creating a 2D image.

The project started with gathering LiDAR sensor data from a simulated urban environment called CARLA [16], which will be further introduced in Section 1.3.3. The sensor data is used to create a point cloud map of all the roads and surroundings in the environment. Training data is created from the sensor data as well, mainly by rotating and translating the LiDAR-measurements arbitrarily. The networks developed in this thesis are divided into two different kinds, depending on which input that is used. Firstly, we focus on developing networks that use 2D images as input, and after that on networks that use point clouds as input. The networks have in common that they use regression to estimate the rigid transformation between the sweep and map cut-out.

### 1.3.1 Aim

The aim of this thesis project is to develop a DNN that can localize an ego vehicle in an a priori known map, by matching current LiDAR measurements with the map. The goal is to develop a network that will localize the vehicle with a centimetre level accuracy using neural network regression. We will develop two different types of networks. One type that takes 2D top-view LiDAR images as input and one type that uses the 3D point cloud generated from the LiDAR as input.

### 1.3.2 Scope and limitations

The scope of this thesis is to implement the last step of a localization process, where ideally multiple sensors have been involved to give an initial prediction of the ego vehicle's position and heading in the map. We will not delve deeper into the sensor fusion behind this process, but focus only on the localization task using a LiDAR and a predicted position and heading of the vehicle.

The training and testing of the DNNs are performed using data from diverse urban environments in the simulation environment CARLA, i.e we will not collect and use data from real environments. Situations where the surroundings are similar over time such as monotonous highways, long tunnels and bridges are not of interest in this thesis since it is considered to be out of scope of the problem. Since dynamic object removal is not in the scope of this thesis, the data used in this project will only contain static objects.

A limiting factor in this thesis work is the amount of time needed to train the neural networks.

### 1.3.3 Tools and equipment

This section describes the tools and equipment that were used, starting with the hardware and continuing with the simulation environment CARLA and finally the open source deep learning platform Pytorch.

#### Hardware

Training neural networks seems to always need more time and computational power than expected. In this project a Google Cloud instance was used when collecting data in the simulation environment and when training the networks. The instance was equipped with a SSD of 25 GB, 4 NVIDIA Tesla K80 GPU with 12 GB memory each and 8 virtual CPUs with 30 GB memory.

#### CARLA

CARLA is an open-source simulator for autonomous driving research [16]. The simulation environment can be customized freely and contains features such as different weather conditions and urban environments where other vehicles and pedestrians can be present. The software contains a Python API that allows the user to control the simulation environment and its actors, eg. a vehicle, in the simulation. There is also a built in autopilot, which allows the actor to drive randomly while respecting traffic rules and other vehicles and pedestrians. The actor in the simulation can be equipped with other actors such as different kind of sensors, eg. a LiDAR sensor, which has been used in this project. CARLA provides information about the actors such as the global position of the LiDAR sensor and the point cloud coordinates in each time frame generated by the LiDAR. Information about the actors such as velocity and global position can also be collected. The CARLA version 0.9.2 was used in this project.

#### PyTorch

PyTorch is an open-source deep learning framework developed primarily by Facebook's artificial-intelligence research group. The PyTorch framework is a Python-based scientific computing package which has two main features; a NumPy substitute to be able to use the power of GPUs and a platform for deep learning research that provide maximum flexibility and speed.

## 1.4 Contribution

The contributions from this thesis are the deep learning network architectures, that can solve the localization task in novel ways using LiDAR data. The networks can use different kind of inputs created from the LiDAR data, and the localization is solved using regression. The results can not compete with state-of-the-art methods used today, however the methods in this thesis could be promising with more research.

## 1.5 Report outline

Following this introduction chapter, the theory behind this thesis is presented. Key concepts of deep learning will be introduced, both regarding the theory behind neural networks and how to train them. Chapter 3 describes the methods that have been used in this thesis project, e.g. how data was gathered and pre-processed and which networks that have been developed. Chapter 4 presents the results, along with some brief interpretations and comparisons of the outcomes. The results are discussed in more detail in Chapter 6, and the thesis is finally concluded in Chapter 7.

# 2 Theory

In this chapter, the theoretical background for this project will be described. First the function of a LiDAR sensor will be presented in Section 2.1 followed by the Sections 2.2-2.3 that cover the theoretical basics of artificial neural networks and how to train them.

## 2.1 LiDAR sensor

LiDAR sensors, Light Detection and Range, are built on the simple fact that light which hits an object will reflect back to the light source. A LiDAR sensor utilizes this fact by sending out laser beams and measure how long it takes for the beams to get back to the sensor. Since the speed of light has a known value the distance to an object can be calculated with high accuracy as,

$$\text{Distance} = \frac{(\text{Speed of light}) \times (\text{Time of flight})}{2}$$

where Time of flight is the time it takes for the laser beam to travel back and forth. There are three different kinds of LiDAR sensors; 1D-, 2D- and 3D-LiDAR. The 1D-LiDAR sensor only measures the distance, while a 2D sensor also register the scanning angle of the beam in the $xy-$plane to find the $x$ and $y$ coordinates of a reflective object. A 3D-LiDAR consists of a set of 2D-LiDAR sensors, arranged in different heights which allows the sensor to also find the $z$-coordinate of an object. The number of lasers that the 3D sensor uses is called the number of channels. Something in common for most of the LiDAR sensors is that they rotate continuously around their center in the horizontal plane and generates a 3D point cloud of its surroundings. The scanning angle, both in the horizontal and vertical plane, is often denoted as the field of view (FOV). Since a LiDAR also detects the intensity of the reflected light it can be used to get information of the structure of the surface that is hit by the beam [17].

A LiDAR can be mounted on an autonomous vehicle, e.g on the roof of a car, which makes it possible for the vehicle to use the information generated by the sensor to navigate itself. The maximum distance that can be detected is called the sensor's range which typically is about 100 m [17].

## 2.2 Artificial Neural Networks

Artificial neural networks (ANNs) are inspired by the brains of mammals as humans and higher animals, which are known to be very complex objects. The computational capability in today's computers are much smaller than the one in a typical brain, however they share some common properties. Similar to a brain, an ANN consists of neurons which are components that interconnect in order to solve tasks such as function approximation or classification. Function approximation has as goal to find a representation of a function $y = f(x)$ for $x$ in the domain of $f$. To find $f(x)$ the network needs to be trained and this is often done using either supervised or unsupervised training. When using supervised training, one normally uses training samples that cover a subset of the domain of interest. For every input to the network there must exist a desired output. By comparing the desired output with the prediction that is given from the network, an error signal can be generated and used when training the network to perform better. A typical example of supervised learning is the classification problem, where the network's task is to classify an input among a set of pre-defined categories. A famous data set for classification is the data set MNIST which contains images of hand-written digits between 0-9 with a label describing the digit in the image. A neural network should in this case learn to categories the input images in to 10 different classes, one class for each digit in the numerical system. This simple data set is often used as a test in deep learning research [18]. Another example of supervised learning are regression tasks, where the network predicts a numerical value. A typical task can be to predict future prices of real estate, based on past labeled data.

Unsupervised training is used when an error signal can not be created. The goal with the training is that the ANN should be able to provide an output that consists of the correct prediction of the intermediate points [19]. Since the training data is not labeled, the goal is often to let the network group similar data points together.

The neural networks in this thesis project are examples of feed-forward neural networks (FFNN). The name stems from the fact that the input $x$ flows through the network to create the output $\hat{y}$, which is compared to the label $y$ assigned to $x$ [18]. Basic theory of neural networks will be presented in this section, specifically fully connected layers, convolutional layers and activation functions, that are the foundations of the networks in this project. How to train neural networks is described in Section 2.3.

### 2.2.1 Fully connected layers

A simple feed-forward neural network can consist of fully connected layers, where each neuron in every layer is connected to all the neurons in the following layer. A network usually consists of multiple of these fully connected layers, yielding a deep neural network that is capable of learning complex functions. See Figure 2.1 for a simple representation of a fully connected neural network.

Each neuron calculates a linear combination of all the inputs and this sum is passed through an activation function as output. The weights are the learnable pa-

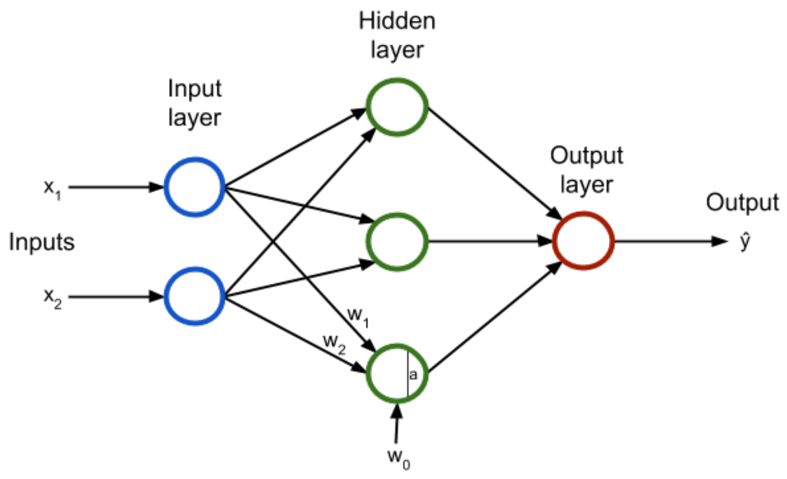


**Figure 2.1:** A simple fully connected neural network with one hidden layer. The blue circles represent the input layer, the green circles the hidden layer and the red circle is the output layer. $x_1$ and $x_2$ are the inputs to the network with the corresponding output $\hat{y}$ $w_1$-$w_9$ are the learnable weights and $\hat{y}$. The calculation from Equation (2.1) is performed in each neuron. The weights and activation function are highlighted in one of the neurons in the hidden layer in the Figure.

rameters that are updated during the training of the neural network. The operation in a neuron with $N$ inputs can be described with the following expression,

$$y_j = a\left(w_0 + \sum_{i=1}^{N} w_i x_i\right) \tag{2.1}$$

where $x_i$ is the $i$th input, $y_j$ is the output from the $j$th neuron and $w_i$ are the learnable weights where $w_0$ is a learnable bias. The activation function $a(\cdot)$ and its role will be described in more detail in Section 2.2.3. All neurons perform these calculations each forward pass when the input propagate through the neural network to calculate the final output. What makes the network so powerful is that each and every weight learns its value by being exposed to training where the correct output is known.

### 2.2.2 Convolutional Neural Networks

Another layer that can be used in a FFNN is the convolutional layer. A FFNN that consists of at least one convolutional layer is called a Convolutional neural network (CNN). CNNs are often used when the input to the network can be represented as a 2D grid, making them very useful when working with 2D images. The specific filtering operation of a convolutional layer is discrete convolution without flipping the kernel, which is a function called cross-correlation. The cross-correlation function is defined as,

$$S(i,j) = (K * I)(i,j) = \sum_{m}\sum_{n} I(i+m, j+n)K(m,n) \tag{2.2}$$

where $I$ is the input image, $K$ the kernel and $i, j$ the size of the 2D input and kernel. The kernel can also be called filter and $S$ is the output feature map. The input is usually a multidimensional array of data and the kernel a multidimensional array of trainable parameters that are learned by the network [18].

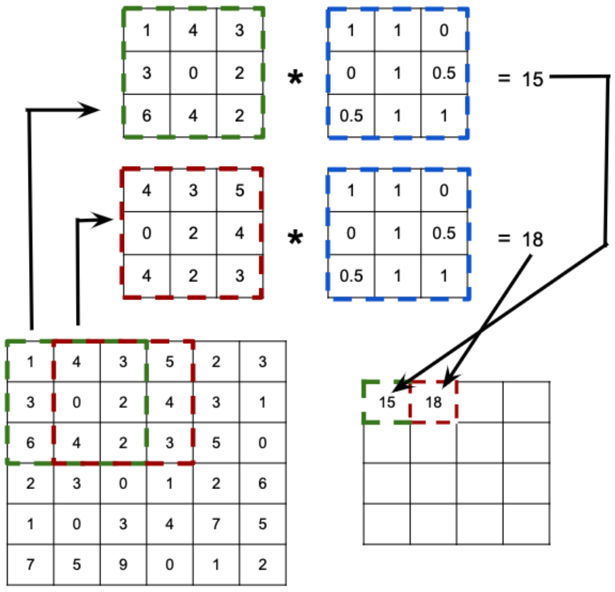


**Figure 2.2:** A schematic of the operation performed on the input in a convolutional layer. The 6×6 sized matrix to the left is the input image and the 4×4 matrix to the right is the output feature map. The filter (blue box) is applied two times. First it operates on the 3×3 sized box in the upper left corner (green box) and then on the red box. The stride length in this example is 1. The filter will sweep over the input one stride at the time and continue to fill the output feature map in the same way.

The filtering can be seen as extraction of important features from the input. The trainable parameters in the filters will during training of the network learn to extract specific features that are used in later layers to assemble more abstract features.Examples of features that a filter can extract in the shallow parts of the network, are sharp edges in the image and gradient shapes of different signals. The deeper the network is, the more abstract are the features that the filter tries to extract [18]. One can use several filters on one input which will result in that the output from that layer will have the same number of channels as the number of filters applied. The filter is applied on the input with a given stride length, i.e if having stride one the filter moves with one pixel increments. The output from the convolutional layer is given by the dot product between the filter and a local neighborhood of entries in the input. The output can be seen as a feature map which consist of valuable information where specific features are located in the input image. The receptive field of the network is how much of the input that is visible to each filter. The receptive field increases linearly with each layer, and depends on the size of the filters and the number of convolutional layers. The convolutional operation is schematically shown in Figure 2.2. As seen in the figure the output is of smaller size compared to the input. This is because the filter can not operate outside the input image's grid unless zero adding is applied. Zero padding is performed by adding an appropriate amount of rows and columns with zeros around the input image to get the same output size as input [18]. For example if using zero padding of size one in Figure 2.2, i.e two extra rows and columns around the grid, will yield a output of size 6×6.

Another layer that is typical for a CNN is the pooling layer. The pooling layer is used to summarize the information in a given neighbourhood of the output. This will make the information in the output feature map more compact and also make the representation more resistant to translations in the input image. There are different kinds of pooling layers, two examples are the max pooling layer that saves the maximum value of a rectangular neighbourhood in the output and the weighted average pooling layer which saves the weighted average of a square around a center pixel [18]. The max pooling operation is shown in Figure 2.3.

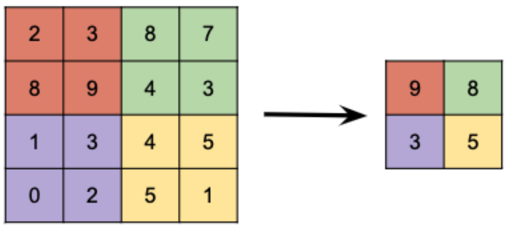


**Figure 2.3:** Schematic of the max pooling operation. The maximum value in each of the colored neighbourhoods (left square) is saved in a new more compact representation of the output (right square).

### 2.2.3 Activation functions

The role of an activation function is to introduce non-linearity to the output of a neuron. Consider the output from a neuron in layer 1, $z^{[1]}$,

$$z^{[1]}(X) = W^{[1]}X + b^{[1]}$$

where $X \in \mathbb{R}^{n\times m}$ is the input, $W[1] \in \mathbb{R}^{m\times n}$ the weights and $b^{[1]} \in \mathbb{R}^{m}$ the bias. $z^{[1]}$ will then be the input to a neuron in the next layer $z^{[2]}$. The output from that layer can be described as,

$$\begin{aligned} z^{[2]}(z^{[1]}) &= W^{[2]}z^{[1]} + b^{[2]} \\ &= W^{[2]}(W^{[1]}X + b^{[1]}) + b^{[2]} \\ &= W^{[2]}W^{[1]}X + W^{[2]}b^{[1]} + b^{[2]}. \end{aligned}$$

Let $W^{[2]}W^{[1]} = W$ and $W^{[2]}b^{[1]} + b^{[2]} = b$. This yields the final output from the second layer

$$z^{[2]} = WX + b.$$

This shows that without an activation function, the output from every layer will be a linear function of the input regardless of the number of layers. The non-linearity that the activation function introduces is necessary for the back propagation process, which will be described in Section 2.3.2. The derivative of $z$ must depend on the input $x$ in order to let the network learn to predict the output. If not introducing non-linearity, the derivative of $z$ is just a constant and not dependent on the change of the input. Introducing non-linearity enables the network to learn more complex mapping functions [20].

There are various kinds of activation functions that can be used in a neural network. Some of the most common ones are the Rectified Linear Unit (ReLU) function and the Tangens hyperbolicus (tanh) function. The ReLU function is the most common and recommended activation function used in feed forward neural networks and is defined as the maximum value of an input $x$ and zero,

$$ReLU(x) = max\{0, x\}.$$

If applied to a linear transformation the function yields a nonlinear transformation keeping the properties that enable linear models to be optimized with gradient based methods [18].

The tanh function, as the name indicate, is a hyperbolic function which returns the hyperbolic tangent of the input value. Tanh is defined as,

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

where x is the output value from the neuron. The output is bound to the interval [-1,1]. In Figure 2.4 the two activation functions are shown.

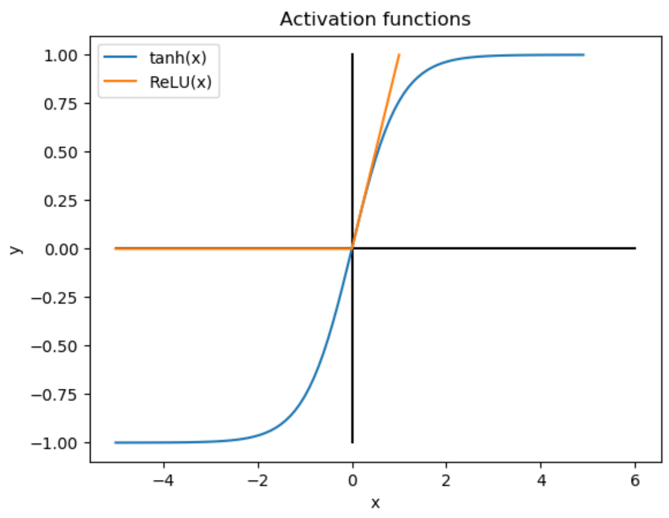


**Figure 2.4:** Graph showing the ReLU function (orange) and tanh function (blue).

## 2.3 Training an Artificial Neural Network

When training a network the task is to tune the learnable parameters to minimize the difference between the output of the network and the ground truth by using a training set. The training set is looped through more than one time. Each loop is called an epoch and since the network will see each sample more than one time it has more than one chance to learn from the sample. During training the trainable parameters of the network are updated to achieve the smallest possible error, ideally 0. Example of parameters that are updated during training are the weights and biases of the network. The difference between the network's output and the ground truth can be described using a loss function, $L(f(\mathbf{x};\, \theta), \mathbf{y})$, where $f(\mathbf{x};\, \theta)$ is the predicted output from the network, $\mathbf{x}$ the input, $\theta$ the learnable parameters and $\mathbf{y}$ is the ground truth. The loss function can be defined in different ways, some of the variants will be presented in Section 2.3.1. The update of the parameters can be done in various ways and one optimization method that can be used is the gradient decent algorithm. In gradient decent the parameters are updated accordingly,

$$\theta_{i+1} \leftarrow \theta_i - \mu \frac{\partial L(f(\mathbf{x};\, \theta), \mathbf{y})}{\partial \theta_i} \tag{2.3}$$

where $\mu$ is the learning rate which is a hyper parameter that decides how large step to take in the update and $\frac{\partial L}{\partial \theta_i}$ is the gradient of the loss with respect to the $i$-th trainable parameter that is updated [18]. To calculate the gradients $\frac{\partial L}{\partial \theta_i}$ with respect to each parameter back propagation is used which will be described in Section 2.3.2.

### 2.3.1 Loss function

When training a network, the objective is to model the input data as accurately as possible. To determine the performance of the network, loss functions are used. Normally the goal is to minimize the error between the predicted estimate given from the network and the ground truth. This means that if the error is large the value given from the loss function will take a large value. To minimize the error a optimization algorithm is used in combination with the loss function which decides how to update the network's weights in a optimal way. The main task for the loss function is that it must describe the good and bad aspects of a model in one number such that improvements in the number also reflects an improvement of the model's accuracy [21].

When dealing with a regression problem a loss function that can be used is the minimum squared error (MSE). MSE measures the average squared difference between the predicted value and the ground truth, which can be formulated as,

$$MSE = \frac{\sum_{i=0}^{n} (\hat{y}_i - y_i)^2}{n} \tag{2.4}$$

where $\hat{y}_i$ is the estimate predicted by the network and $y_i$ is the ground truth [21].

Another loss function used for regression problems is the Huber Loss or, as it also is called, SmoothL1 function. SmoothL1 is defined as,

$$L(\hat{y}, y) = \frac{1}{n} \sum_i z_i \tag{2.5}$$

where $n$ is the total number of elements and $z_i$ is given by,

$$z_i = \begin{cases} \frac{1}{2}(\hat{y}_i - y_i)^2, & \text{if } |\hat{y}_i - y_i| < \delta \\ \delta\,|\hat{y}_i - y_i| - \frac{1}{2}\delta, & \text{otherwise.} \end{cases} \tag{2.6}$$

The function is quadratic for values smaller than $\delta$ and linear for values larger than $\delta$, and therefore less sensitive to outliers compared to the MSE loss [22].

### 2.3.2 Backpropagation

To describe how the derivation of the loss' gradient, $\frac{\partial L}{\partial w_i}$, in Equation (2.3) is performed it is easiest to consider a simple network as in Figure 2.1. The loss function, in this example, can be defined as the squared distance between the predicted output, and the desired output, $L = (\hat{y} - y)^2$ and the parameters to be learned are the weights, $w_i$. The output from the $i$-th neuron in the input layer can be described as,

$$z_i^{[I]} = a\left(w_0^{[I]} + w_i^{[I]} x_i\right)$$

where the superscript $I$ referrers to the input layer, $z_i^{[I]}$ is the output from the input layer, $x_i$ the $i$-th input to the layer, $w_i^{[I]}$ the $i$-th weight of the neuron and $w_0^{[I]}$ the bias term.

The output from the $i$-th neuron in the hidden layer ($H$) is in a similar way described as,

$$z_i^{[H]} = a\left(w_0^{[H]} + \sum_{i=1}^{2} w_i^{[H]} z_i^{[I]}\right).$$

Since the $i$-th neuron in the hidden layer gets the weighted output from each neuron in the previous layer, these outputs needs to be summarized, thereof the summation in the expression.

Finally, the output from the neuron in the output layer ($O$) can be described as

$$y^{[O]} = a\left(w_0^{[O]} + \sum_{i=1}^{3} w_i^{[O]} z_i^{[H]}\right).$$

The derivation of the gradient, $\frac{\partial L}{\partial w_i}$ is performed by making use of the chain rule. The calculations in Equation (2.7) show derivations of three derivatives. The first one is the derivative of the loss with respect to the weights in the output layer, the second one is the derivative of the loss with respect to the weights in the hidden layer and the last third one is the derivative of the loss with respect to the weights in the input layer.

$$\begin{aligned} \frac{\partial L}{\partial w_i^{[O]}} &= \frac{\partial L}{\partial y_i^{[O]}} \frac{\partial y_i^{[O]}}{w_i^{[O]}} = 2(\hat{y} - y^*)a'(\cdot) z_i^{[H]} = \alpha^{[O]} z_i^{[H]} \\ \frac{\partial L}{\partial w_i^{[H]}} &= \frac{\partial L}{\partial y_i^{[O]}} \frac{\partial y_i^{[O]}}{\partial z_i^{[H]}} \frac{\partial z_i^{[H]}}{\partial w_i^{[H]}} = \alpha^{[O]} w_i^{[O]} a'(\cdot) z_i^{[I]} = \alpha^{[O]} \alpha^{[H]} z_i^{[I]} \\ \frac{\partial L}{\partial w_i^{[I]}} &= \frac{\partial L}{\partial y_i^{[O]}} \frac{\partial y_i^{[O]}}{\partial z_i^{[H]}} \frac{\partial z_i^{[H]}}{\partial z_i^{[I]}} \frac{\partial z_i^{[I]}}{\partial w_i^{[I]}} = \alpha^{[O]} \alpha^{[H]} w_i^{[H]} a'(\cdot) x_i. \end{aligned} \tag{2.7}$$

To update all the weights the same derivations are made with respect to all the weights in the network [23].

### 2.3.3 Optimizers

When updating the weights in a neural network, the ambition is to update them in an optimal way. To do this an optimization algorithm is used. A very important parameter when choosing optimization algorithm is that it should operate as efficiently as possible to reduce the training time. To find the optimal parameters different optimizers can be used. The gradient decent (GD) optimizer, that is mentioned in Section 2.3, is one of them. When using GD, the gradients for all samples in the training set are calculated to make an update step. This results in a very time consuming and computationally heavy operation if the training set consists of many samples, which is one reason to why the GD is seldom used in machine learning applications. A commonly used optimizer is instead the stochastic gradient decent optimizer (SGD), which is a modification of GD. Unlike GD, SGD updates the coefficients after each data sample, or after a mini batch of randomly selected samples. The samples, and the mini-batches, have to be shuffled in order to calculate an unbiased estimate of the gradient. When using SGD the choice of hyperparameters is of great importance to tune the algorithm. One example of an important hyper parameter is the learning rate. When using SGD the decision to use learning rate decay needs to be set manually which highly can affect the performance [18].

An extension to SGD is the Adam optimization algorithm. Adam is computationally efficient and is well suited for problems that contains a large amount of data and parameters. Unlike SGD, Adam uses estimates from the first and second moments to compute adaptive learning rates for different individual parameters and typically needs less tuning [24].

### 2.3.4 Data sets

The data that is used when training a network is commonly divided into three different data sets; a training set, a validation set and a test set. The training set is usually the largest of the three sets and is used during training of the network in the optimization step where it is used to update the learnable parameters. This means that the network model is exposed to the training data a lot of times and learns from it. If the model has a low error on the training set it is said to have a high capacity i.e it can fit the training set. The validation set is used to evaluate how well the model performs on data it is not allowed to train on. The validation set can be used to fine tune the model's hyperparameters, i.e the model is allowed to see the set but does not learn from it. However, since the validation set is used to fine tune hyperparameters, the model is in some way influenced by it. The validation set can also be used to determine when to stop training. The test set is used to evaluate how well the final model performs on unseen data. The test set is usually the smallest set and is only used once. The error on the test set represents how good the final model is and when the test error is low it means that the model can generalize well to new data. The ratio between the training, validation and test set can vary depending on the size of the total amount of data, but a first split is usually made with the ratios 0.7, 0.2, 0.1 for training, validation and test respectively [19].

### 2.3.5 Regularization methods

One of the main challenges of machine learning problems is that the algorithm should perform well on input data that has not been seen before, this ability is called generalization. If trained enough, a network can learn to represent almost any set of data. However at some point the predictive performance of the network will reach a peak followed by a decrease of the predictive power. This is called overfitting, which is a consequence of that given to much training, the network will start to fit the noise in the training data [19]. In order to prevent overfitting, regularization methods can be used. Regularization are any modifications made to the learning algorithm with the purpose of reducing the generalization error, sometimes at the expense of increased loss at the training set [18]. There is no regularization method that is the best one, instead the choice of the most suitable regularization method will depend on the task that is to be solved. In this section, different regularization methods that are widely used will be presented.

#### Early stopping

Early stopping is used to interrupt the training of the network when its predictive power starts to decrease. A performance metric is monitored and when the metric is met the training process is stopped. During training the error with respect to the validation set is often set to be the monitored metric. This metric represents how well the network performs on unseen data. The error normally decreases in the beginning of the training, but when the network starts to overfit the training data, the validation error starts to increase, whilst the training error continues to decrease. To achieve a network with good generalization performance, the model is saved at the point of the smallest validation error [25].

#### Dropout layers

Dropout is a computationally inexpensive but powerful method to prevent overfitting when training neural networks. The main idea is simple; randomly selected neurons are multiplied by zero and therefore ignored during training. The neurons are retained with a probability $p$, independently of other neurons. The contribution from these neurons are temporarily lost, which forces the network to update the remaining weights to perform the task of the network. This is thought to prevent single weights to specialize in specific feature extractions, and training the model to learn multiple independent representations. A drawback of dropout is that it requires more training time, since not all weights are updated each pass through the network [26].

### Data augmentation

A way to get a machine learning model to generalize better on unseen data is simply to train it on more data. If the amount of data is limited, a solution can be to create fake training data from the existing data and then add it to the training set. Consider the situation when having an image as input to a neural network. The image can then be modified in different ways, e.g translated, rotated and scaled [18].

### Batch Norm layers

Normalization is a common pre-processing of the input data to get a more symmetric representation of the data, which in many cases can speed up the training. Batch Norm (BN) works in a similar way, but instead of just normalizing the input, BN normalizes the values inside the network, more specifically, the output from the previous hidden layer. BN can be applied both before and after the activation function but an advice from Andrew Y. Ng is to use it before the activation function [27].

The BN calculation is performed as follows, consider some intermediate values in a neural network, $z^{[l](1)}, ..., z^{[l](n)}$ where $[l]$ represents the l-th layer in the network and $(1), .., (m)$ represents the 1-st to m-th neuron in a layer. For simplicity the superscript $[l]$ is left out in the following calculations. For each batch the normalization of the $z$-value is performed by subtracting the batch mean and dividing it by the batch standard deviation,

$$\begin{aligned}\mu &= \frac{1}{m}\sum_i z^{(i)} \\ \sigma^2 &= \frac{1}{m}\sum_i (z^{(i)} - \mu)^2 \\ z^{(i)}_{norm} &= \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}}\end{aligned}$$

where $\mu$ is the batch mean, $\sigma^2$ the standard deviation, $z^{(i)}_{norm}$ the normalized z-value and $\varepsilon$ is a small number to avoid division with 0. In the last step BN introduces the variable $\tilde{z}^{(i)}$ which includes two new trainable parameters $\gamma$ and $\beta$,

$$\tilde{z}^{(i)} = \gamma z^{(i)}_{norm} + \beta$$

$\gamma$ and $\beta$ is updated in the same way as the other trainable parameters in the network. $\tilde{z}^{(i)}$ is then used instead of $z^{(i)}$ as the input to the activation function in the neuron. The parameters $\gamma$ and $\beta$ makes it possible to chose the mean of $\tilde{z}^{(i)}$ according to what is suitable [27].

# 3 Method

This chapter describes the methods regarding data that have been used in this project, as well as all the networks that have been developed and the training procedures.

Section 3.1 describes how we gathered LiDAR data from the urban driving simulator CARLA and how the raw data was processed in order to speed up the training of the neural networks. Section 3.2 presents all aspects of the neural networks developed for 2D input data; creation of the training data, the network architectures and how the networks were trained. The last section in this chapter presents the networks that were developed for point clouds as input, as well as the creation of training data and how the networks were trained.

## 3.1 Collecting data in CARLA

The data used in this thesis project was collected in CARLA by connecting to the simulation server through python scripts. A car was fitted with a rotating LiDAR sensor. The LiDAR was equipped with 32 channels, a field of view in the range [-10,30] degrees, a rotation frequency of 10 Hz and a range of 100 m. The number of points generated by all lasers per second was 56000 points.

In order to get LiDAR data covering all roads in the simulation environment's towns, waypoints were placed evenly approximately 50 m apart. Waypoints are coordinates that the simulation deems as suitable to position the vehicle. The vehicle drove on auto pilot roughly 15 seconds with start in each waypoint. A LiDAR measurement was collected every 5 metres. The data was visualized using Matlab to ensure that the data covered the whole town. An example of the raw data is shown in Figure 3.1.

Since waypoints were placed on both lanes of the roads, each road is visited on average twice, gathering enough data to create a dense point cloud. The collected data from the LiDAR sensor is saved in a ply-file[1] for each LiDAR sweep. The file consists of three columns representing the coordinates $x, y, z$ where each row contains the coordinates for one detection made by the LiDAR. All detections were given in coordinates relative to the LiDAR which was located in the origin. The coordinates were transformed into global coordinates such that multiple sweeps could be visualized together. In order to do that, the global position of the LiDAR at each measurement was also gathered from the simulator and saved separately in a csv-file.

[1]Polygon File Format (PLY), originally developed to store information from 3D-scanners.

Two different towns were used for training, validation and testing of the network. Town 1, which was the largest town, was used for training and Town 2 was used for validation and testing. To get two separate sets for validation and testing, Town 2 was divided into two geographically separated parts where the largest part was used for validation and the smaller part for testing. Bird's eye view images of all LiDAR sweeps covering both towns are shown in Figure 3.2. Town 1 and Town 2 consists of 2.9 km and 1.4 km drivable roads respectively [16].

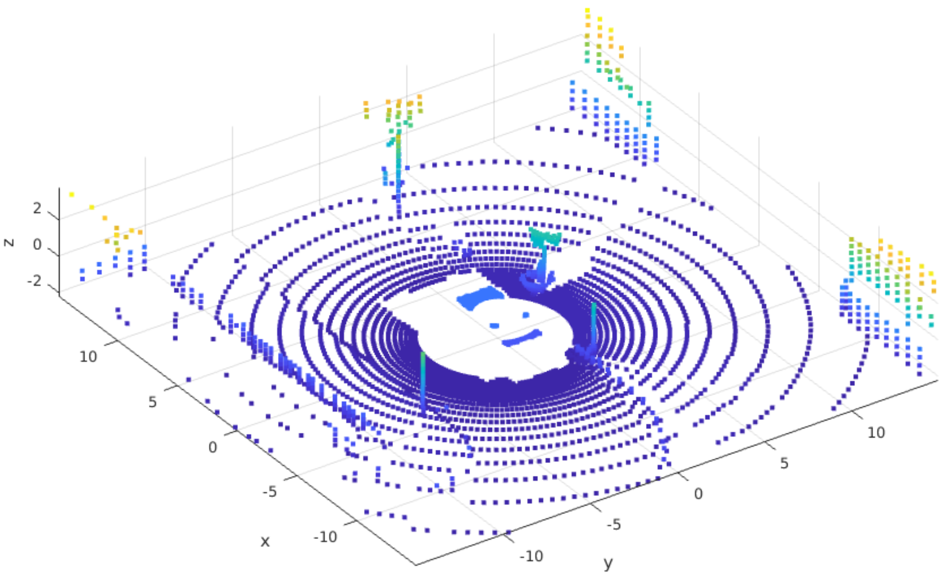


**Figure 3.1:** Raw data from one single LiDAR measurement visualized in MATLAB. Each point represents one detection, the points are color coded after height in this image. The side walks can be seen as the straight lines adjacent to the car, and some trees and houses are visible as well.

## 3.1.1 Pre-processing and sorting the data

The collected data from CARLA was converted, from point clouds with coordinates relative to the LiDAR, into global coordinates where each detection makes up a point in the town where the data was collected. For each sweep, all detections were rotated according to the vehicle's heading, and then translated to the vehicle's global coordinates in the town. Detections from the vehicle's roof and hood were removed, since they are not static objects that belong in a static map. In order to retrieve detections when creating training data later on, all detections were sorted into a spatial grid where each grid cell spanned 15 m $\times$ 15 m. For Town 2 this corresponds to 414 grids, and 832 grids for the larger Town 1. Each LiDAR detection was sorted into its corresponding grid, and all detections in each grid were then saved as a csv-file. When retrieving all points in a specific area of the town, it is enough to look up the grid containing the specific location, and all adjacent grids. This sorting

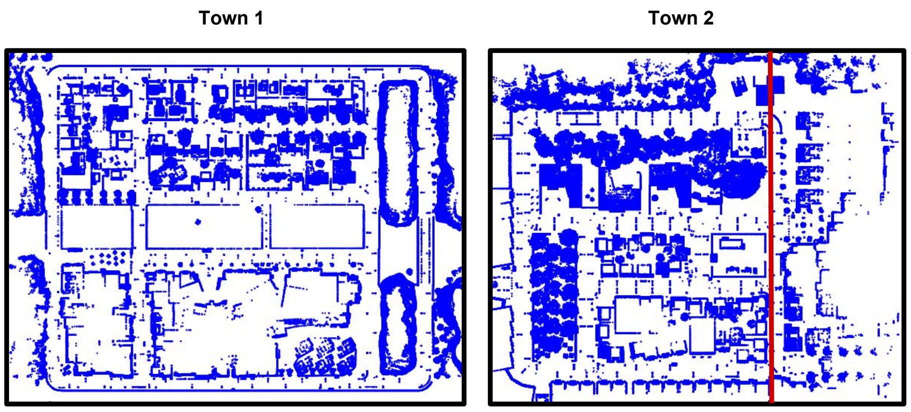


**Figure 3.2:** Bird's eye view images of Town 1 and Town 2. The red vertical line in Town 2 visualizes where the town is separated for validation and test set. The part on the right side is used for testing and the left is for validation.

speeds up the process of retrieving points corresponding to a certain region, since all relevant points are already gathered in one file which corresponds to $1 - 2\%$ of the total amount of detections.

## 3.2 Networks with top-view LiDAR images as input

In this section we will present the first kind of networks that were developed in this project. The input to these networks consists of LiDAR sweeps converted to 2D top-view image, as described in Section 3.2.1. The network architectures differs between the networks but they have in common that they contain multiple 2D-convolutional layers and fully connected layers.

### 3.2.1 Creating training data

The training samples were created on the go in the training loop, and each sample consisted of LiDAR sweeps of the current surroundings and a cut-out from the map at the vehicle's global location. The sweeps were rotated and translated slightly to introduce a rigid transformation relative to the map cut-out. The rigid transformation was used as the label for the training sample. The transformation for each sample was drawn from a uniform distribution, allowing the network to train on similar, but different, training samples between the epochs. In more detail, 5 subsequent LiDAR sweeps were loaded and overlayed, creating a dense point cloud of the vehicle's surroundings. The next step was to discretize the point cloud into a grid structure covering an area of $30 \times 30$ m around the vehicle, where each cell of the grid represented one pixel in the top-view image. The spatial resolution of the discretized point cloud was 0.1 m or 0.05 m yielding grid structures of sizes $300 \times 300$ pixels or $600 \times 600$ pixels respectively. Each detection was assigned to its corresponding cell depending on its coordinates. The value of each cell was the number of detections in that area. The cutout was created by collecting all detections in the area of the sweep, by reading the previously mentioned csv-files in Section 3.1.1 that covered the specific area of the town. The detections in the map make up a very dense point cloud, we therefore decided to sample twice as many detections from the map as there were detections in the sweeps. These detections were then discretized into an image, as described above when discretizing the sweep.

Next, the sweep and cut-out were concatenated into an array with size $2 \times 300 \times 300$ pixels for the case when the spatial resolution was 0.1 m and $2 \times 600 \times 600$ pixels when 0.05 m was used. Each element was then normalized to a value between 0 and 1, according to the highest number of detections in the whole layer. The normalization was done to help the network to find similarities between the sweep and the cut-out. We assumed that the structures that got most detections in the map would be the same structures that yielded most detections the sweep. Therefore, a normalization with respect to the number of detections should yield similar values in the cutout and sweep for the same structures. Instead of normalizing this way, occupancy grids can be applied. All cells that contains at least 1 detection will then be assigned the value 1, yielding training samples consisting of only ones and zeros. This variant of training data will also be used in the thesis project. An example of an occupancy grid training sample is visualized in Figure 3.3.

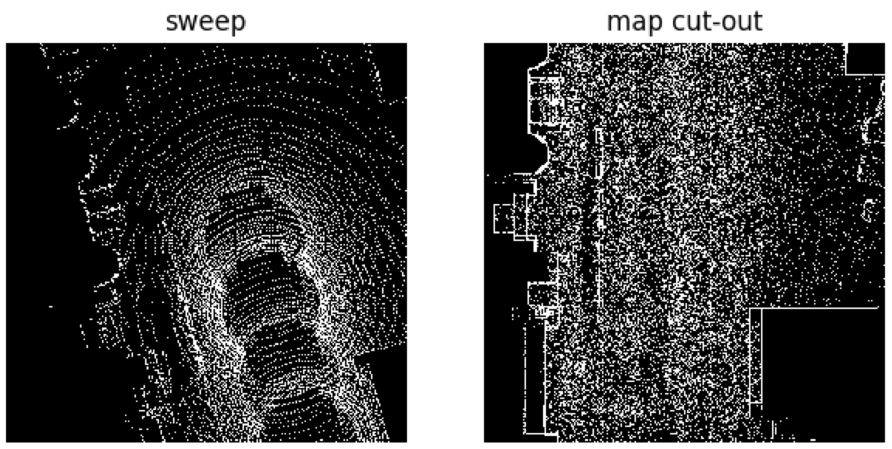


**Figure 3.3:** Example of rotated and translated sweeps and a cut-out from the map as a Bird's Eye View image. Each pixel represents a square region of 0.1 m. This sample is created using occupancy grid, where the pixel values are either 0 or 1. Contours of the vehicle can be seen in the sweeps, and contours of buildings in the surroundings can be seen in both sweep and cut-out. The rigid transformation for this specific sample is $[x, y, \theta] = [5, 5, 15]$ in metres and degrees respectively. This transformation is large compared to the ones used when training the networks, and only used here for visualization purposes.

We also used training data with extra information in the input, besides the number of detections in each pixel. One type of input consisted of an extra layer with the point cloud's mean height in each cell, as well as the original layers with the number of detections. After the sweep and cut-out were concatenated the height layers were normalized with the largest value found in both the sweep and cut-out, such that structures of the same height get similar pixel values both in the sweep and in the cut-out. Another type of input we trained the networks with, was the usual training samples with number of detections, with two extra channels containing information about the pixel coordinates. These kind of layers are called CoordConv [15]. The first layer represented the x-values and was filled with constant values where the first row consisted of zeros, the second row of ones etc. The second layer represented the y-values and was similar but with columns filled in the same way. The CoordConv layers were then normalized to values in the range $[0, 1]$. The purpose of CoordConv is to let the kernels in the convolutional layers know where in the image they are operating.

We believed that additional information, such as height and coordinates, may help the network to find similarities between the sweeps and the cut-out.

#### Simple training samples

Before using the training data that consisted of a concatenated sweep and map cut-out, we created simple training examples. This was done to quickly get started with the development and training of our first network. The main idea was to see at the start of the project if the networks can solve this kind of localization task at all. The simple training examples were created in the same way as the training examples in Section 3.2.1, but instead consisted of a sweep concatenated with the same sweep somewhat translated. The translation was drawn from a uniform distribution. This reduces the localization task of scan matching two similar but not identical LiDAR images, to a simpler task of finding the displacement of two identical images.

### 3.2.2 Network architectures

In this section the networks that were developed for 2D LiDAR images as input will be presented. They were named after the order they were created; Network 1, 2, 3 and 4 respectively. Each network has a few variations depending on parameters such as spatial resolution. All the networks have the same output which is a vector containing the translation and rotation angle, $[x, y, \theta]$. Since the network utilizes regression, the output is a vector with continuous values and no activation function is used in the output layer. For regularization, Batch Norm (BN) and drop out with $p = 0.2$, are used in all networks.

The architecture of Network 1 was inspired by the network presented by Caltagirone et al. in [28] where the purpose of the network was semantic segmentation of 2D LiDAR images. The input consists of either training samples of a sweep and map cut-out as described in Section 3.2.1, or the simple training data as described in Section 3.2.1. The input is processed by multiple convolutional layers, then passed to multiple fully connected layers which finally generates the output vector. The ReLU activation function is used in the first seven convolutional layers, and tanh is used in the following convolutional layers and the two first fully connected layers. The complete network architecture can be seen in Figure 3.4.

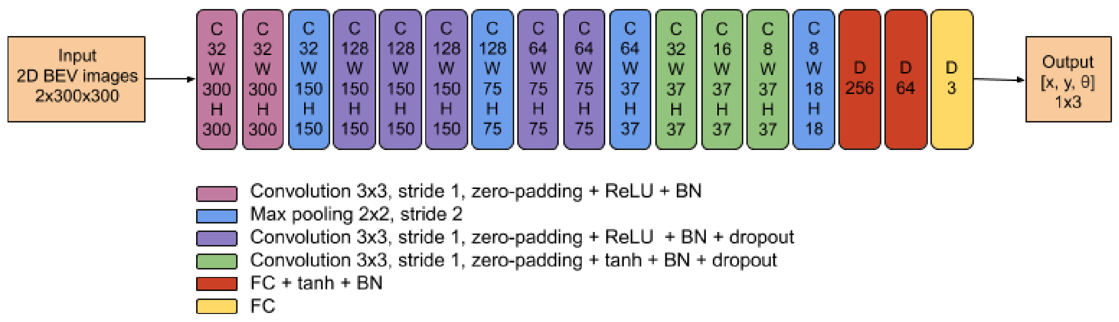


**Figure 3.4:** Visualization of the architecture of Network 1. The input consists of two concatenated top-view images of LiDAR measurements. The input is first passed through convolutional layers and finally propagated through three fully connected layers. The color coding describes the settings for each layer.

Network 2 was developed from Network 1. Unlike Network 1, Network 2 takes the sweep and the cut-out as two separate inputs through three convolutional layers with different weights. The kernels are of size $3 \times 3$ with stride one and zero padding. The outputs from the convolutional layers are two feature maps. These feature maps are concatenated and passed through multiple convolutional layers and finally three fully connected layers to yield the output vector. The ReLU activation function is used in all layers except the three last convolutional layers and the two first fully connected layers, where the tanh function is used. A schematic of Network 2 is shown in Figure 3.5.

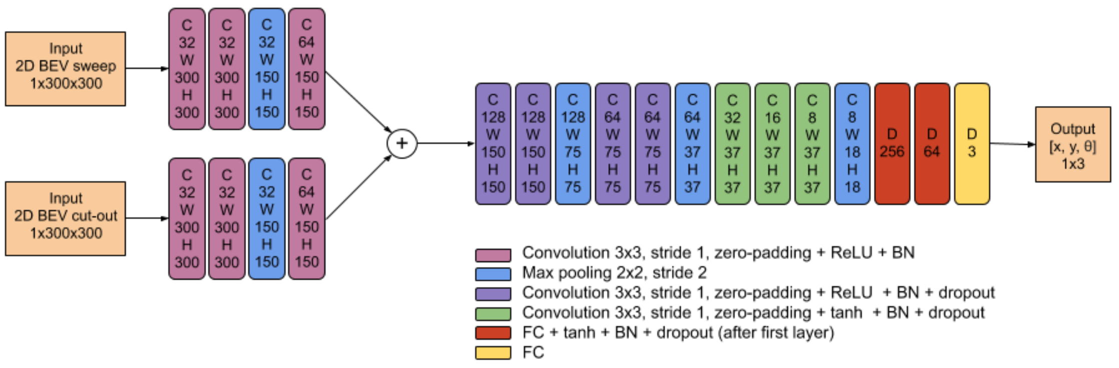


**Figure 3.5:** Schematic of Network 2 where the sweep and the cut-out are processed as two separate inputs through three convolutional layers before being concatenated and passed through multiple convolutional layers and fully connected layers. The color coding describes the settings for each layer.

Network 3 was inspired by the network FlowNetSimple presented in [13]. The input is fed through multiple convolutional layers. The first layers have a larger kernel size to increase the receptive field and the last 5 layers are fully connected layers. In Figure 3.6 a schematic is shown of the network used for images with a spatial resolution of 0.1 m. A similar version of the network adapted for inputs with a spatial resolution of 0.05 m can be found in Appendix A in Figure A.1. When the network was trained with inputs containing layers with height information, the same architecture was used with the only difference that the input layer has 4 inputs channels instead of 2. To investigate the importance of the receptive field in relation to the grids spatial resolution, an architecture for a spatial resolution of 0.05 m was developed with larger kernels, to get an equal receptive field in terms of meters comparable with Network 3 for a spatial resolution of 0.1 m. This architecture can be found in Appendix A, Figure A.2.

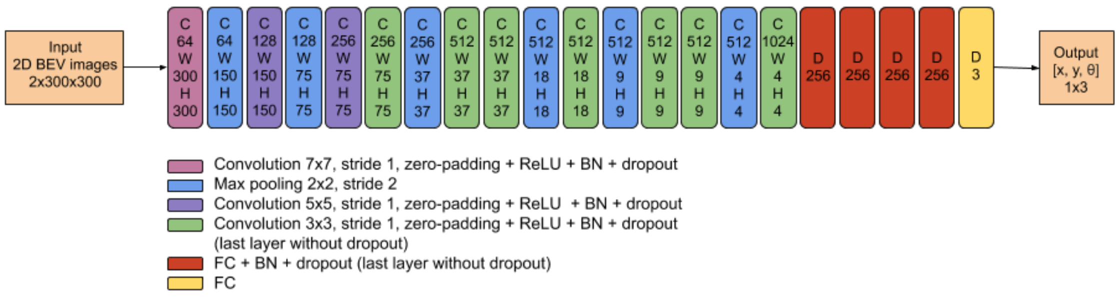


**Figure 3.6:** A schematic of Network 3 with 2 concatenated 2D LiDAR images as input. This architecture is used for spatial resolution 0.1 m. The color coding describes the different features of the layers.

We wanted to see if the a change of the activation function made a difference on the performance and therefore changed the activation function in the last three convolutional layers from ReLU to the tanh activation function. Tanh was also added as activation function in the two first fully connected layers. The modified architecture of Network 3 is shown in Figure A.4 found in Appendix A.

We also modified Network 3 to handle two separate inputs, similar to Network 2. The network was named Network 4. Each input is fed separately through the three first convolutional layers in Figure 3.6 before being concatenated and fed through rest of the network. The same weights are used to process the separate inputs. The architecture can be found in Appendix A in Figure A.3.

The last network developed was named Network 5 and was inspired by LinkNet [1]. The sweep and cut-out are processed separately by a LinkNet module and then concatenated and processed together by convolutional and fully connected layers. An overview of the network is presented in Figure 3.7 and the LinkNet module is visualized in Figure 3.8.

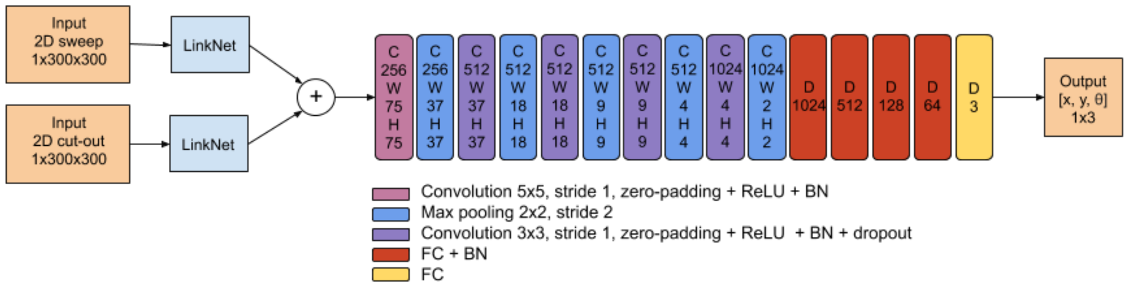


**Figure 3.7:** The architecture of Network 5, that consists of LinkNet modules followed by convolutional and fully connected layers. This architecture is developed for a spatial resolution of 0.1 m.

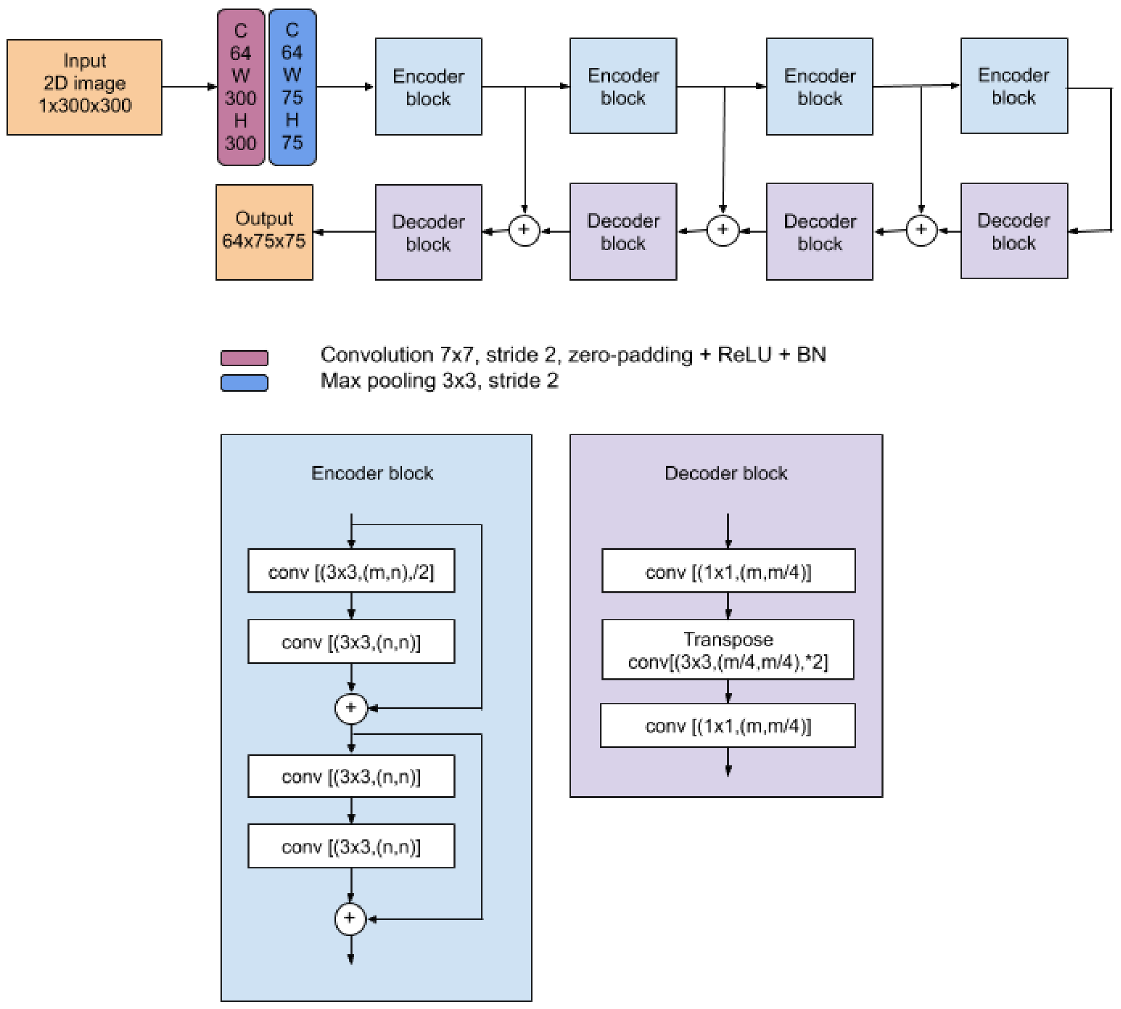


**Figure 3.8:** A visualization of LinkNet [1] that is used in Network 5. The input is first processed by a convolutional layer with stride 2 followed by a max-pooling operation. The Encoder block and the Decoder block are explained in more detail below the overall architecture in the image. In the encoders' and decoders' convolutional layers, the kernel sizes are $(3 \times 3)$ or $(1 \times 1)$ and the number of input and output channels are given by the second pair of parameters, e.g. $(n, m)$. Furthermore, /2 means convolution with stride 2, and *2 means upsampling by a factor 2. The size of the input decreases for each encoder block, and increases again with each decoder block. Feature maps from higher resolutions are added along the way between the decoders.

### 3.2.3 Training

When training Network 1, we started with the simple training data to investigate if the network had the capacity to solve a similar but simpler localization problem. When choosing which optimizer and loss function we wanted to use we tested two different optimizers; Stochastic Gradient Decent (SGD) and Adam in combination with the loss functions Mean Squared Error (MSE) and SmoothL1. This was done to see which combination that performed best. Finally the Adam optimizer in combination with the SmoothL1 loss function was used. Next, we trained Network 1 using the training samples that consisted of the sweep and the cutout from the map and used the Adam optimizer. We developed a customized loss function which weighted the translation loss and rotation loss differently. The total loss was defined as,

$$L_{tot} = (\hat{y}, y) = \alpha L_{trans}(\hat{y}, y) + \beta L_{rot}(\hat{y}, y)$$

where $L_{trans}$ is the translation loss and $L_{rot}$ is the rotation loss. The parameters $\alpha$ and $\beta$ can be adjusted to weight the importance of the different losses. SmoothL1 was used as loss function for both the translation and rotation errors.

All networks were first trained on pure translation in the range $[-1, 1]$ m with no rotation. The networks were then tested on a test set with the same attributes. Each network training was then resumed, again with a translation in the range $[-1, 1]$ m, but this time also with a rotation error in the range $[-2, 2]$ degrees. The purpose of this was to investigate if the combined translation and rotation error was harder to predict than just pure translation. When training on pure translation, the loss weights of the loss function above was set to $\alpha = 0.9, \beta = 0.1$. When introducing the rotation, the loss weights were set to $\alpha = \beta = 0.5$ to encourage the network to prioritize the rotation error. The translation and rotation up to 1 m and 2 degrees respectively were chosen since they correspond to approximately up to 10 pixels error between the sweep and the map cutout, and hence of the same magnitude.

The batch size varied when training the different networks, with the goal of fully utilizing the memory on each GPU. The batch size ranged from 50-170 samples, distributed on 4 GPUs. All networks were trained with a learning rate scheduler that decreased the learning rate with a factor 0.1 if the validation loss had not decreased the last 5 epochs. The initial learning rate was set to 0.01. Early stopping was used, with a patience of 10 epochs, allowing the learning rate to decrease once before interrupting the training. The training data set consisted of 3692 samples, the validation set of 923 samples and the test set of 1077 samples. Table 3.1 presents a summary of all the networks that have been trained and tested, including different versions for spatial resolutions or other special attributes.

**Table 3.1:** This table summarizes all the different networks that have been trained and tested. The results will be presented in Chapter 4.

| Network number | Spatial resolution | Special attribute |
|---|---|---|
| Network 1 | 0.1 m | Simple training data |
| | 0.1 m | |
| | 0.1 m | Different activation function |
| Network 2 | 0.1 m | |
| Network 3 | 0.1 m | |
| | 0.1 m | Occupancy grid |
| | 0.1 m | CoordConv |
| | 0.1 m | Height layers |
| | 0.1 m | Different activation function |
| | 0.05 m | |
| | 0.05 m | Larger kernels |
| Network 4 | 0.1 m | |
| | 0.05 m | |
| Network 5 | 0.1 m | |

## 3.3 Networks with LiDAR point clouds as input

A part of this project consists of developing a network than can perform the localization task using point clouds as input. A network of this type, taking raw point clouds as input, will then learn to extract the necessary features to perform localization. In the former case, using 2D images as input, the network was fed handcrafted features, e.g. the number of detections in each pixel. This section will begin with an explanation of how the training data was created, followed by a section that describes the network architectures that were used. Lastly, an overview of the training process is presented.

### 3.3.1 Creating training data

The training samples were created on the go in the training loop with rigid transformations drawn from uniform distributions, with the consequence that the network will never train on the exact same training sample between epochs. The method of creating the training samples is closely related to the first steps of creating the BEV-images for the networks using 2D images as input. First, 5 consecutive sweeps of the current surroundings were loaded and used as the first part of the input. This point cloud was rotated and translated to introduce an error in the alignment between the sweeps and the map. Secondly, to create a cut-out of the map, all detections in the region of the sweeps were loaded. The two point clouds, the current surrounding and the map cut-out, could then be pre-processed into a representation that was used as an input to the network. The following steps are inspired by the network PointPillars [12] created by Alex H. Lang et al., and is performed on the two parts of the input separately.

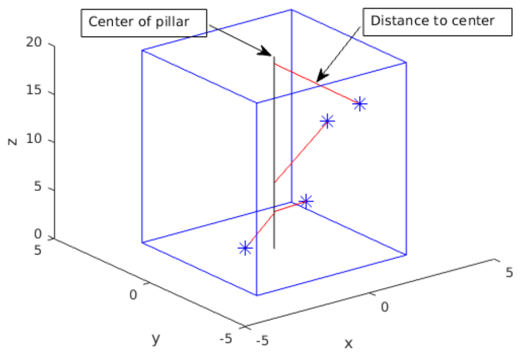


**Figure 3.9:** A visualization of the features that are calculated for each point in one pillar. In this case, there are 4 points marked as blue stars in the pillar, and the offset in $x$ and $y$ for each point is marked with red lines. The features for one point is $[x_p, y_p, z]$ where $x_p$ and $y_p$ are the calculated offsets, and $z$ is the height of the point.

The point cloud was discretized into to an evenly spaced grid in the $xy$-plane, creating a set of pillars, where each pillar is a separate point cloud. The spatial resolution of the grid was set to 0.5 m or 0.25 m. The offset in the $xy$-plane to the center of the pillar, $x_p, y_p$, was calculated for each point in each pillar and saved together with the corresponding $z$-coordinate. The pillars then consisted of a set of three features ($D$) for each point, namely the distance to the center and the height of that point, $[x_p, y_p, z]$ which is shown in Figure 3.9. A great amount of the pillars would turn out empty and to exploit the sparsity a limit was set on both number of non-empty pillars ($P$) and points per pillar ($N$). For the cases when the spatial resolution was 0.5 m the limit on the pillars was set to 1260 and maximum points in one pillar was set to 900 points. For the case when using a spatial resolution of 0.25 m the limit on the pillars was set to 5040 and maximum points in one pillar was set to 225 points. This yields a tensor of size $D \times P \times N$. If $P$ or $N$ exceeds the limit, the data is randomly sampled and if there are not enough pillars or points, zero padding is applied.

### 3.3.2 Network architectures

A network that handles point clouds as input must take into account that the data is an unordered set where the input order of the coordinates should not matter. Furthermore, points that are close in space have a geometrical meaning, which should also be modelled by the network. The network used in this thesis project relies on the first parts of Point Pillars, where point cloud pillars are passed trough a first part of the network that computes a feature tensor for each point cloud pillar. The first part is called a Pillar Feature Net (PFN) and will be described in more detail in the next section. The second part of the network that handles the feature tensors and outputs a rigid transformation is our own contribution, we call it the Backbone of the network. In Figure 3.10 an overview of the whole network is shown.

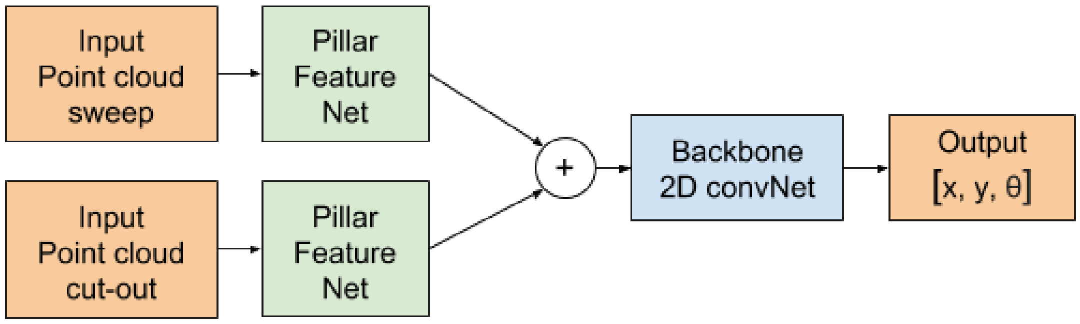


**Figure 3.10:** An overview of the networks that use point clouds as input. The green module is inspired by PointPillars, and processes the pillar point clouds to produce a pseudo-image. The pseudo-images are processed in the blue module.

#### Point cloud to Pseudo-Image

The input to the network consists of a tensor of the size $D \times P \times N$, where $D$ is the number of features for each point, $P$ the number of pillars and $N$ is the number of points in each pillar as described in Section 3.3.1. The main idea of the first part of the network, the Pillar Feature Net (PFN) layer, is to compute a feature tensor for each pillar and then create a pseudo-image where each pixel is the feature tensor of the pillar in that location. For each point a convolutional layer with kernel size 1 and $C = 64$ or $C = 32$ channels is applied followed by a Bach Norm and a ReLU layer yielding a tensor of size $C \times P \times N$. Lastly a max operator is applied over the channels resulting in a $C \times P$ tensor. The symmetric max operator function is used to aggregate the information from each point making the output tensor invariant to the input order. This tensor can be concatenated with the pillar point cloud input tensor, and processed again by multiple PFN-layers. To create the pseudo image, the output features are scattered back to the location of the original pillar. The pseudo-image is of size $C \times H \times W$ where $H$ represents the height and $W$ the width of the image. The architecture for the PFN layer is shown in Figure 3.11 and the process to convert a point cloud into a pseudo image is visualized in Figure 3.12. For the case when using multiple PFN-layers see Appendix A, Figure A.5.

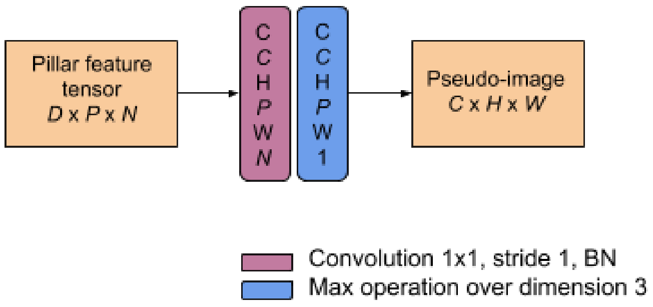


**Figure 3.11:** Schematic of a single PFN-layer which takes a pillar feature tensor as input and outputs a pseudo-image of the tensor.

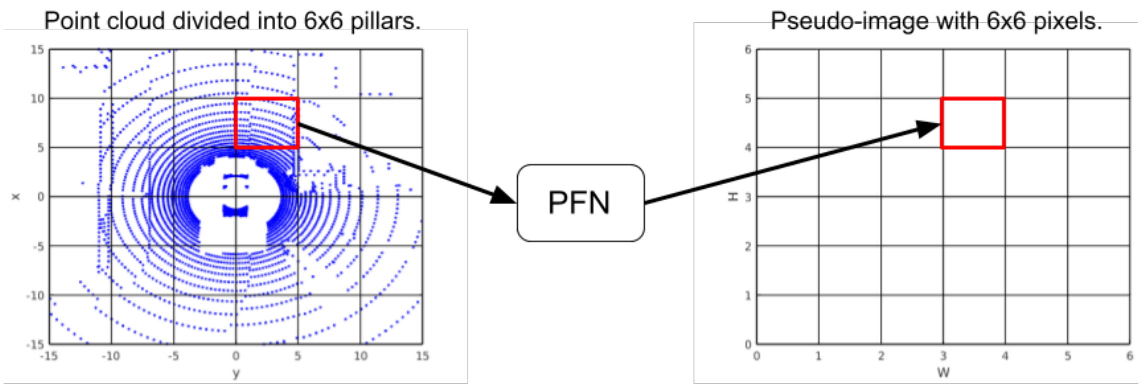

Point cloud divided into 6x6 pillars.
PFN
Pseudo-image with 6x6 pixels.
x
y
H
W

**Figure 3.12:** Visualization of the process to convert a point cloud into a pseudo-image. To the left is a point cloud, in this figure seen as a top-view image, which is divided into pillars. Each pillar point cloud is processed by a Pillar Feature Net (PFN), and the resulting feature tensor is saved in a tensor to the right, in the element corresponding to the original spatial location of the pillar. In this way a pseudo-image is created, with a feature tensor making up each pixel.

### Backbone architectures

The task of the Backbone architecture is to take the pseudo-images of both the sweep and the cutout as input, and output the rigid transformation that is needed to align the sweep with the map. The sweep and cut-out can either be concatenated before used as input or processed separately. The pseudo-images are of the dimensions $C \times H \times W$, where $H$ and $W$ represents the number of pillars spatially in $x$ and $y$ direction, and can be seen as an image where each tensor with $C$ elements represents one pixel. The idea is to use the same principles that were used in the network for 2D images, mainly to apply 2D-convolutional layers on the pseudo images and use regression to output a rigid transformation. Several architectures for the Backbone were developed; Backbone 1, 2, 3 and 4.

Backbone 1 was developed for two spatial resolutions; 0.5 m and 0.25 m. The input consisted of concatenated pseudo-images of the sweep and the cut-out. The number of features for the input was 64 in total for both sweep and map cut-out. In Figure 3.13 Backbone 1 for spatial resolution 0.5 m is presented. The similar architecture for spatial resolution 0.25 m is visualized in Figure A.6 in Appendix A.

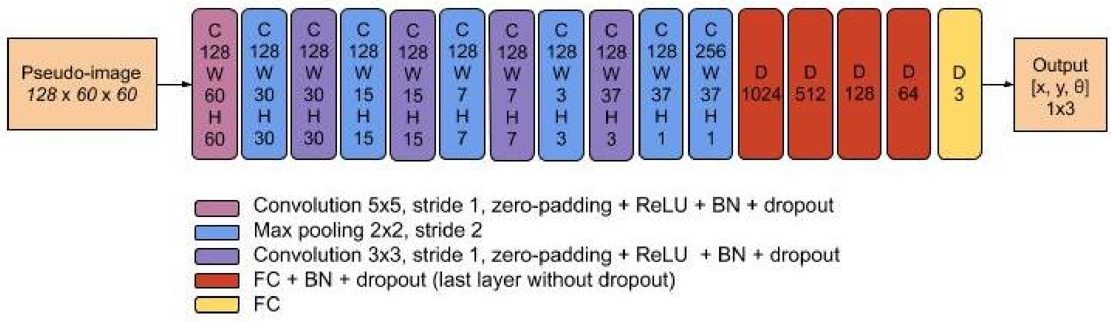


**Figure 3.13:** Schematic of Backbone 1 for spatial resolution 0.5 m. The color coding presents the different settings for each layer.

We developed the architecture for Backbone 2, which is shown Figure 3.14, from Backbone 1 by adding more convolutional layers and changing the number of channels in some layers. Backbone 2 was only developed for a spatial resolution of 0.5 m.

Unlike Backbone 1 and 2, Backbone 3 and 4 takes two inputs, one input is the pseudo image for the LiDAR sweep and the other input is the pseudo image for the cut-out. After passing multiple convolutional layers they are concatenated before sent through a stack of fully connected layers. Backbone 3 was developed for spatial resolution 0.5 m, whilst Backbone 4 also was developed for spatial resolution 0.25 m. The number of features of the pseudo images used in Backbone 3 was 32 or 64, Backbone 4 only used 32. Backbone 3 and 4 are shown in Figure 3.15 and Figure 3.16. Backbone 4 for spatial resolution 0.25 m is shown in Figure A.7 found in Appendix A.

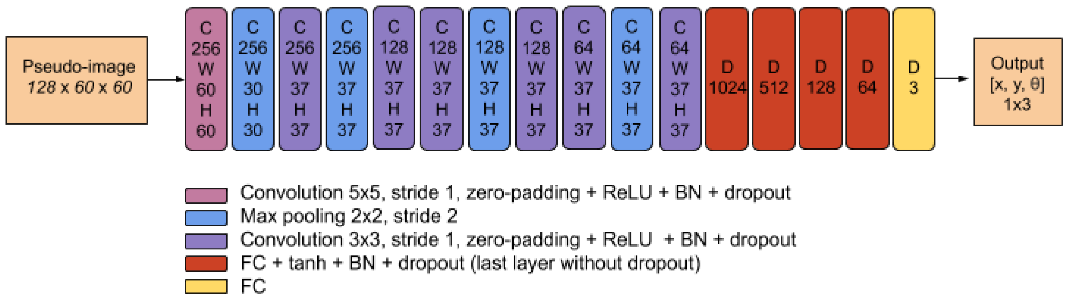


**Figure 3.14:** Schematic of Backbone 2, which was developed from Network 1, for spatial resolution 0.5 m. The color coding presents the different settings for each layer.

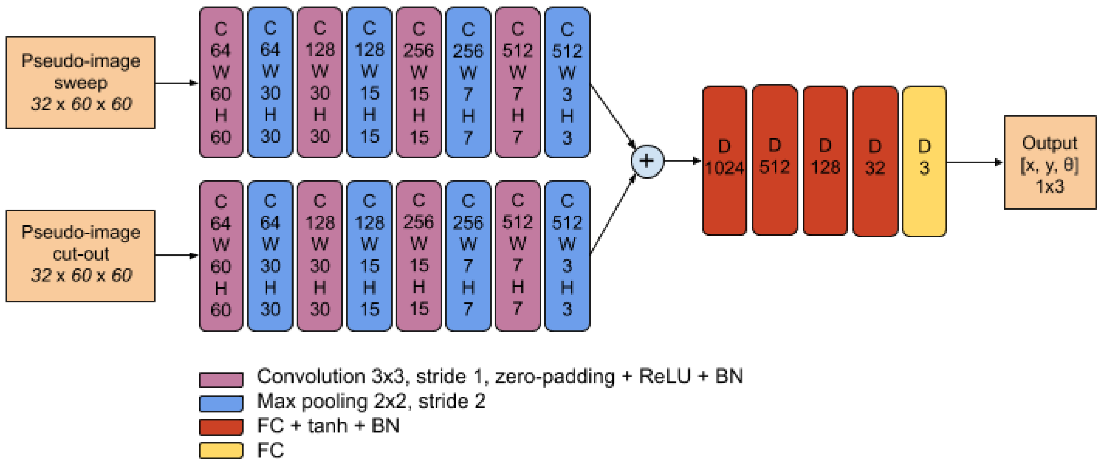


**Figure 3.15:** Backbone 3 with spatial resolution 0.5 m and 32 features from PFN-layer. Sweep and cutout pseudo-images are processed separately by convolutional layers before being concatenated and sent through fully connected layers.

### 3.3.3 Training

All networks were trained using the Adam optimizer and the customized loss described in Section 3.2.3 using the SmoothL1 loss function. When training on no rotation, the loss parameters were set to $\alpha = 0.9$ and $\beta = 0.1$. When training with rotation the parameters were instead set to $\alpha = \beta = 0.5$. When training on pure translation, the range used was $[-1.5, 1.5]$ m which corresponds to up to 3 and 6 pixels in the pseudo-image for spatial resolution 0.5 m and 0.25 m respectively. The rotation used was in the range $[-3, 3]$ degrees, which would correspond to up to 2 pixels in the pseudo-image for a spatial resolution 0.5 m, and up to 4 pixels for spatial resolution 0.25 m. All networks were trained with a learning rate scheduler that decreased the learning rate with a factor 0.1 if the validation loss had not decreased the last 5 epochs. The initial learning rate was set to 0.01. The batch size varied

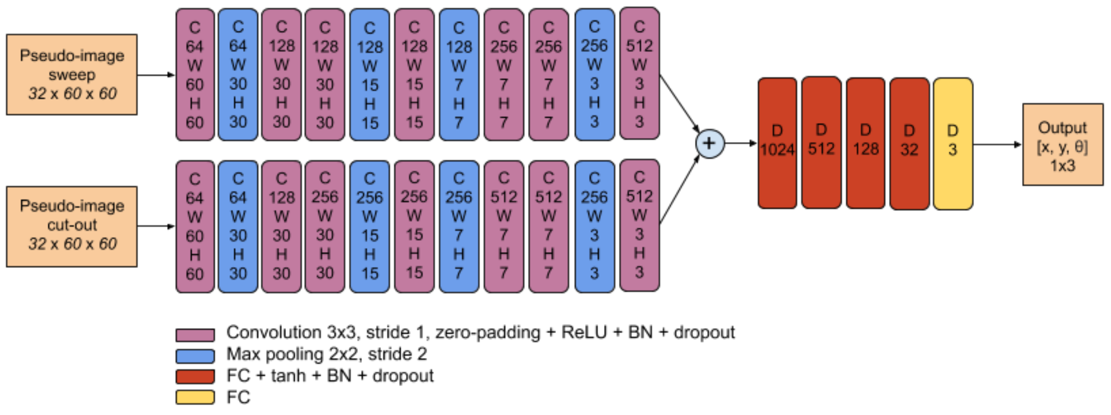


**Figure 3.16:** Backbone 4 with spatial resolution 0.5 m and 32 features from PFN-layer. Sweep and cutout pseudo-images are processed separately by convolutional layers before being concatenated and sent through fully connected layers.

when training the different networks, with the goal of fully utilizing the memory on each GPU. The batch size ranged from 8-34 samples, distributed on 4 GPUs. Early stopping was used, with a patience of 10 epochs, allowing the learning rate to decrease once before interrupting the training. The training data set consisted of 3692 samples, the validation set of 923 samples and the test set of 1077 samples.

Different combinations of networks were trained, with variations in the pillar size, which of the Backbones that were used, the number of features in the PFN-layer and also how many PFN-layers to use. Table 3.2 presents a summary of all the networks that have been trained and tested, including different versions for spatial resolution or other special attributes.

**Table 3.2:** This table summarizes all the different networks that have been trained and tested. The Backbones are described in this chapter, and the results will be presented in the next chapter.

| Backbone number | Spatial resolution | Number of features from PFN-layer |
|---|---|---|
| Backbone 1 | 0.5 m | 32 |
| | 0.25 m | 32 |
| | 0.25 m | 64 |
| | 0.25 m | 64, 64 |
| Backbone 2 | 0.5 m | 64 |
| Backbone 3 | 0.5 m | 32 |
| Backbone 4 | 0.5 m | 32 |
| | 0.25 m | 32 |

# 4
# Results

The performances of the different networks are presented in this chapter starting with the networks developed for using top-view LiDAR images as input, followed by the results when using LiDAR point clouds as input. The chapter commences with some brief overview of how the results are presented and which metrics that indicate a good performance. The section with results for networks with top-view images as input, starts with the results from an initial study of the localization task using simple training data. Some brief discussions and comparisons are included in this chapter when presenting the results.

## 4.1 Networks with top-view LiDAR images as input

In this section we will present the results from training the networks that takes 2D LiDAR images as input from Section 3.2. First the results from training Network 1 on simple training data will be presented as an initial study of the localization task. Then the results when training Network 1, Network 2, Network 3 and Network 4 on training data consisting of a sweep and a cut-out from the map are presented. All the networks were tested using the SmoothL1 loss function with equal weighting for the error in translation $x$ and $y$ and angle $\theta$, independently of the loss used when training to be able to compare the networks' performances.

The test results are presented in tables and figures with histograms. The tables contain important metrics which are used to compare the performance of the networks. These metrics are; test loss, median translation prediction error and median rotation prediction error. The prediction error is calculated by subtracting the true value from the predicted value. The test loss is calculated by taking the total test loss and divide it with the number of samples, which will enable a comparison between the different test losses since the data sets were not always equal in size depending on batch size. The network with the lowest test loss is considered to be the one that performs the best. Note that the median translation prediction error that is presented in the tables is the total translation error, i.e the hypotenuse between the ground truth and the prediction. The median translation and rotation errors are shown to give a feeling about how well the network performs, however a small error does not always automatically mean that the network generalizes well on the test set. Uniformly distributed prediction errors indicate that the network performs poorly. In these cases, histograms of the distribution of the prediction errors are presented. We expect the prediction error distribution to be narrowly centered around zero for a well performing network.

### 4.1.1 Initial studies using simple training data

The network, named Network 1 which is described in Section 3.2.2, was first trained on simple training data to investigate if the network had the capability to solve an easier localization problem before training on more realistic data. The batch size was set to 170. The network was first trained on pure translation in the range $[-1, 1]$ m, and then tested on Test sets 1-3. Then it was trained from scratch on pure rotation in the range $[-2, 2]$ degrees and tested on Test set 4. The Test sets are presented below:

- Test set 1: translation in the range $[-1, 1]$ m, to see how the network handles translation errors that it has been trained on.
- Test set 2: translation in the range $[-3, 3]$ m with no rotation, to see how the network handles unseen large translation errors.
- Test set 3: translation in the range $[-1, 1]$ m with rotation in the range $[-2, 2]$ degrees, to see how the network performs on unseen rotation errors.
- Test set 4: rotation in the range $[-2, 2]$ degrees, to see if the network has potential to predict a rotation error.

After being trained on pure translation and tested on Test sets 1-3, the weights of the network were then trained another round on simple training data with the same translation error in the range $[-1, 1]$ m and with a rotation error in the range $[-2, 2]$ degrees. The network was then tested again on Test set 3, to confirm that the performance improved if trained on similar training data with rotation errors.

Tables 4.1 and 4.2 present the results with metrics in the form of the test loss, the median translation prediction error in the $xy$-plane and the median rotation prediction error. In Table 4.1 it can be seen that when the network has been trained on a translation error in the range $[-1, 1]$ m the network can predict the rigid transformation with a median translation error of 17 cm. It can to some extent also correct larger translation errors in the range $[-3, 3]$ m as seen in Test set 2. Figure 4.1 indicates that Network 1 is indeed able to recognize pure translation between two images, since the prediction error spans a smaller range than the introduced error in the training samples. When tested on Test set 3 that has training data with rotation, the performance degrades. Test set 4 does however show that the network can predict pure rotation with a relatively good accuracy.

The network was also trained on rotation and tested again on Test set 3. The result can be seen in Table 4.2. This result can be further explored together with Figure 4.2, showing the prediction errors before and after training on rotation. Even though the test loss and median translation error increase when trained on rotation, the histogram indicates that the network have learned to identify and correct rotation, since the width of the histogram for the rotation angle decreases after training on rotation.

| | test loss (1e-4) | median translation prediction error | median rotation prediction error |
|---|---|---|---|
| Test set 1 | 0.3832 | 0.17 m | 0.0009 ° |
| Test set 2 | 12.6510 | 0.93 m | -0.0028 ° |
| Test set 3 | 11.6242 | 0.16 m | -0.1864 ° |
| Test set 4 | 7.4441 | 0.01 m | -0.0072 ° |

**Table 4.1:** Table with test loss as well as median prediction errors for translation and rotation. The network the was first trained on pure translation and then tested on Test set 1-3. Then it was trained from scratch on pure rotation and tested on Test set 4.

| | test loss (1e-4) | median translation predicted error | median rotation predicted error |
|---|---|---|---|
| Test set 3 | 12.3329 | 0.25 m | -0.0201 ° |

**Table 4.2:** Table with test loss as well as the median translation and rotation error for Test set 3. The network is trained on both translation and rotation.

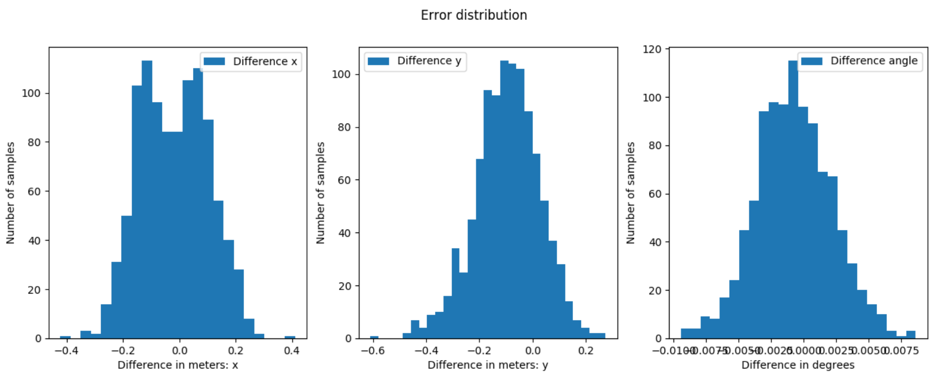
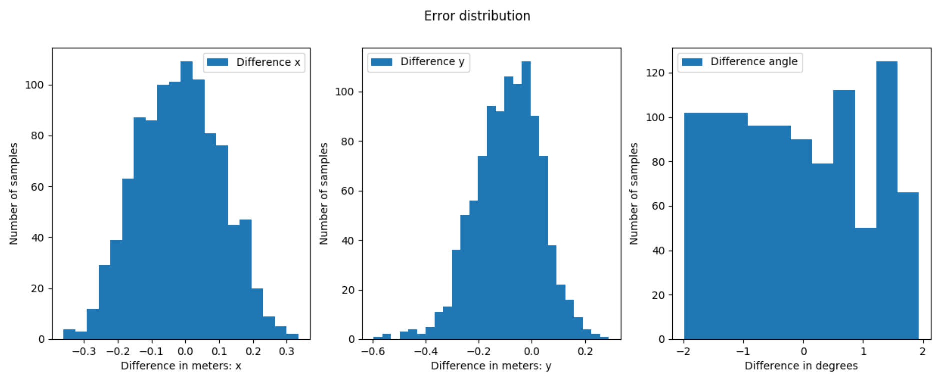
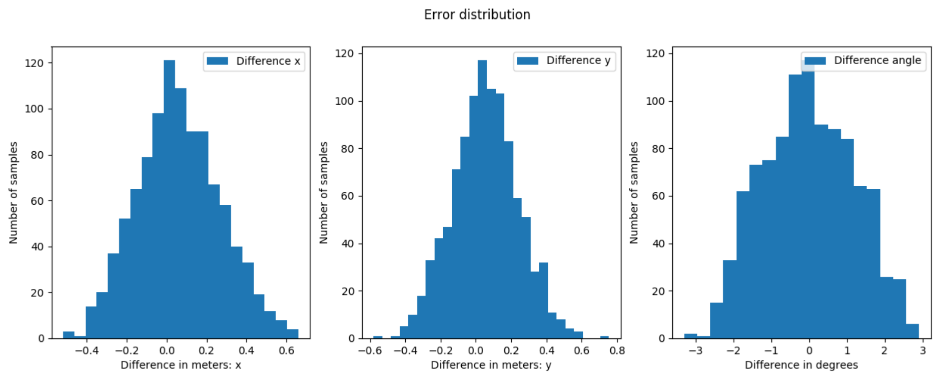


**Figure 4.1:** Histograms showing the test results for Network 1 trained on translation in the range $[-1, 1]$ m with no rotation and tested in Test set 1. The histograms represents the prediction error for the translation in $x, y$ in metres and the prediction error for the rotation in degrees.

A quick conclusion is that the localization task should indeed be possible to solve using neural network regression, since Network 1 can at least minimize the translation to a median error of 17 cm which corresponds to the adjacent pixel or the pixel next to that from the ground truth pixel. We can also conclude from the metrics in Tables 4.1 and 4.2 that rotation seems to be a more difficult problem for the neural network than pure translation, since the test loss ended at a higher value when introducing a rotation error. The network's performance on predicting the rotation angle improved, at the cost of decreased accuracy for translation errors.



**(a)** Network 1 trained on pure translation, but tested on rotation.



**(b)** Network 1 trained and tested on translation and rotation.

**Figure 4.2:** Results for Network 1 on simple training data. These histograms show the prediction error in $x$, $y$, and rotation angle $\theta$. Subfigure b) shows that the network can predict the rotation with more accuracy after it has been exposed to training samples with rotation errors. The error in translation does however increase.

### 4.1.2 Network 1 and Network 2

In this section, the results when training Network 1 and 2 will be presented. The training data consisted of a sweep concatenated with a cut-out of the map at the initial guess of the position and heading of the vehicle. The batch size was set to 170 for Network 1 and 128 for Network 2. A reminder of the main difference between Network 1 and Network 2, is that the sweep and the map cut-out are processed separately before being concatenated after three convolution layers in Network 2. The architectures are the same after these first layers. The training parameters that were used are described in Section 3.2.3.

The network was initially trained on pure translation in the range $[-1, 1]$ m, and thereafter trained on rotation in the range $[-2, 2]$ degrees as well. We also wanted to see if the choice of activation function had any impact on the performance of Network 1. Therefore we removed the tanh activation function and trained the network again using the same setup as before. The results from training and testing the networks on pure translation are presented in Table 4.3. The results from the resumed training and testing on rotation is presented in Table 4.4.

**Table 4.3:** Table presenting the performance of Network 1, Network 2 and Network 1 without tanh on a test set with a translation error in the range $[-1, 1]$ m and no rotation.

| Network name | test loss (1e-4) | median translation prediction error | median rotation prediction error |
|---|---|---|---|
| Network 1 | 0.6128 | 0.19 m | 0.0005 ° |
| Network 2 | 0.9116 | 0.22 m | -0.0010 ° |
| Network 1 without tanh | 1.2802 | 0.27 m | -0.0006 ° |

**Table 4.4:** Table presenting the performance of Network 1, Network 2 and Network 1 without tanh on a test set with a translation error in the range $[-1, 1]$ m and rotation error in the range $[-2, 2]$ degrees.

| Network name | test loss (1e-4) | median translation prediction error | median rotation prediction error |
|---|---|---|---|
| Network 1 | 7.5300 | 0.21 m | 0.0347 ° |
| Network 2 | 10.2021 | 0.22 m | 0.0955 ° |
| Network 1 without tanh | 11.7482 | 0.32 m | 0.0051 ° |

Network 1's test loss increased from 0.6128e-4 to 7.5300e-4 when adding the rotation error, this shows that the rotation angle is harder for the network to predict than pure translation. The results for Network 2 follows the same trend with an increase of the loss. By comparing the test loss between Network 1 and Network 2 in Tables 4.3 and 4.4 it can be seen that Network 1 performs better than Network 2 on both test sets, e.g. both on pure translation as well as rotation. The conclusion from this is that the network architecture of Network 1 is more suitable for this task compared to Network 2. It can also be seen in the tables that the test loss for Network 1 without the usage of the tanh activation is higher than the loss for regular Network 1. This indicates that the tanh activation function is suitable to use in Network 1.

### 4.1.3 Network 3

Since Network 3 performs significantly better than the earlier networks, as will be clear in this section, many different combinations and ideas were tested with this architecture as a starting point. Network 3 is designed differently compared to Network 1 and 2, with an increasing of the capacity of the model to solve the localization task with higher accuracy. The network was trained using different set ups with special attributes as described below,

- Spatial resolution 0.1 m: using grids with a spatial resolution of 0.1 m.
- With tanh: using grids with a spatial resolution of 0.1 m and tanh as activation function in some layers.
- Occupancy grid: using occupancy grid instead of number of detections as input feature, with grids with a spatial resolution of 0.1 m.
- CoordConv: adding coordinate layers to the input as described in Section 3.2.1, using grids with a spatial resolution of 0.1 m.
- Height: adding height-layers to the input as described in Section 3.2.1, using grids with a spatial resolution of 0.1 m.
- Spatial resolution 0.05 m: using grids with a spatial resolution of 0.05 m.
- Larger kernels: using grids with a spatial resolution of 0.05 m with larger convolution kernels to increase the receptive field.

All versions were first trained and tested on pure translation, and then trained again on rotation, in the same way as for Network 1 and 2. The results for pure translation can be seen in Table 4.5, and with rotation in Table 4.6.

**Table 4.5:** Performance of Network 3 trained on different set of parameters. All networks were trained on translation in the range $[-1, 1]$ m with no rotation. Each trained network model was then tested on a test set with the same set of parameters as the training set it was trained on. The best performance is marked in bold text.

| Special attribute | test loss (1e-4) | median translation prediction error | median rotation prediction error |
|---|---|---|---|
| Spatial resolution 0.1 m | 0.1161 | 0.08 m | **0.0000 °** |
| With tanh | **0.0366** | **0.05 m** | -0.0001° |
| Occupancy grid | 0.6236 | 0.18 m | 0.0001 ° |
| CoordConv | 12.9339 | 0.79 m | 0.0013 ° |
| Height | 13.3525 | 0.80 m | 0.0015 ° |
| Spatial resolution 0.05 m | 0.4403 | 0.08 m | -0.0015 ° |
| Larger kernels | 16.3682 | 0.52 m | 0.0027° |

**Table 4.6:** Performance of Network 3 trained on different set of parameters. All networks were trained on translation in the range $[-1, 1]$ m with rotation $[-2, 2]$ degrees. Each trained network model was then tested on a test set with the same set of parameters as the training set it was trained on. The best performance is marked in bold text.

| Special attribute | test loss (1e-4) | median translation prediction error | median rotation prediction error |
|---|---|---|---|
| Spatial resolution 0.1 m | 0.4236 | 0.07 m | **-0.0149 °** |
| With tanh | **0.4183** | **0.05 m** | 0.0151° |
| Occupancy grid | 8.4105 | 0.29 m | 0.0813 ° |
| CoordConv | 35.6720 | 0.80 m | 0.0734 ° |
| Height | 36.5592 | 0.80 m | 0.0137 ° |
| Spatial resolution 0.05 m | 2.5242 | 0.14 m | -0.0440 ° |
| Larger kernels | 22.8829 | 0.50 m | 0.1076° |

The test loss is lowest for Network 3 with a spatial resolution of 0.1 m with the activation tanh, in both Tables 4.5 and 4.6 for pure translation and with rotation respectively. We can see that the choice of activation function improved the performance slightly. Instead of not using any activation functions in the fully connected layers, as is the case in the regular Network 3 architecture, the two first fully connected layer uses the tanh activation function. The tanh is also used as activation function in the last three convolutional layers. The network with CoordConv and with height features perform noticeably poor for cases both with and without rotation errors. In Figure 4.3 it can be seen that the prediction errors for all output parameters for the CoordConv network are uniformly distributed, indicating that the training has not improved the performance at all.

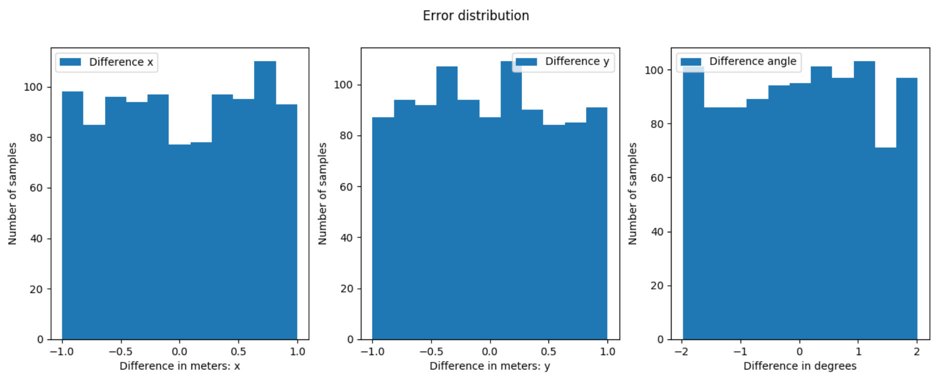


**Figure 4.3:** Histograms showing the distribution of the translation prediction error in $x$ and $y$ as well as the rotation angle for Network 3 with coordinate layers. The network was trained on a translation error in the range $[-1, 1]$ m and rotation error in $[-2, 2]$ degrees. The prediction errors seem to be uniformly distributed, in the same ranges as the introduced errors in the training samples, which indicates that the training has not improved the network's performance at all.

As can be seen in Tables 4.5 and 4.6, the performance did not improve when decreasing the spatial resolution to 0.05 m, even though the test loss was still in comparable size with the best performing network. The same architecture was used again, but with larger kernels to increase the receptive field of the network, but this unexpectedly severely degraded the performance.

Network 3 with tanh, using grids with spatial resolution 0.1 m without any extra attributes such as occupancy grids, performed well with a low test loss compared to the others. One final test was executed on Network 3 with tanh trained on both translation and rotation. It was tested on a test set with no errors at all, i.e. the sweep and the map cut-out are perfectly aligned. The test yielded a test loss of 0.2124e-4, a median translation prediction error of 0.03 m and median rotation prediction error of 0.0006 °, where the test loss is slightly larger and the median error is slightly smaller than the metrics in Table 4.6. This indicates that the network can correct large rigid transformations accurately, but fail to recognize small misalignments, since it still predicts small transformations even when the test samples are perfectly aligned.

### 4.1.4 Network 4

Network 4 is similar to Network 3, with the difference that it processes the sweep and the map cut-out as two separate inputs. Network 4 was first trained and tested using pure translation in $[-1, 1]$ m, with the results presented in Table 4.7 for grids with a spatial resolution of 0.1 m and 0.05 m. The network was then trained and tested on data with translation in the range $[-1, 1]$ m as well as rotation in the range $[-2, 2]$ degrees. The results are presented in Table 4.8.

**Table 4.7:** Performance of Network 4 trained and tested on pure translation without rotation. The translation error was in the range $[-1, 1]$ m.

| Spatial resolution | test loss (1e-4) | median translation prediction error | median rotation prediction error |
|---|---|---|---|
| 0.1 m | 0.1153 | 0.06 m | 0.0000 ° |
| 0.05 m | 1.3801 | 0.18 m | 0.0000 ° |

**Table 4.8:** Performance of Network 4 trained and tested on translation and rotation. The translation error was in the range $[-1, 1]$ m and the rotation in the range $[-2, 2]$ degrees.

| Spatial resolution | test loss (1e-4) | median translation prediction error | median rotation prediction error |
|---|---|---|---|
| 0.1 m | 0.6568 | 0.10 m | -0.0388 ° |
| 0.05 m | 5.1144 | 0.17 m | 0.0198 ° |

Comparing these results with Network 3's performance, we see that Network 4 does not outperform Network 3. All error metrics are higher for Network 4 except the median translation prediction error when using pure rotation which is the same for the two networks.

### 4.1.5 Network 5

Network 5 has a different network architecture compared to the rest, and its performance is presented in Table 4.9. We can see that the results are comparable with the best performing Network 3, which indicate that this architecture is also suitable to solve the localization task.

**Table 4.9:** Performance of Network 5 first trained on pure rotation in the range range $[-1, 1]$ m and then with a rotation in the range $[-2, 2]$ degrees. The network was then tested on a test set with the same set of parameters as the training set it was trained on.

| Rotation | test loss (1e-4) | median translation prediction error | median rotation prediction error |
|---|---|---|---|
| No | 0.0543 | 0.06 m | -0.0001 ° |
| Yes | 0.4038 | 0.08m | 0.0257 ° |

### 4.1.6 Summary of the networks' performances

In Figure 4.4, all our networks that uses 2D top-view images as input are presented. The blue markers show the results of the networks that were trained and tested on pure translation and the red markers show the networks that were trained and tested on both translation and rotation. The dotted line represents the spatial resolution of the input to the networks. In the left plot the networks with an input with a spatial resolution of 0.1 m are shown and in the right the networks with an input with spatial resolution of 0.05 m are shown.

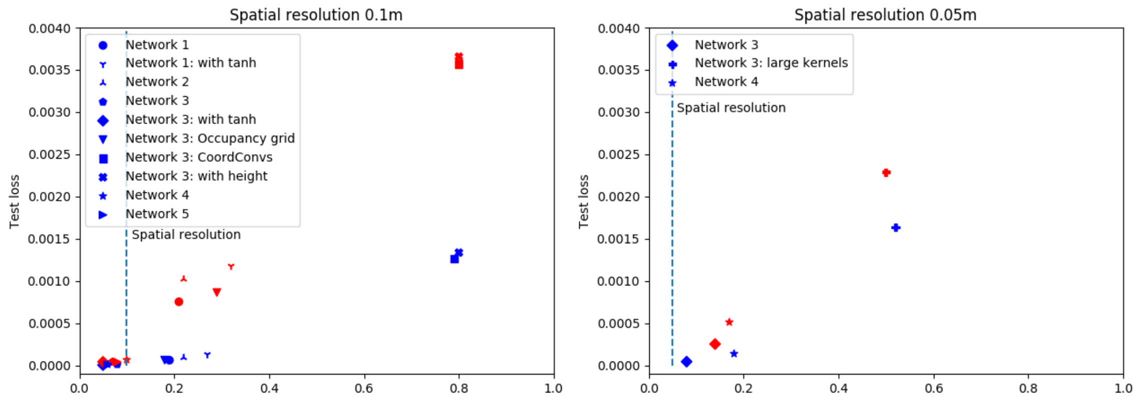


**Figure 4.4:** Plots presenting the test loss with respect to the median displacement error for all the networks, with 2D top view images as input, that we have trained. The left plot shows the networks with an input with a spatial resolution of 0.1 m and the right plot the networks with an input with a spatial resolution of 0.05 m. The vertical dotted lines represent the size of the spatial resolutions. The blue markers show the networks that were trained and tested on pure translation and the red markers show the networks that were trained and tested on translation and rotation. The best performing networks for a spatial resolution of 0.1 m in the left corner are Network 3, Network 3: with tanh, Network 4 and Network 5.

When comparing all the networks in the left plot with a spatial resolution of 0.1 m in Figure 4.4, it is seen that four networks succeed with yielding predictions that are more accurate than the spatial resolution itself. This holds both for pure translation and for translation with rotation. When looking at the right plot with a spatial resolution of 0.05 m, it is seen that none of the networks succeed with this. It can also be seen that Network 3 with tanh and input with a spatial resolution of 0.1 m outperforms all other networks. When testing on pure translation, its test loss ended at 0.0366 e-4 with a median translation error of 0.05 m. This network also outperformed the others when a rotation error was introduced in the samples. The test loss ended at 0.4183 e-4 with a median translation and rotation prediction error of 0.05 m and 0.0151 ° respectively.

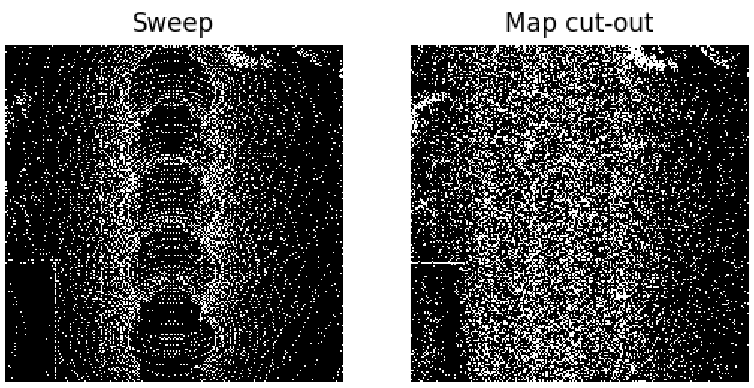
### 4.1.7 Which samples are easy or difficult to predict?

To see if there was some pattern regarding which samples that was easier or more difficult for Network 3 with the tanh activation function to predict, we compared the labels of the three best predictions and the three worst. In Table 4.10 the labels of the samples are presented starting with the one the network predicted best (sample 1) and ending with the poorest prediction (sample 6).

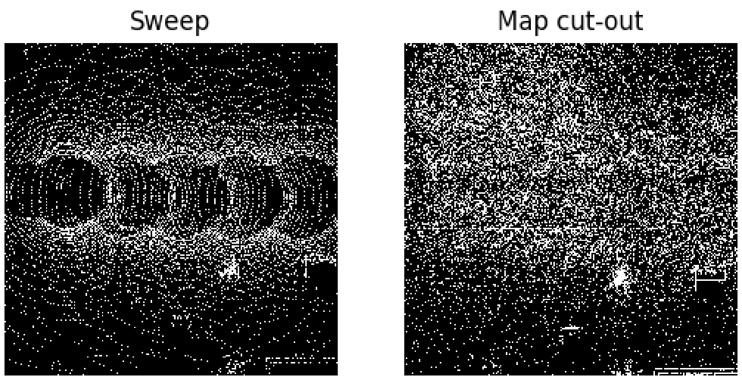
**Table 4.10:** This Table presents labels and predictions for some test samples on Network 4 with tanh as activation function. Sample 1-3 are the samples that the network performed best on, and sample 4-6 are the samples that it performed worst on. The labels are presented in full in column 2. The total translated distance introduced in the samples are given in column 3, whereas the predicted translated distance can be found in column 4. All the translation values are given in meters.

| Sample | Labels [x, y, $\theta$] | Total label translation | Total predicted translation |
|---|---|---|---|
| 1 | [ 0.85, -0.70, 0.00] | 1.10 | 1.10 |
| 2 | [-0.48, -0.73, 0.00] | 0.88 | 0.88 |
| 3 | [0.20, 0.75, 0.00] | 0.77 | 0.77 |
| 4 | [-0.93, -0.77, 0.00] | 1.21 | 1.03 |
| 5 | [-0.33, -0.74, 0.00] | 0.81 | 0.63 |
| 6 | [0.96, 0.72, 0.00] | 1.20 | 1.00 |

In Figure 4.5 the easiest and most difficult sample for the network to predict (sample 1 and sample 6) are visualized. Looking at the images one see that they are quite similar, which gives the impression that the network does not find some samples easier to predict than others. The images of samples 2-5 are shown in Figures B.1 and B.2 found in Appendix B.



(a) Sample 1, which was the easiest for the network to predict.



(b) Sample 6, which was the most difficult for the network to predict.

**Figure 4.5:** The figure is showing the easiest (a) and most difficult (b) sample for the network to predict. The left image is the sweep and the right the cut-out from the map.

## 4.2 Networks with LiDAR point clouds as input

In this section, the results from training the networks that takes the LiDAR point cloud as input will be presented. First the results from networks using Backbone 1 will be presented followed by the ones that use Backbone 2, 3 and 4. All the networks were tested using the SmoothL1 loss function with equal weighting for the error in translation $x$ and $y$ and angle $\theta$, independently of the loss used when training to be able to compare the networks' performances.

The test results are presented in tables and figures in the same way as for the networks presented in Section 4.1, with tables for test loss and other metrics, and clarifying histograms for the prediction error distributions when needed. Regarding the training and testing of the networks, all the networks have been trained and tested on a translation error in the range $[-1.5, 1.5]$ m and a rotation error in $[-3, 3]$ degrees if nothing else is stated in connection to the presented results.

### 4.2.1 Network with Backbone 1

The results presented in this section are all from training networks using Backbone 1. The backbone architecture can differ slightly at the input layers and the first fully connected layer depending on the spatial resolution and number of features that are used in the Pillar Feature Net, but is otherwise identical. Different combinations of spatial resolutions and number of features in the PFN-layer were trained and tested, as will be described below. The first combination was trained on training data with translation errors in the range $[-3, 3]$ m, and combination 2 and 3 were trained on training data with translation errors in the range $[-1.5, 1.5]$ m. No rotation errors were introduced in any combination.

1. The first version of the network was trained with spatial resolution of 0.5 m using using 32 features from the PFN-layer.
2. The second version of the network was trained with a spatial resolution of 0.25 m using 32 features from the PFN-layer.
3. The third version of the network was also trained with a spatial resolution of 0.25 m using 64 features from the PFN-layer.

All three versions are tested on a test set with pure translation in the range of $[-1.5, 1.5]$ m. The results are presented in Table 4.11.

**Table 4.11:** Test results for different versions of Backbone 1. All networks are tested on pure translation in the range of $[-1.5, 1.5]$ m. No rotation errors were introduced in the training or test data sets.

| Spatial resolution | Number of features | Test loss (1e-4) | Median translation prediction error | Median rotation prediction error |
|---|---|---|---|---|
| 0.5 m | 32 | 66.7357 | 1.07 m | 0.0129 ° |
| 0.25 m | 32 | 121.3999 | 1.03 m | -0.0031 ° |
| 0.25 m | 64 | 218.2622 | 0.95 m | -0.0092 ° |

The results in Table 4.11 are not satisfying in terms of test loss or median errors, and increasing the number of weights in the PFN-layer does not improve the overall performance. As seen in the table, the first version of the network with spatial resolution 0.5 m and 32 features got the highest median translation prediction error, while the test loss is the lowest. This trend also follows for the second and third version, where the second version with spatial resolution 0.25 m and 32 features has a lower test loss than the third version, but higher median translation error. This can be explained when looking at the distribution of the predicted errors. Version 1 has errors that are centered around zero, which may yield a lower test loss even if the distributions are not narrow for the predicted translation in $x$ and $y$. Version 1 and 2 predicts the translation in $x$ relatively well but the prediction of $y$ is very poor, which may lead to an increase in test loss even if the median translation error is lower. The histograms that visualize the distribution of the median prediction errors are shown in Appendix B in Figure B.4.

To investigate how Backbone 1 with a spatial resolution of 0.25 m and 64 features in the PFN-layer performed when predicting the rotation we continued to train it on rotation. We also trained a similar architecture with two PFN-layers with 64 channels each, since we wanted to see if the PFN's capacity needed to be increased to find important features in the point cloud. The results from these versions are presented in Table 4.12.

**Table 4.12:** The results when using Backbone 1 with a spatial resolution of 0.25 m and when using Backbone 1 with two PFN-layers.

| Spatial resolution | Number of features | test loss (1e-4) | median translation prediction error | median rotation prediction error |
|---|---|---|---|---|
| 0.25 m | 64 | 652.6360 | 0.92 m | 0.0768° |
| 0.25 m | 64, 64 | 749.7436 | 1.18 m | 0.1710° |

Figure 4.6 visualizes the distribution of the error of network's predictions when testing the version with a spatial resolution of 0.25 m and a single PFN-layer, trained on both translation and rotation. The error seems to be uniformly distributed for both translation in $y$ and for the rotation angle, indicating that this network does not have the capability of solving the localization task at all. The performance of the network with two PFN-layers was also poor. Due to time constraints we did not have time to investigate why, however a quick guess is that it might have been caused by an implementation error.

### 4.2.2 Network with Backbone 2

The network with Backbone 2 was trained with pillars of spatial resolution 0.5 m. The network was trained on pure translation and the number of features that was used in the PFN layer was 64. When testing the network on a test set with the same parameters as the training set the test loss was 142.2248 e-4 and the median translation prediction error was 1.13 m. By comparing these results with the results in Section 4.2.1 we drew the conclusion that Backbone 2 did not perform good enough to be developed further.

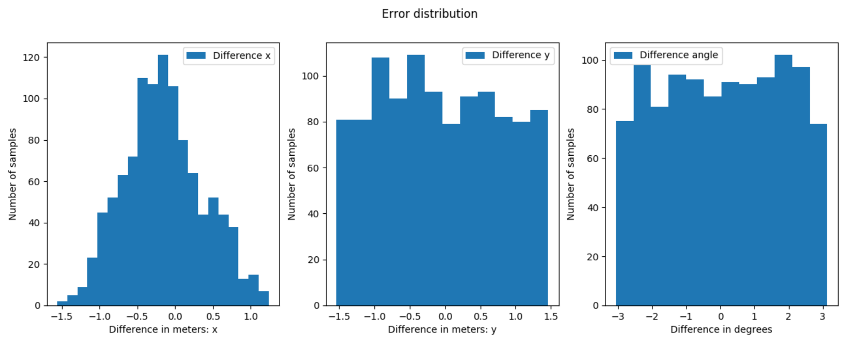


**Figure 4.6:** Histograms showing the translation and rotation error when testing on a set with translation errors in the range $[-1.5, 1.5]$ m and rotation errors in $[-3, 3]$ degrees. The distributions are uniformly distributed for $y$ and the angle, indicating that the performance for these output parameters have not been improved at all after training.

### 4.2.3 Networks with Backbone 3 and Backbone 4

Backbone 3, unlike Backbone 2, takes two separate inputs. The pseudo image of the sweep and the pseudo image of the map cut-out is processed separately through multiple convolutional layers before being concatenated and propagated through multiple fully connected layers. The results from this network are presented in Table 4.13.

**Table 4.13:** Performance of network with Backbone 3 trained on translation in range $[-1.5, 1.5]$ m with and without rotation in the range $[-3, 3]$ degrees.

| Spatial resolution | Number of features | Rotation | test loss (1e-4) | median translation prediction error | median rotation prediction error |
|---|---|---|---|---|---|
| 0.5 m | 32 | No | 27.4700 | 0.64 m | -0.0001 ° |
| 0.5 m | 32 | Yes | 132.4586 | 0.90 m | 0.4501 ° |

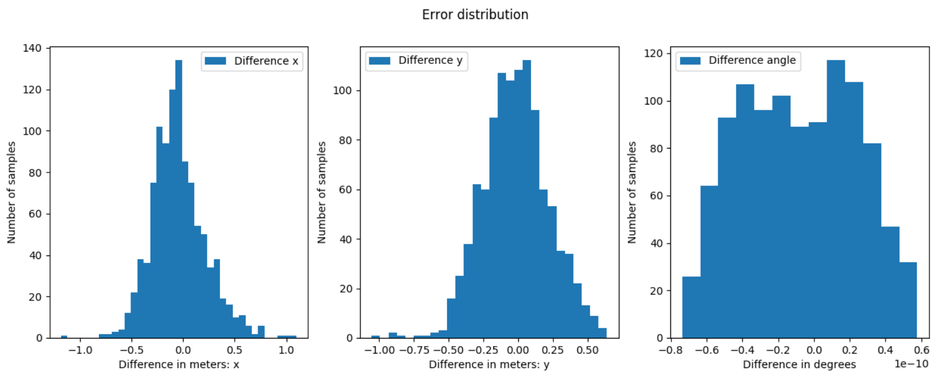
The test loss for Backbone 3 is considerable lower than for the other backbones, but it still performs poorly when predicting the rotation angle. Since we got promising results we decided to develop Backbone 3 further, resulting in Backbone 4. We tested Backbone 4 using two different spatial resolutions, 0.5 m and 0.25 m. The results are presented in Table 4.14.

By comparing the results in Table 4.14 with all other backbones' results, the conclusion is that this network with spatial resolution 0.5 m has the best performance. It is interesting to visualize the distribution of the networks error predictions as in Figure 4.7. Subfigure a) shows the prediction error histograms when trained and tested on pure translation, and we can see that the prediction errors are centered around zero, indicating that the network can predict the translation between sweep and cut-out fairly accurately. Subfigure b) shows the prediction errors when trained on samples with rotation as well, and the performance degrades noticeably as seen in Table 4.14. The histograms show that while the network can predict translation in $y$ and correct some of the rotation angle, it looses the ability to predict translation

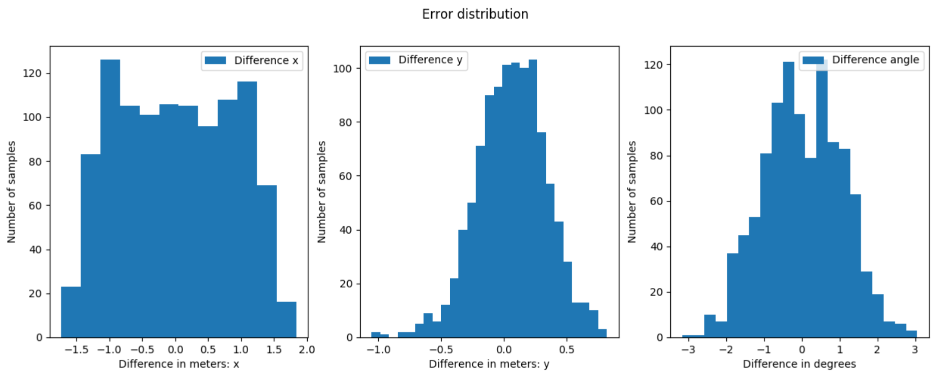
**Table 4.14:** Performance of network with Backbone 4 trained on translation in range $[-1.5, 1.5]$ m with and without rotation in the range $[-3, 3]$ °. The best performance is marked in bold text.

| Spatial resolution | Number of features | Rotation | test loss (1e-4) | median translation prediction error | median rotation prediction error |
|---|---|---|---|---|---|
| 0.5 m | 32 | No | **6.1458** | **0.28 m** | **0.0000 °** |
| 0.5 m | 32 | Yes | **85.3636** | **0.80 m** | **0.0064 °** |
| 0.25 m | 32 | No | 53.7752 | 0.64 m | 0.0000 ° |
| 0.25 m | 32 | Yes | 215.3096 | 0.92 m | 0.0913 ° |

in $x$. This will be further discussed in Chapter 5.



**(a)** Test results of Backbone 4 on pure translation, visualized as a histogram with the prediction error for the translation in $x, y$ in metres and the prediction error for the rotation in degrees.



**(b)** Test results of Backbone 4 on translation and rotation in, visualized as a histogram with the prediction error for the translation in $x, y$ in metres and the prediction error for the rotation in degrees.

**Figure 4.7:** a) Histograms showing the test results for Backbone 4 trained on pure translation in the range $[-1.5, 1.5]$ m. b) Histograms showing the test results for Backbone 4 trained on translation and rotation in the range $[-1.5, 1.5]$ m and $[-3, 3]$ degrees.

### 4.2.4 Summary of the networks' performances

In Figure 4.8, the losses with respect to the median displacement error for all the networks that uses point clouds as input are shown. The blue markers represent the performance of networks that have been trained and tested on only pure translation and the red markers represent the networks that have been trained and tested on translation as well as rotation. The dotted lines show the the spatial resolution.

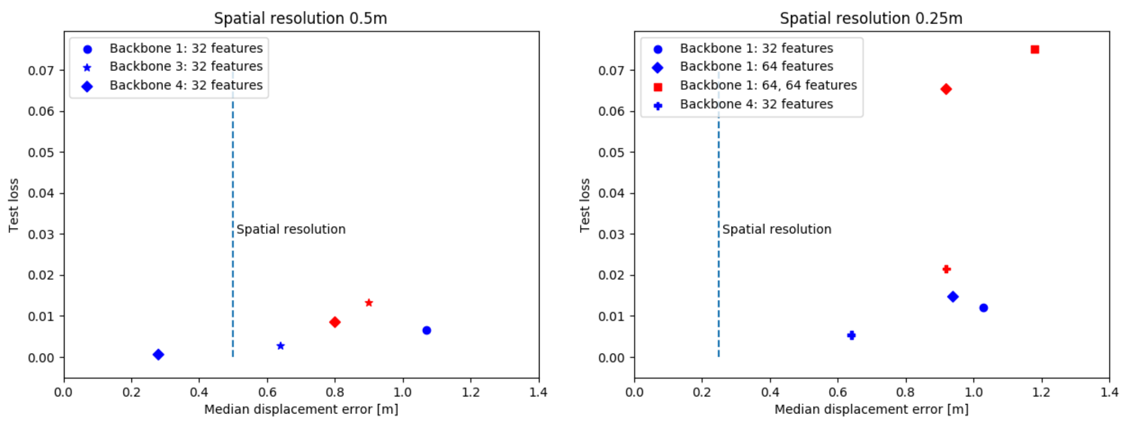


**Figure 4.8:** Plots presenting the test loss with respect to the median displacement errors for all the networks with point cloud as input that we have trained. The left plot shows the networks with a spatial resolution of 0.5 m and the right plot the networks with an a spatial resolution of 0.25 m. The vertical dotted lines represent the spatial resolutions. The blue markers show the networks that were trained and tested on pure translation and the red markers show the networks that were trained and tested on translation and rotation.

As seen from the plots, the best performing network when using point clouds as input was Backbone 4 in combination with a PFN-layer with 32 features for a spatial resolution of 0.5 m. It was the only network that succeeded to predict transformations with higher accuracy than the spatial resolution, when trained and tested on pure translation. The test loss ended at 6.1458 e-4 with median translation prediction error of 0.28 m. When trained and tested on translation together with rotation, the test loss was 85.3636 e-4 with median translation prediction error of 0.80 m and median rotation prediction error 0.0064°. The introduced rotation degrades the performance noticeably, but it is still the best performing network.

# 5
# Discussion

In this chapter, we discuss our results beginning with a discussion about important features in the inputs to the networks. Different kind of handcrafted features in the 2D input yielded varying results, and some features degraded the performance completely. Different sizes of the grids and pillars in 2D-input and point cloud input greatly affected the performance as well. The discussion continues with the design of the network architectures, and which choices that led to an improved performance. Some interesting design choices are for example if the sweep and cut-out are processed separately by the network, or which activation function that worked best. We will also discuss the test data that was easiest and most difficult for the networks to predict, and if there are any patterns to these samples. The chapter continues with a broad discussion about regression for this kind of localization problem. From the results we can see that rotation errors are difficult for the network to predict, and hard to distinguish from pure translation. Finally, we will conclude the chapter with some future work that should be investigated, should this project continue.

## 5.1 Input to networks

What we have found to be one the most crucial parameters when developing our networks, is the input to the network itself. The network architecture and training hyperparameters do of course impact the results, but even our best performing network degraded completely when changing details in the input. Some parameters have remained unchanged during the whole project and will be discussed later. Given more time, their importance might be interesting to investigate. Other parameters that we have altered will be discussed in the next section.

When processing the LiDAR data, we removed the detections that originate from the vehicle itself, such as detections on the hood. This was done after visual investigation of all detections overlayed to visualize the towns, where we saw that the roads were cluttered with detections from the vehicle. Since these detections are not static, we decided to remove all detections within 1 m of the LiDAR. A related idea we wanted to investigate was to remove all detections on the road surface, letting the networks learn that areas that lack detections are most likely drivable areas, whereas sidewalks would emerge as distinct structures that separate roads from all other structures. This idea was difficult to implement, since the urban simulator CARLA behaved unexpectedly when moving the vehicle to new locations during the collection of data. The LiDAR data from CARLA lacks the specific property of reflective intensity that is often used in data from real LiDARs. This feature would most likely have been included in the training data if it had been available.

Normalization of the data can have a big impact on the results. We decided to normalize the number of detections relative to the maximum number of detections. This is important, since the sweep and map cut-out contain different amounts of detections in total. To make the pixel values comparable, we normalize them separately. This should yield similar pixel values for the same structures in the sweep and the cut-out.

## Input features

We have identified some input parameters that seem to be more important than others for how well the neural networks perform. In the case of networks with a 2D images as input, we as developers have to decide how to pre-process the data and which information the network should be fed as input. Our standard approach was to discretize the LiDAR data into a grid structure, where each pixel represents an intensity of detections. This produces a feature map where structures that obstructs the LiDAR sensor beams, such as building walls, yield many detections. These pixels will then have a larger value than a pixel for road surface, which only have a few detections from the ground. This kind of input have performed best. Feature maps like this should work well for environments where there are different kind of objects with differing heights around the vehicle, such as buildings, bus stops, fences, as is the case in the simulated environment CARLA. We also trained networks using occupancy grid, with the hope that the localization task would boil down to finding patches of zeros and ones that matched in size and shape. This approach worked, but did not yield better results than using the number of detections as described above. A natural next step in the development of the input was to include a layer of mean height in each pixel. We think that this should help the network to distinguish more structures, such as sidewalks and road surfaces which are both flat but at different heights. This did, surprisingly enough, yield poor results. A modified version of Network 3 was trained with this input, but also deeper networks with more layers since we anticipated that the networks capacity should increase when including more information in the input, but still with poor results.

## Spatial resolution

Intuitively, the grid size should be a very important parameter in the neural network. A finer grid structure should lead to a more accurate prediction, since the network is then able to pinpoint smaller displacements between the sweep and the cut-out. Network 3 was modified such that it could use inputs with a spatial resolution of 0.1 m and 0.05 m, with slight differences in the architecture to handle the different input sizes. The results show that decreasing the spatial resolution did not improve the accuracy, instead the test loss and median translation error almost doubled. We still believe firmly that decreasing the spatial resolution should lead to better predictions, however it seems that the network architecture can only be used on the spatial resolution that it was developed for. In order to actually make use of the finer resolution, the architecture has to be developed differently to perform better. One example of a parameter that could be changed in the architecture can for example be the kernel size of the convolution layers. If the spatial resolution is decreased in size, the kernel's size should be increased such that the receptive field remains the same as for a larger spatial resolution. This was implemented, but with poor results indicating that the kernel size is not the most crucial factor when designing an architecture for a finer spatial resolution. This line of reasoning also applies to the backbones when using point clouds as input. Smaller pillars yield more feature vectors which should then also be processed with a larger receptive field. This has not yet been tested.

## 5.2 Architecture design

In this section, the architecture designs of the networks with top-view LiDAR images as input will be discussed, followed by the networks with the point cloud as input. We will highlight differences that may be a factor as to why some of them are better to solve the localization task than others.

### Networks with top-view LiDAR images as input

The network that performed best among the top-view LiDAR images networks was Network 3 with tanh, which is inspired by a network designed for estimating optical flow in image pairs. One thing that may be a factor to why Network 3 with tanh outperforms the others, is that it has a larger kernel size in the first three convolutional layers. This increases the receptive field of the network, which we believe is an important parameter, as mentioned before in the section about spatial resolution. Overall, the network is deeper than the other networks with more channels in many of the layers and more fully connected layers. A deeper network often have a higher potential to solve difficult tasks and it seems that this is necessary to solve the localization task. The network also has dropout in almost every layer, which can be an additional factor. The drop out parameter $p$, which decides the amount of neurons to ignore during training, can be a promising parameter to experiment with in order to increase the performance. Network 3 with tanh also uses two different activation functions; the ReLU function was used in the first half of the network while the three last convolutional layers and the two first fully connected had the tanh as activation function and the rest of the fully connected layers did not use an activation function. When using the tanh activation function in the network instead of only the ReLU function the network performed marginally better, however we think that the difference was too small to state that the choice of activation function had a vital role in this case.

Processing the sweep and map cut-out separately was a design choice that we thought would lead to better performance, since the network can in some sense find important structures in the images firstly, and secondly compare these structures in the images. Our results show the opposite, the performance degrades when processing the inputs separately. This trend is also seen when looking at Network 1 and 2, where Network 2 also takes two separate inputs.

One of the network architectures, Network 5, is very different from the other, with residual blocks and deconvolutional layers. The network uses a version of LinkNet to process the sweep and cut-out separately before concatenating them and processing them trough convolutional and fully connected layers as usual. This new approach performed well on the localization task and should definitely be investigated further by changing hyper parameters such as number of channels, kernel sizes, etc.

#### Networks with point cloud as input

The backbone that performed best when solving the localization task was Backbone 4. The main difference between Backbone 4 and the other backbones is that Backbone 4 processes the inputs separately through all the convolutional layers, before concatenating them deeper in the network to pass them through multiple fully connected layers. We think that processing the inputs separately may help the network to find important features for the sweep and cut-out. What is surprising is that this tactic did not work for the top-view input networks, even though it seems to be crucial for the backbone design. An idea to further increase the performance of Backbone 4 is to increase the kernel size in the first layers to increase the receptive field of the network. It can also be the case that the network should be deeper to have the potential to solve the task better.

## 5.3 Is some data easier to learn than other data?

To understand our networks better we investigated if some kind of samples were easier to predict than others. We compared samples where Network 3 gave a prediction close to the ground truth with samples that yielded a poor prediction. Our expectation was that samples with clear structures or distinct objects should be easier to predict than samples that were blurry with many detections distributed over the whole image. Unfortunately a clear trend was not found. The sample that gave a prediction close to the ground truth was similar to the one that the network found hard to predict. The labels of the samples, i.e the size of errors that was introduced in the data, did not seem to have an impact on the performance either, since a small error sometimes was difficult to predict while a large error was easy. To investigate this further it can be a good idea to include an output parameter from the network that shows how certain it is about each prediction. This knowledge can then be used to find a possible pattern among the samples.

## 5.4 Regression for rotation problems

A common issue for all networks in this thesis project is that the networks' performances degrade when introducing data that has been rotated. The networks can identify and correct pure translation, but rotation errors increase the overall loss, often affecting and degrading the translation predictions in one direction, e.g in $x$ or $y$. A hypothesis to this phenomenon is that the networks have difficulties distinguishing pure translation from displacements that are induced by rotation. A pixel in the outer edge of an image shifts more pixels than a pixel in the origin when the image is rotated around its center. The convolutional kernels work locally and may have a hard time identifying this subtle difference, and instead interpreting the outer edge pixels as affected by pure translation. Since both translation and rotation errors are possibly interpreted as translation in two dimensions by the networks, one of the elements in the output vector seems to be neglected. Two elements are enough to describe the transformation between the sweep and the cut-out, if you experience only translation as the networks seem to do.

A plausible conclusion is that regression is simply not suitable when rotation errors are prone to occur. Another possible way to attack this kind of problem is to instead use classification methods. This can be done using different bins for possible angle intervals and then use cross-correlation to check which one of the rotated versions that match the map cut-out best. A downside with using classification methods is that the accuracy of the rotation error that can be predicted by the network is limited. Our initial thought with using a regression method instead of classification was that we wanted the networks to be able to solve for more than a number of predefined possible rotation errors.

## 5.5 Future work

In this section we will give ideas of what we think could be interesting to investigate further in the future.

Since we did not succeed with using regression methods to solve this localization problem with results that can compete with state-of-the-art methods, more investigations should be carried out. Some things that can be further investigated are for example different kinds of architectures, experiment with hyper parameters and/or use different kind of handcrafted features in the input. The point cloud input could for example include features like cluster mean and $x$ and $y$ coordinates. For the 2D input we could include features such as maximum, minimum and mean height of the detections.

Another approach could be to use regression in combination with classification. Regression could be used to predict the translation and classification could be used for rotation bins. Another suggestion on how to proceed with the development of the networks is to introduce iterative parts, where the network corrects the sweep with its own prediction of the rigid transformation, before processing the sample again through the network to output a new refined prediction. One idea can also be to solve the problem in two steps. Firstly the network could solve the translation task and correct the error, and then pass the corrected sample to another part of the network that solves for the rotation angle.

When a network architecture that can solve the task is found, it would be interesting to see if the network can generalize well on different kind of LiDARs. When the networks perform satisfyingly, some run time analysis should be performed in order to evaluate if the network can run in real time. Another interesting idea that could be investigated further is to develop neural networks where the sweep and cut-outs are processed separately that perform well, such that the whole map can be processed offline before it is stored in the vehicle computer. If the network can find a suitable representation of the map that takes up less memory than the original point cloud map, valuable storage on the hard disk can be freed. Ideally the map should be memory efficient such that it can have potential to be stored in a vehicle computer, or downloaded during the ride.

Lastly, some code can be optimized to decrease the training time, especially in the case for the point cloud networks where training a network can take up to 24 hours. Some of the poor results could also be explained by poor implementations, such as the networks using CoordConvs or height information as an input feature. In these cases, the networks did not converge during training.

# 6 Conclusion

In this thesis project, we present different types of neural networks that estimate a rigid transformation between a LiDAR sweep and a map cut-out using neural network regression, to perform localization in an a priori known map. We have found that the localization task can be solved by using 2D images as input with only one feature, which is the intensity of detections in each pixel. Networks using point clouds should intuitively perform better since the point cloud contains more information and is not pre-processed, and the results show that the point cloud networks do perform relatively better compared to their pillar size, however they do not outperform the networks using 2D images. The overall aim of the thesis was to localize a vehicle in an a priori known map with centimetre level precision, using a LiDAR sensor and deep neural networks. Our results show that this localization task can indeed be carried out with our methods, but the evaluated networks can not compete with state-of-the-art methods today.

# Bibliography

[1] A. Chaurasia and E. Culurciello. Linknet: Exploiting encoder representations for efficient semantic segmentation. In *2017 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4, December 2017.

[2] National Highway Traffic Safety Administration. Automated vehicles for safety. Online: `https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety#topic-benefits`. Accessed: 2019-01-15.

[3] On-Road Automated Driving (ORAD) committee. Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems, January 2014. Standard J3016_201401.

[4] S. Yu, T. Westfechtel, R. Hamada, K. Ohno, and S. Tadokoro. Vehicle detection and localization on bird's eye view elevation images using convolutional neural network. In *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pages 102–109, October 2017.

[5] W. Burgard, O. Brock, and C. Stachniss. Map-based precision vehicle localization in urban environments. In *Robotics: Science and Systems III*. MIT Press, 2008.

[6] P. Besl and H.D. McKay. A method for registration of 3-d shapes. ieee trans pattern anal mach intell. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 14:239–256, March 1992.

[7] J. Ziegler, H. Lategahn, M. Schreiber, C. G. Keller, C. Knöppel, J. Hipp, M. Haueis, and C. Stiller. Video based localization for bertha. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 1231–1238, June 2014.

[8] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, June 2006.

[9] I. A. Barsan, S. Wang, A. Pokrovsky, and R. Urtasun. Learning to localize using a lidar intensity map. In *Proceedings of The 2nd Conference on Robot Learning*, volume 87, pages 605–616, October 2018.

[10] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, pages 77–85, July 2017.

[11] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *CVPR*, pages 4490–4499, June 2018.

[12] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, October 2018.

[13] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766, December 2015.

[14] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox. Demon: Depth and motion network for learning monocular stereo. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5622–5631, 2017.

[15] R. Liu, J. Lehman, P. Molino, F. Petroski Such, E. Frank, A. Sergeev, and J. Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. In *Advances in Neural Information Processing Systems 31*, July 2018.

[16] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. Carla: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[17] O. Yalcin, A. Sayar, O. F. Arar, S. Apinar, and S. Kosunalp. Detection of road boundaries and obstacles using lidar. In *2014 6th Computer Science and Electronic Engineering Conference (CEEC)*, pages 6–10, September 2014.

[18] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[19] M. Wahde. *Biologically Inspired Optimization Methods: An introduction.* WIT Press, Southhampton, Boston, 2008.

[20] Medium. *Understanding Activation Functions in Neural Networks.* `https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0`. Accessed: 2019-04-30.

[21] Machine Learning Mastery. *Loss and loss functions for training deep learning neural networks.* `https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/`. Accessed: 2019-04-30.

[22] Pytorch. *SmoothL1.* `https://pytorch.org/docs/stable/nn.html`. Accessed: 2019-04-30.

[23] O. Enqvist. *Lecture Notes in Image Analysis.* Chalmers University of Technology, Gothenburg, Sweden, 2018.

[24] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, December 2014.

[25] C. M. Bisgop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[27] deeplearning.ai. Normalizing activations in a network (c2w3l04). Online: `https://www.youtube.com/watch?v=tNIpEZLv_eg`. Accessed: 2019-05-14.

[28] L. Caltagirone, S. Scheidegger, L. Svensson, and M. Wahde. Fast lidar-based road detection using fully convolutional neural networks. *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1019–1024, 2017.

# A

# Supplementary network architectures

This Appendix presents all versions of the main architectures presented in Chapter 3. For example, architectures with slight modifications for different input sizes or versions with different activation functions can be found here.

## Networks with top-view LiDAR images as input

Complementary architectures for networks with top-view LiDAR images as input are presented in this section.

### Network 3

Figure A.1 presents the network version of Network 3 used when the input with a spatial resolution 0.05 m and therefore an input size of $2 \times 600 \times 600$ pixels. There is a similar version in Figure A.2 where the convolutional kernels are larger, to increase the receptive field. Figure A.3 is the architecture used when the sweep and cut-out are processed separately by the first three convolutional layers. The last version of Network 3 is presented in Figure A.4, which is identical to the original but with tanh as activation function in some convolutional layers and in some fully connected layers.

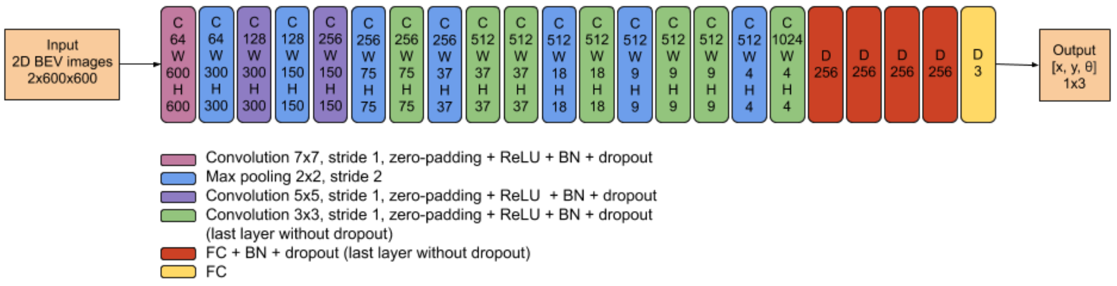

Input 2D BEV images 2x600x600
C 64 W 600 H 600
C 64 W 300 H 300
C 128 W 300 H 300
C 128 W 150 H 150
C 256 W 150 H 150
C 256 W 75 H 75
C 256 W 75 H 75
C 256 W 37 H 37
C 512 W 37 H 37
C 512 W 37 H 37
C 512 W 18 H 18
C 512 W 18 H 18
C 512 W 9 H 9
C 512 W 9 H 9
C 512 W 9 H 9
C 512 W 4 H 4
C 1024 W 4 H 4
D 256
D 256
D 256
D 256
D 3
Output [x, y, θ] 1x3
Convolution 7x7, stride 1, zero-padding + ReLU + BN + dropout
Max pooling 2x2, stride 2
Convolution 5x5, stride 1, zero-padding + ReLU + BN + dropout
Convolution 3x3, stride 1, zero-padding + ReLU + BN + dropout (last layer without dropout)
FC + BN + dropout (last layer without dropout)
FC

**Figure A.1:** Architecture for Network 3 when the input has a spatial resolution of 0.05 m.

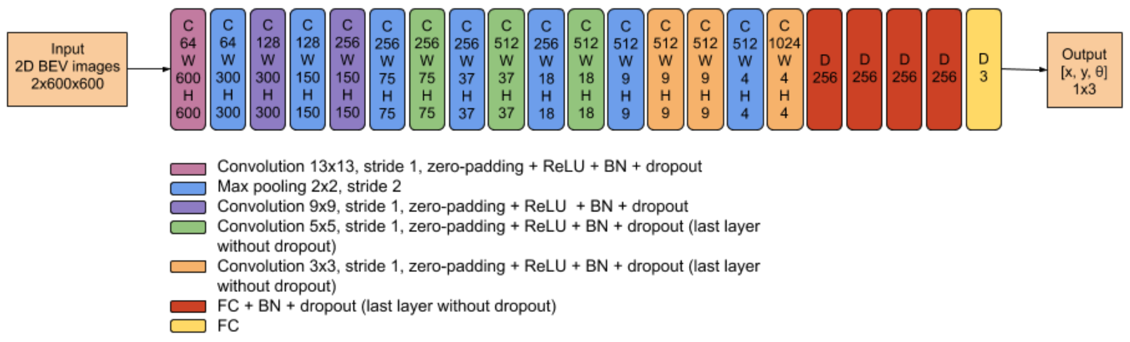

Input 2D BEV images 2x600x600
C 64 W 600 H 600
C 64 W 300 H 300
C 128 W 300 H 300
C 128 W 150 H 150
C 256 W 150 H 150
C 256 W 75 H 75
C 256 W 75 H 75
C 256 W 37 H 37
C 512 W 37 H 37
C 256 W 18 H 18
C 512 W 18 H 18
C 512 W 9 H 9
C 512 W 9 H 9
C 512 W 9 H 9
C 512 W 4 H 4
C 1024 W 4 H 4
D 256
D 256
D 256
D 256
D 3
Output [x, y, θ] 1x3
Convolution 13x13, stride 1, zero-padding + ReLU + BN + dropout
Max pooling 2x2, stride 2
Convolution 9x9, stride 1, zero-padding + ReLU + BN + dropout
Convolution 5x5, stride 1, zero-padding + ReLU + BN + dropout (last layer without dropout)
Convolution 3x3, stride 1, zero-padding + ReLU + BN + dropout (last layer without dropout)
FC + BN + dropout (last layer without dropout)
FC

**Figure A.2:** Architecture for Network 3 when the input has a spatial resolution of 0.05 m. The kernels are larger compared to Figure A.1, such that the receptive field increases. This receptive field is now as large, in terms of meters, as for Network 3 for spatial resolution 0.1 m.

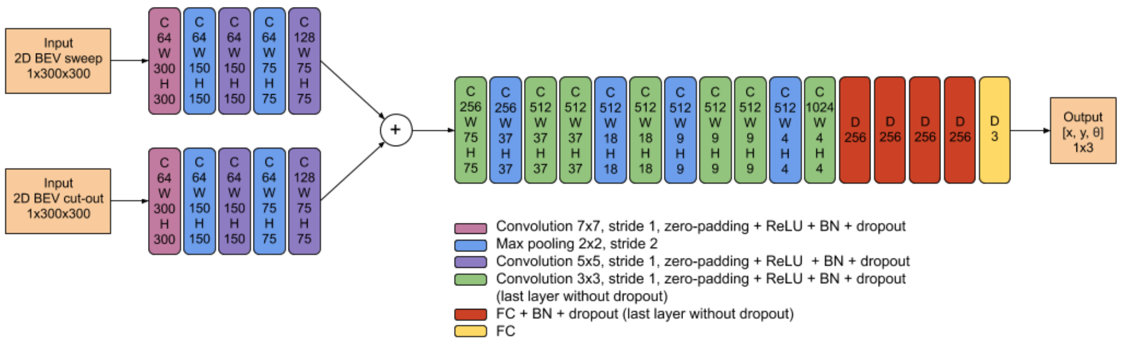

Input 2D BEV sweep 1x300x300
Input 2D BEV cut-out 1x300x300
C 64 W 300 H 300
C 64 W 150 H 150
C 64 W 150 H 150
C 64 W 75 H 75
C 128 W 75 H 75
+
C 256 W 75 H 75
C 256 W 37 H 37
C 512 W 37 H 37
C 512 W 37 H 37
C 512 W 18 H 18
C 512 W 18 H 18
C 512 W 9 H 9
C 512 W 9 H 9
C 512 W 9 H 9
C 512 W 4 H 4
C 1024 W 4 H 4
D 256
D 256
D 256
D 256
D 3
Output [x, y, θ] 1x3
Convolution 7x7, stride 1, zero-padding + ReLU + BN + dropout
Max pooling 2x2, stride 2
Convolution 5x5, stride 1, zero-padding + ReLU + BN + dropout
Convolution 3x3, stride 1, zero-padding + ReLU + BN + dropout (last layer without dropout)
FC + BN + dropout (last layer without dropout)
FC

**Figure A.3:** Architecture for Network 3 when the input sweep and cut-out are processed separately by the first layers. The spatial resolution is 0.1 m.

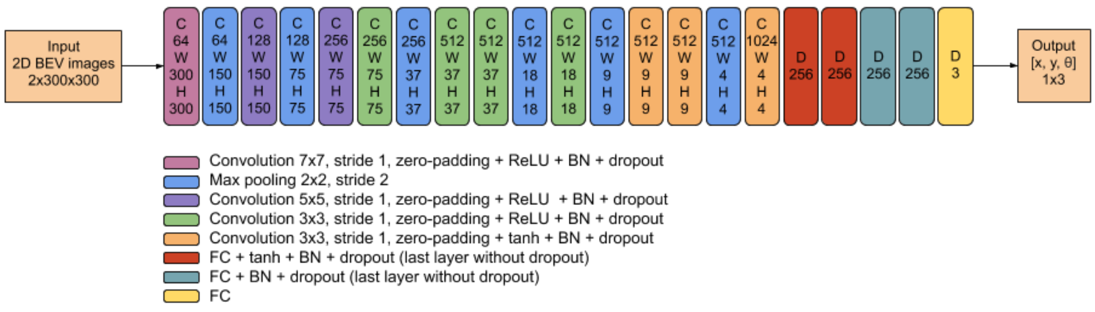

Input 2D BEV images 2x300x300
C 64 W 300 H 300
C 64 W 150 H 150
C 128 W 150 H 150
C 128 W 75 H 75
C 256 W 75 H 75
C 256 W 75 H 75
C 256 W 37 H 37
C 512 W 37 H 37
C 512 W 37 H 37
C 512 W 18 H 18
C 512 W 18 H 18
C 512 W 9 H 9
C 512 W 9 H 9
C 512 W 9 H 9
C 512 W 4 H 4
C 1024 W 4 H 4
D 256
D 256
D 256
D 256
D 3
Output [x, y, θ] 1x3
Convolution 7x7, stride 1, zero-padding + ReLU + BN + dropout
Max pooling 2x2, stride 2
Convolution 5x5, stride 1, zero-padding + ReLU + BN + dropout
Convolution 3x3, stride 1, zero-padding + ReLU + BN + dropout
Convolution 3x3, stride 1, zero-padding + tanh + BN + dropout
FC + tanh + BN + dropout (last layer without dropout)
FC + BN + dropout (last layer without dropout)
FC

**Figure A.4:** Architecture for Network 3 when using tanh as activation function in the last three convolutional layers instead of the ReLU function. Tanh is also used as activation function in the two first fully connected layers. The spatial resolution is 0.1 m.

## Networks with LiDAR point clouds as input

Complementary architectures for networks with LiDAR point clouds as input are presented in this section. Figure A.5 shows the design of multiple PFN-layers that are used before to create the pseudo-images. Figure A.6 shows the network architecture for Backbone 1, but modified to suit inputs with spatial resolution 0.25 m. Figure A.7 shows the network for Backbone 4, modified for inputs with spatial resolution 0.25 m and 32 features from the PFN-layers.

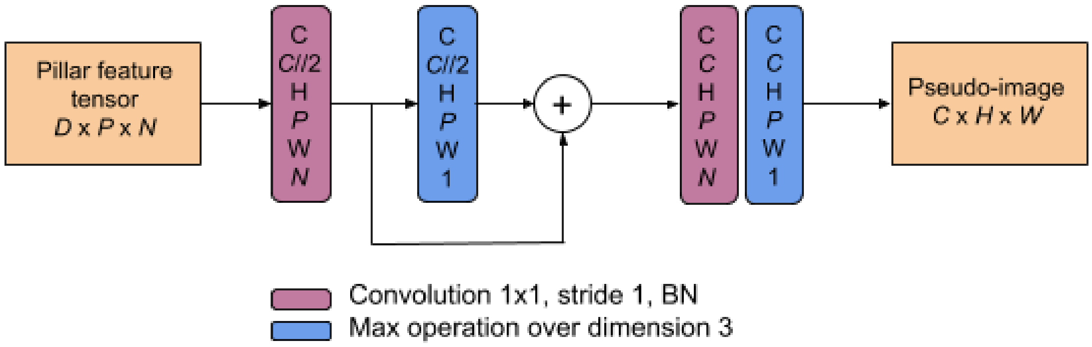


**Figure A.5:** A visualization of the PFN network with 2 PFN-layers. The output from the first layer is concatenated with the max operation value, before passed through a new PFN-layer.

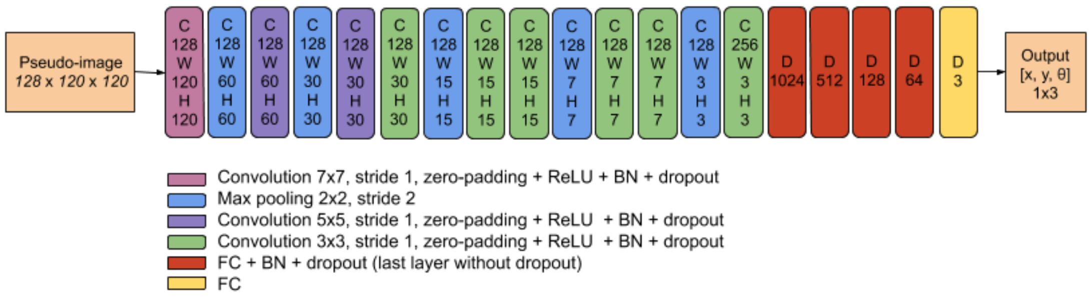


**Figure A.6:** A version of Backbone 1, but where the input has a spatial resolution of 0.25 m.

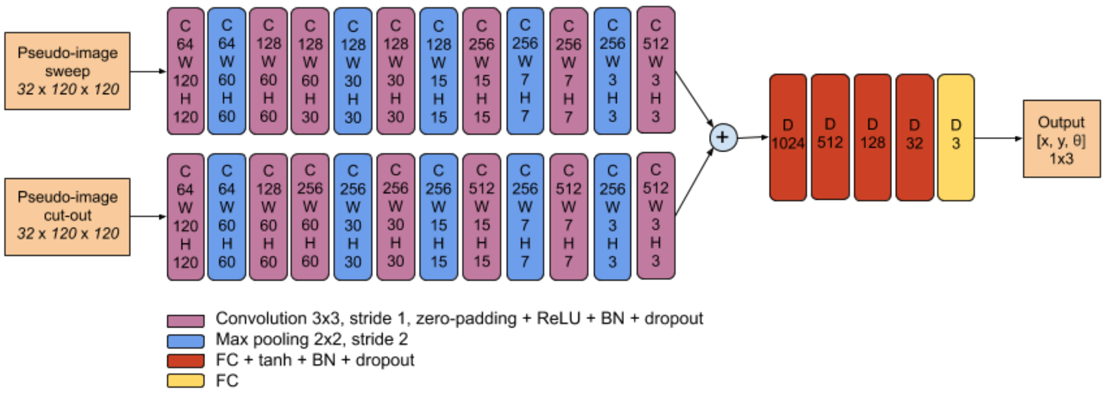


**Figure A.7:** Backbone 4 with spatial resolution 0.25 m and 32 features from PFN-layer. Sweep and cutout pseudo-images are processed separately by convolutional layers before being concatenated and sent through fully connected layers.

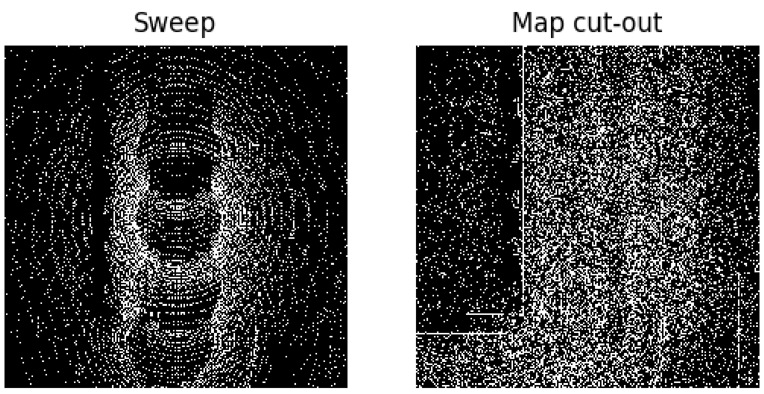
# B
# Supplementary results

## Visualization of samples that was predicted by Network 3

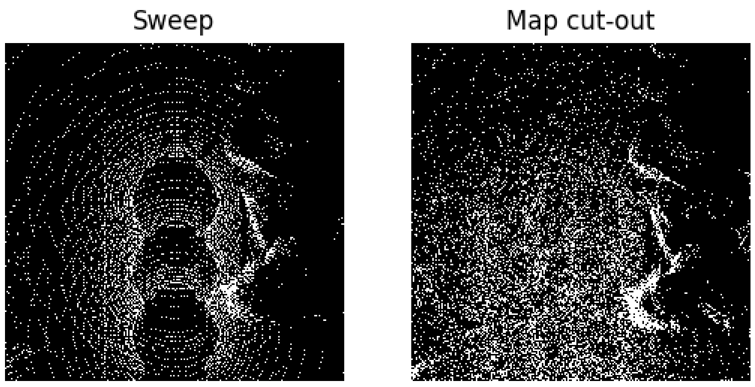
Figure B.1 shows the 2nd and 3rd easiest samples for Network 3 to predict and Figure B.2 shows the 2nd and 3rd most difficult samples. The labels for the images are presented in Table B.1.

**Table B.1:** Labels of the samples shown in Figures B.1 and B.2

| Sample | Labels [x, y, $\theta$] | Total label translation | Total predicted translation |
|---|---|---|---|
| 2 | [-0.48, -0.73, 0.00] | 0.88 | 0.88 |
| 3 | [0.20, 0.75, 0.00] | 0.77 | 0.77 |
| 4 | [-0.93, -0.77, 0.00] | 1.21 | 1.03 |
| 5 | [-0.33, -0.74, 0.00] | 0.81 | 0.63 |



(a) Sample 2, which was the 2nd easiest for the network to predict.



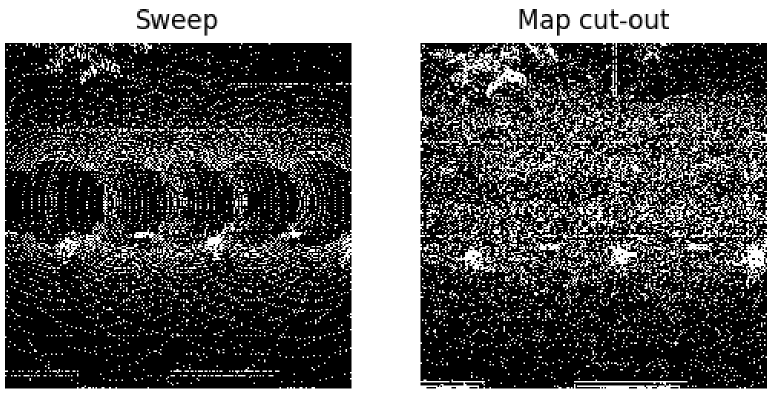
(b) Sample 3, which was 3rd easiest for the network to predict.

**Figure B.1:** The figure is showing sample 2 and sample 3 which where the 2nd and 3rd easiest samples for Network 3 to predict. The left image is the sweep and the right the cut out from the map.

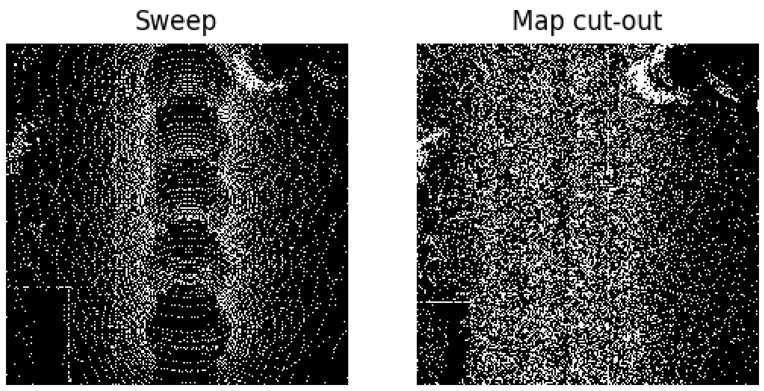
## Prediction error histograms

Figure B.3 shows the test results of Backbone 4 when using a spatial resolution of 0.25 m, 32 features in the PFN layer. Figure B.3 a) shows the results when the backbone was trained and tested on pure translation in the interval $[-1.5, 1.5]$ m and Figure B.3 b) shows the test results after continuing training on rotation in the range $[-3, 3]$ degrees.

Figure B.4 shows the results of Backbone 1 when it is trained and tested on pure translation. Figure B.4 a) shows the the predicted error distribution when using a spatial resolution of 0.5 m and 32 features from the PFN-layer, Figure B.4 b) shows the predicted error distribution when using a spatial resolution of 0.25 m and 32 features from the PFN-layer and Figure B.4 c) shows the predicted error distribution when using a spatial resolution of 0.25 m and 64 features from the PFN-layer.

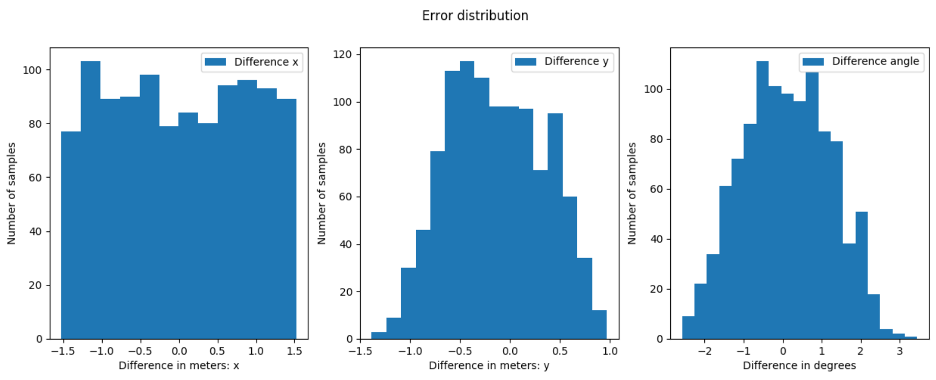


**(a)** Sample 4, which was the 3rd hardest for the network to predict.

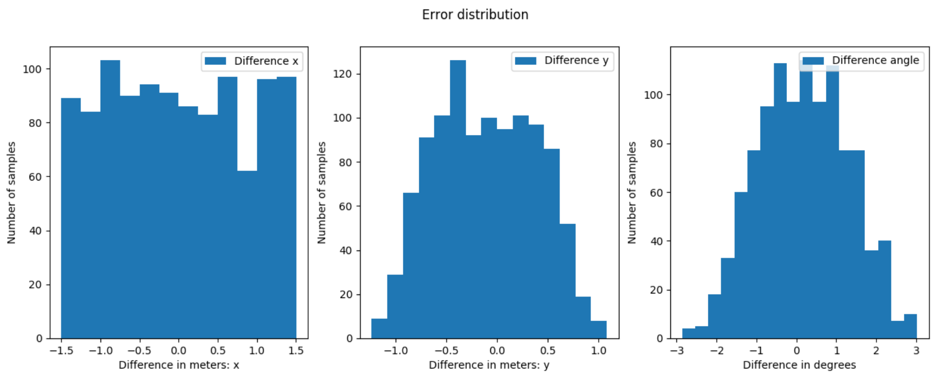


**(b)** Sample 5, which was the 2nd hardest for the network to predict.

**Figure B.2:** The figure is showing sample 4 and 5 which where the 2nd and 3rd hardest samples for Network 3 to predict. The left image is the sweep and the right the cut out from the map.

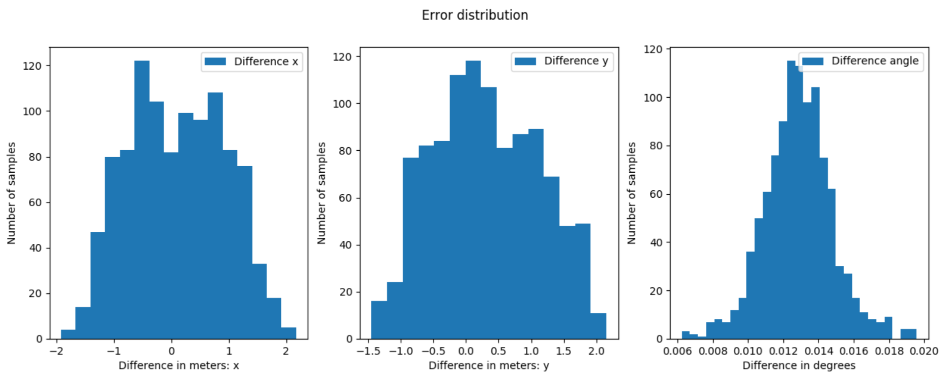


**(a)** Test results of Backbone 4, with a spatial resolution of 0.25 m, on pure translation, visualized as a histogram with the prediction error for the translation in $x, y$ in metres and the prediction error for the rotation in degrees.

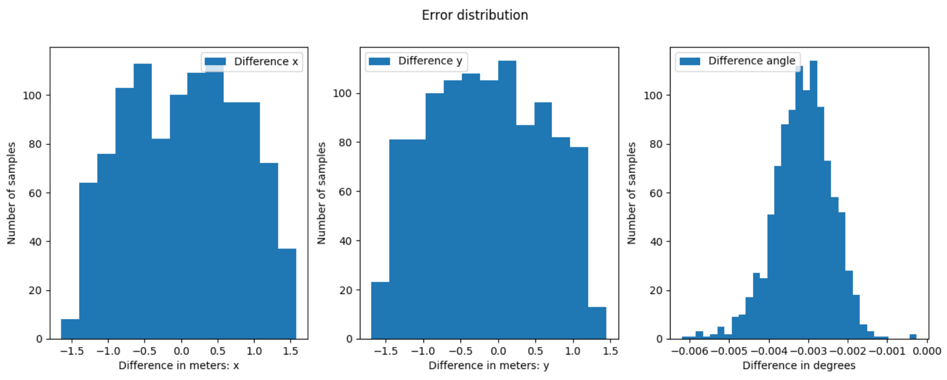


**(b)** Test results of Backbone 4, with a spatial resolution of 0.25 m, on translation and rotation in, visualized as a histogram with the prediction error for the translation in $x, y$ in metres and the prediction error for the rotation in degrees.

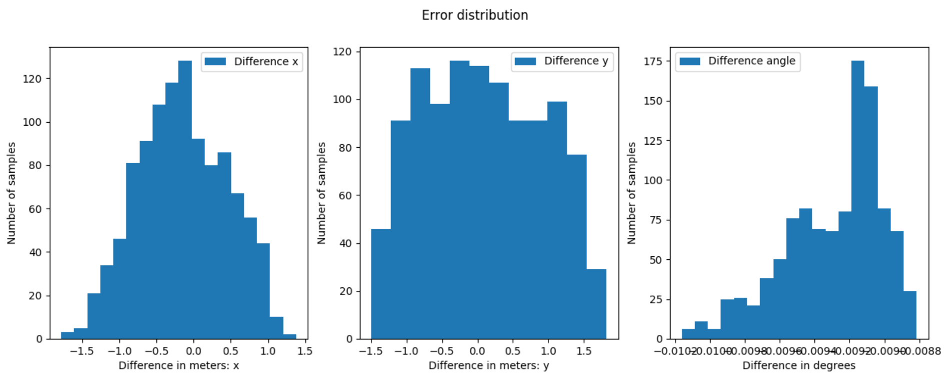
**Figure B.3:** a) Histograms showing the test results for Backbone 4, with a pillar size of 0.25m, trained on pure translation in the range [-1.5,1.5]m. b) Histograms showing the test results for Backbone 4 trained on translation and rotation in the range [-1.5,1.5]m and [-3,3]°.



**(a)** Histogram showing the predicted error distribution when using a spatial distribution of 0.5 m and 32 features from the PFN-layer.



**(b)** Histogram showing the predicted error distribution when using a spatial distribution of 0.25 m and 32 features from the PFN-layer.



**(c)** Histogram showing the predicted error distribution when using a spatial distribution of 0.25 m and 64 features from the PFN-layer.

**Figure B.4:** The figure presents histograms over the predicted error distribution of a network using Backbone 1. a) shows the predicted error when using spatial resolution 0.5 m and 32 features from the PFN-layer. Subfigure b) shows the predicted error when using spatial resolution 0.25 m and 32 features from the PFN-layer. Subfigure c) shows the predicted error when using spatial resolution 0.25 m and 64 features from the PFN-layer.