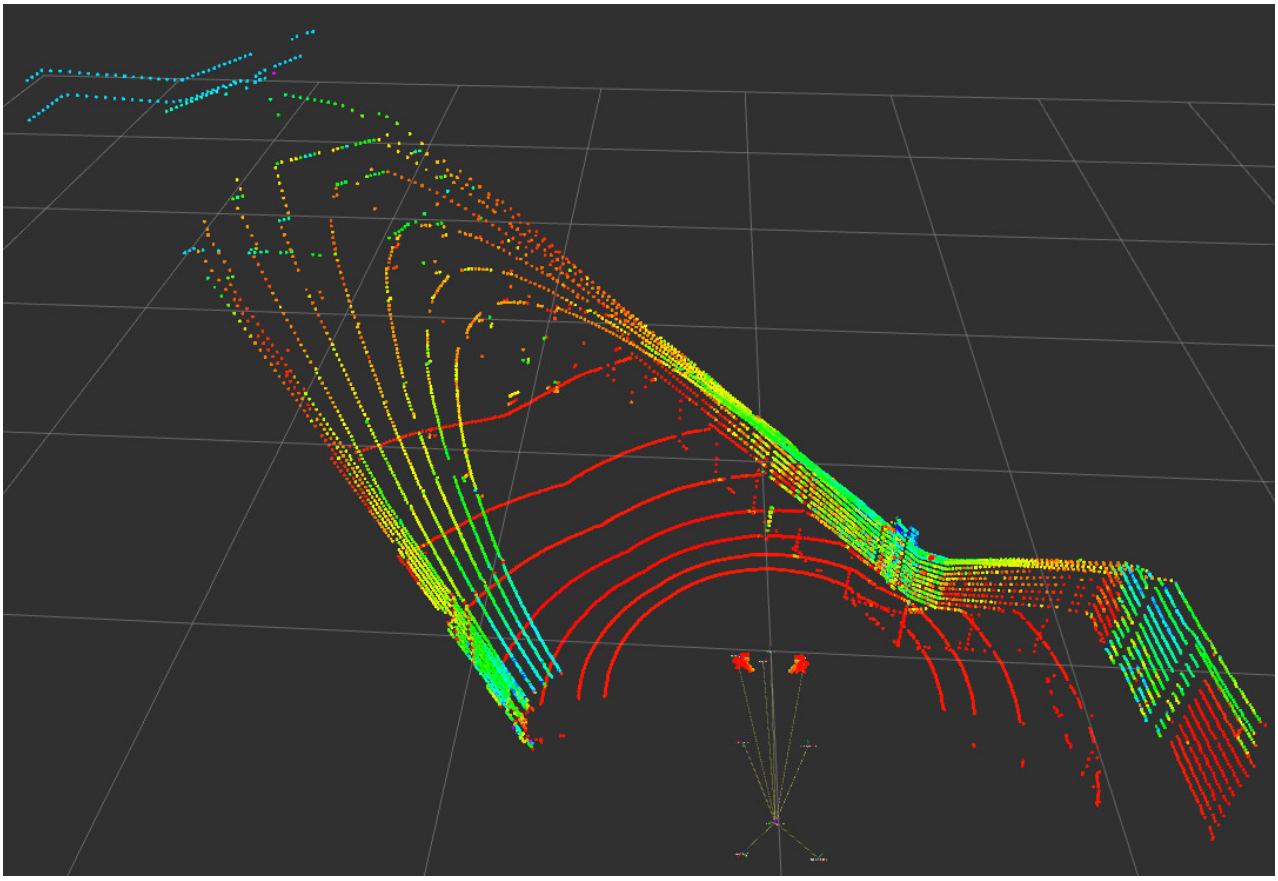
CHALMERS



Local short path generation for autonomous commercial vehicles

Master's thesis in Applied Mechanics

VIKTOR HELLAEUS
YAOWEN XU

Department of Applied Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2017

MASTER'S THESIS IN APPLIED MECHANICS

Local short path generation for autonomous commercial vehicles

VIKTOR HELLAEUS
YAOWEN XU

Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2017

Local short path generation for autonomous commercial vehicles
VIKTOR HELLAEUS
YAOWEN XU

Master's thesis 2017:53
ISSN 1652-8557
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
Pointcloud visualization of the readings from a 3D LIDAR sensor placed on a truck in a mine.

Chalmers Reproservice
Göteborg, Sweden 2017

Local short path generation for autonomous commercial vehicles
Master's thesis in Applied Mechanics
VIKTOR HELLAEUS
YAOWEN XU
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

In recent years, breakthroughs in artificial intelligence (AI) have drawn the attention to the subject from many fields including the automotive industry, where it could become a cornerstone in order to develop fully autonomous vehicles. In the automotive industry the applications for these AI-techniques varies from classification of a vehicle's surroundings to behavioral-reflex approaches that mimics human behaviour. In this master thesis, the capability to navigate a truck in mining environments using neural networks has been investigated, tested and verified in a simulated 3D environment. As input to the neural networks, Light Detection And Ranging (LIDAR) sensors in different configurations has been used. The main focus has been to create an algorithm that can create short paths at a high rate using limited computational power. Consequently, the networks has been tried on Raspberry Pi to prove their capability. Several approaches are proposed using both 2D LIDARs as well as 3D LIDARs. The developed networks are simple, does not require high performance computational units and are able to make decisions at intersections according to a global planner. Apart from the developed networks, a tool-chain for collection of training data, network training and testing in simulated environment is described in detail in the report.

Keywords: neural network, autonomous vehicles, short path generation, decision making, end-to-end learning, machine learning

Acknowledgements

We would like to thank CPAC Systems for the opportunity to dwell into this interesting area during this spring. We also want to thank our supervisor Peter Forsberg for his guidance and support during the thesis as well as all his valuable help with finalizing the report. At last we would like to thank our examiner Leo Laine.

Viktor Hellaeus and Yaowen Xu, *Gothenburg, May 2017*

List of Abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
BNN	Biological Neural Network
Caffe	Convolutional Architecture for Fast Feature Embedding
CNN	Convolutional Neural Network
DIGITS	Deep Learning GPU Training System
E2E	End to End
FCN	Fully Connected Network
GPS	Global Positioning System
LIDAR	Light Detection And Ranging
LSTM	long short-term memory
ReLU	Rectified Linear Unit
ROS	Robot Operating System

Contents

1 Introduction

As technology advances, the possibility to develop autonomous vehicles of different kinds increases. The challenges in this field are huge and ranges from technical issues to ethical aspects. It has the potential to change the automotive transportation sector radically in terms of efficiency, safety and accessibility, which will have big impact on the society. The idea of autonomous vehicles is not novel and can be traced back to the 1930s science fiction literature and later in the 1960s as a problem that has been investigated by the academic research community [1]. Many expert systems [2, 3, 4, 5, 6] that supply solutions needed for fully autonomous vehicles already exist and have been combined in recent years to create systems which can drive autonomous in different environments [7, 8, 9, 10] with great success. The advances in the field between early 2000s until now have been faster than most experts predicted.

A fully autonomous vehicle needs to have the ability to adapt to all kinds of situations while following the traffic rules that may apply. For all of these situations, decisions need to be made at a high pace and without failures. A human driver still outperforms these systems in terms of generality, since many of the current systems still struggle with difficult weather conditions, complicated urban road environments or to simply follow the traffic rules, but this gap is closing fast. A drawback, however, is that the most advanced systems still rely on hardware and sensors that are too expensive to fit in most commercial vehicles. It is therefore of great interest to investigate different approaches for autonomous vehicles to find techniques that are as general as possible and examine which hardware and sensors are crucial for good performance.

Since the human mind is capable to adapt to different weather conditions and traffic situations, artificial intelligence (AI) techniques that resemble the human way of thinking could be a stepping stone in the development of fully autonomous vehicles. Neural networks show the ability to learn similar to the way that can be seen in biological systems [11]. With the advent of deep neural nets, it may be possible to create an AI inspired algorithm that understands the different tasks that driving is composed of [12]. In a recent report from Nvidia [13], a convolutional neural network was used to create an end to end solution, which took camera images as input to directly generate steering angles used in a car with good results. One could argue that even if the results are good, it is still hard to know a priori how the network will act in all situations. Could there be a blind spot in the network where it will generate steering commands that are potentially dangerous? Another approach is to let the network generate plausible local short paths to be acted upon by a decision layer based upon criteria such as safety. This approach enhances flexibility, since the network could be used as a low level system to create collision free paths first in a local environment, then be used by the high level system for safety check or global planning. In [14], a fully convolutional network (FCN) in combination with a long short-term memory (LSTM) recurrent neural network is used to predict future vehicle motion based on camera observations which could be used as paths as described above.

When making maps for control of autonomous vehicles, it is often inefficient to store every detail in the surrounding environment as this would take up large amount of storage. It would also be hard to monitor changes in the environment in great detail in an efficient way. Further on, following a pre-defined track in absolute coordinates would require an accuracy that could be hard to achieve under all conditions as well. Instead, local short paths could be generated at high update rate according to immediate surroundings. Given the sensor data from e.g. LIDAR sensors, a path is then preferably directly generated.

1.1 Purpose and scope

The project investigates how different neural network techniques could be used to calculate short paths at high rate given readings from Light Detection and Ranging (LIDAR) sensors. The main focus is to create a network that could be implemented on low performance hardware with good results. The goal is to have an algorithm which generates short paths fast and with high accuracy, hence enables a truck to run autonomously, provided the paths are followed.

1.1.1 Limitation of scope

- The resulting planner will generate local plausible paths. It will need a global planner that gives instructions of choosing paths that comply with the planned global path at intersections. This global planner will not be a part of this project.

- The intended environment for the local planner will be limited to mines, since the neural network will be trained with data from these environments.

1.2 Planning in autonomous driving

The framework of autonomous driving planning is often divided into several hierarchical layers as in [15, 16, 17]. In one recent survey [15], these four layers are discussed:

- Route planning
- Path planning
- Manoeuvre choice
- Trajectory planning

The route planner is responsible for finding the best route between a starting point and a destination, and could be seen as a global planner which makes high level decisions. The three latter layers concern control with respect to the vehicle dynamics, interaction with traffic as well as handling dynamic obstacles on the path. These sub-systems are often combined together to one unit. More specifically, using the definitions from [15], path planning solves the problem of finding a feasible path that avoids collisions and applies traffic laws that may come into place. Manoeuvre choice concerns the problem of making the best decision when taking both the path from the path planner and the global route into account. Lastly, trajectory planning is the layer that solves the control problem that consists of following the proposed reference path given from the path planner, while taking dynamic constrains of the vehicle as well as the environment into account. Since the idea in this thesis is to use neural networks to generate short paths directly based on raw sensor data, the above layered structure is not applicable. However, it is helpful to have a understanding of these concepts in order to be able to compare our results with other techniques as well as compare the different networks that we present in this report.

1.3 Path planning

Autonomous driving is achieved by utilizing information from the surroundings obtained by different sensors, in order to enable the vehicle to make decisions which move it between two locations without any need of any interventions from a driver [18]. To that extent, the topic of path planning is an essential step to be able to realize autonomous vehicles [19]. The path planner's responsibility is to find a collision-free path from a starting point to a target point in a known environment. Commonly used techniques for path planning are cell decomposition, road-map and potential fields [20, 21, 19]. Path planning could be divided into two different fields, namely global and local path planning. Global path planning requires knowledge of the environment and calculates the full path offline or before execution since the terrain is, here, known beforehand [22]. As mentioned in the introduction, it is hard to monitor changes and often inefficient to store all the data that is needed for the global path planner to be able to complete a path. Even if this could be done for more long lasting changes such as road works, it will still pose problems with more dynamic obstacles such as other vehicles, humans, etc., which are not part of the static map. This problem could be addressed by using a local path planner to handle collision avoidance and calculate paths based on current position and sensor input. A local path planner is usually combined with a global path planner [23]. In this case, the global path planner controls the vehicle from the information that is known in advance, i.e. a static map, in the same way as a route planner does, and the local path planner responds to the surrounding environment of the vehicle's current position based on different sensors.

1.4 Localization

In order to navigate, a vehicle needs to be able to determine its current position and orientation based on readings from sensors [24], which is termed localization. The localization technique can be classified into two main categories [25]:

- Absolute (global) localization - Use beacons, landmarks or satellite signals (e.g. Global Positioning System (GPS)) to obtain the vehicles absolute position.
- Relative (local) localization - Use various on-board sensors (gyroscopes, accelerometers etc) to obtain the position and orientation.

Relative localization utilizes filtered odometry information to obtain a vehicle's relative location to a starting location, commonly known as dead-reckoning [26]. Since the on-board sensors are updated at a high rate they can be used to capture short term deviation from a specified path with ease. But it also suffers from inaccuracies since the underlying models used for the sensor fusion are simplified, which gives rise to errors over time. Absolute localization is used to estimate the position of a vehicle in the global frame, and is independent of time and current location. Therefore it can act as a correction to the previous relative localization. However, absolute localization is not feasible in all circumstances, since it could be cumbersome and expensive to setup necessary infra-structure or because GPS signals could be too weak to use. To reduce these drawbacks, absolute and relative localization are often combined [27].

1.5 End-To-End Learning

As described in the introduction, the need for different path planning and localization techniques could be bypassed by using an end-to-end approach, which generates control input directly to the actuators based on raw sensor input. Using end-to-end learning in the neural network framework means that the optimization done in the training of the neural network will eliminate the need for feature design and hand-tuning of different parameters in dynamic models used for localization and path planning. It could therefore utilize the information available directly from a sensor to calculate a response compared to conventional techniques that first have to classify the environment, then find a plausible path from that information and make a decision based on that. Advantages are that it will require a less computing power while still getting good results [13, 28]. However, the behavior of this approach could be hard to control, since it does not have clear directions.

1.6 Outline of the report

The report is divided into five chapters starting with the introduction. The introduction chapter is followed by a theory chapter regarding the background of artificial neural networks along with brief explanations of the training techniques and network structures used in this project.

In chapter 2, the environments and frameworks where the project has been developed are introduced together with information regarding the modelling of the truck and sensor configurations. This chapter also explains how training data has been collected and processed. To be able to compare the different approaches that has been tested during the project, an evaluation process has been created, which concludes this chapter.

In chapter 3, the different approaches that has been investigated during the project is described in detail.

In chapter 4, the results from the evaluation process for all approaches are shown. The results for each approach are discussed in detail and finally compared against each other based on different criteria. The chapter ends with a general discussion on the project and bring up several thoughts on the differences observed in the results based on the evaluation.

In chapter 5, conclusions from the project are presented alongside a discussion on the future of neural networks in the domain of autonomous vehicles.

2 Artificial neural networks

Just as many other technologies inspired by the nature, artificial neural networks (ANN) started as an attempt to mimic biological neural networks (BNN) by abstracting it to a mathematical model [29]. The aim was to create a model influenced by how the human mind learn solutions. Simple tasks for a human, e.g driving a car, differentiating a viable path from the environment, recognizing obstacles and pedestrians as well as other vehicles, might be very difficult for conventional approaches that are based on rules and instructions. Using an ANN is one way to address the problem. In this chapter, the basics of artificial neural networks and several models related to this work are briefly introduced.

2.1 Biological neurons

The basic unit of a human brain is a kind of cell called a neuron. As seen in Figure 2.1, there are three types of components that are mainly responsible for information processing and transfer: the dendrite, the axon and the soma. The dendrites close to the soma as well as the soma itself receive signals from surrounding axons. These

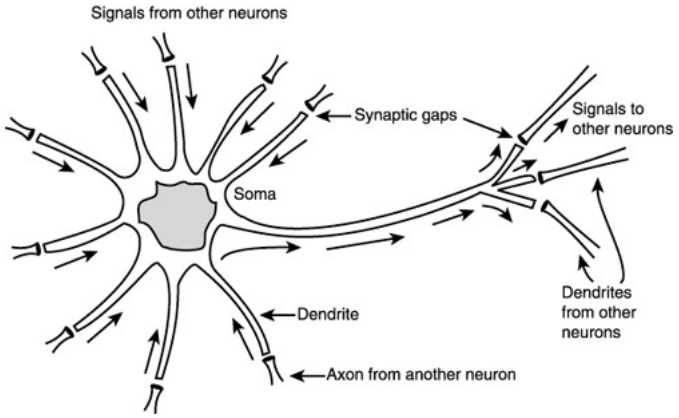


Figure 2.1: *The biological neuron [30]*

signals are processed by the somas and sent to other neurons over the axon. Unlike the numerous dendrites that one neuron can have, neurons generally have only one axon. However, the end of an axon can connect to a large number of other neurons. An axon sends the signal to dendrites or somas of other neurons through a synapse, a structure that can transmit excitatory or inhibitory signals between neurons.

When sufficient excitatory inputs are received by a neuron compared to its inhibitory inputs, the neuron will send out an excitatory output to other neurons. Conversely, the neuron will transmit an inhibitory output, when sufficient inhibitory inputs are received. This output is binary, which means it is either excitatory or inhibitory, no intermediate levels.

The learning process in the neural level can be explained as the process of neurons changing the weights on the input from connected neurons. It is worth mentioning that there is still a lot of work left to fully understand the process of learning, but the pattern described above is what the artificial neurons are initially based on.

2.2 Artificial neurons

As the counterpart of biological neurons, artificial neurons use mathematical equations to model the functions of biological neurons. The model introduced below is the most commonly used model proposed by McCulloch and Pitts [29].

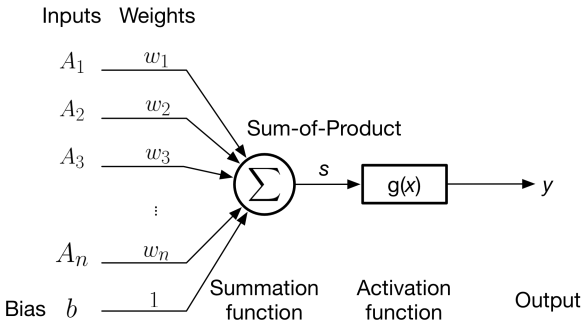


Figure 2.2: *The artificial neuron*

As shown in Figure 2.2, similar to the dendrites of a bio-neuron , an artificial neuron has inputs. Every input is given a weight and sent to a summation function (Equation 2.1). In addition to the inputs, there is usually a bias term, denoted by b in Equation 2.1.

$$s = \sum_{i=1}^{n} w_i A_i + b \tag{2.1}$$

The result of the summation function is passed to an activation function that controls the state of the neuron based on the input.

2.2.1 Activation function

An activation function g can be described by Equation 2.2, where y is the output of the neuron and s is the summation function.

$$y = \mathrm{g}(s) \tag{2.2}$$

The type of activation function varies from different applications. As stated above, the activation function is a binary function in biological neurons. However, that is not practical in some applications. Instead, differentiable non-linear functions are generally used in artificial neurons to give a graded output. Historically, the most common one is the sigmoid function (Equation 2.3).

$$\sigma(s) = \frac{1}{1 + e^{-s}} \tag{2.3}$$

Though the sigmoid is not a binary function, it is still motivated by biological neurons. One can interpret the output of the sigmoid function as the "firing" frequency or probability of a neuron output given its input [31]. Another popular activation function related to sigmoid is the hyperbolic tangent function, or tanh function (Equation 2.4).

$$\tanh(s) = 2\sigma(2s) - 1 = \frac{2}{1 + e^{-2s}} - 1 \tag{2.4}$$

As seen in Equation 2.4, the tanh is basically the sigmoid function shifted 1 units downwards, squeezed 2 times along the x-axis and stretched 2 times along the y-axis. Therefore, it has larger gradients around the origin and a non-zero output when strongly negative input is given, compared to the sigmoid.

In recent years, the rectified linear unit (ReLU) activation function has been vastly adopted in various applications. It is defined as Equation 2.5.

$$\mathrm{g}(s) = \max(0, s) \tag{2.5}$$

Compared to the sigmoid and tanh, the most important benefit is that ReLU avoids the vanishing gradients problem. The gradients of the sigmoid and tanh vanishes away from the origin quickly, which is not ideal for gradient-based learning methods. The ReLU activation function does not require any expensive operations in implementation either as sigmoid and tanh functions that requires exponentials in implementation. However, ReLU has the so-called "dying ReLU" problem. When $s < 0$, the gradients and output from ReLU are both 0, which means gradient-based learning methods will never change the weights and the neuron will never fire

again. To mitigate this issue, it is important to set a proper learning rate and monitor the amount of neurons in the network that are "dead" after training. Other ways of solving the "dying ReLU" problem include the leaky ReLU. It introduces small gradients for $s < 0$ as seen in Equation 2.6. The constant α is a small positive number that prevents the activation function from being zero at negative inputs.

$$g(s) = \begin{cases} \alpha s & (s < 0) \\ s & (s >= 0) \end{cases} \tag{2.6}$$

It is worth noticing that the ReLU is not differenciable at $s = 0$. To enable gradient-based learning methods during training of nets, the derivative at $s = 0$ is defined as 1.

2.3 Topologies of neural networks

An artificial neural network consists of an organized set of connected artificial neurons, therefore different ways of organizing the neurons, in other words, different topologies of the network have been proposed. In this section, some types which are applied in this work are briefly introduced.

2.3.1 Feed-forward neural networks

One of the simplest and earliest artificial network topology is shown in Figure 2.3. It is called feed-forward neural network, since data passes through each layer in only forward direction, from left to right, without any feed-back.

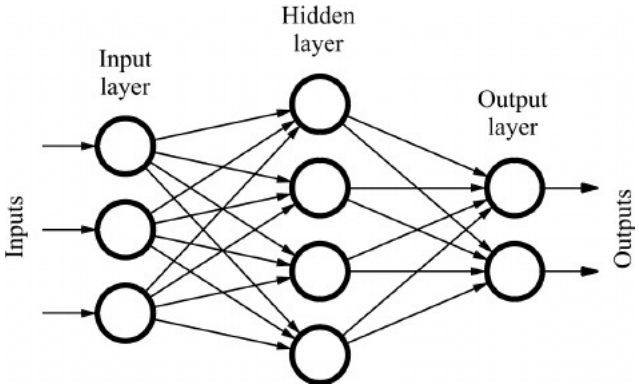


Figure 2.3: *Feedforward neural network [32]*

A feed-forward neural network usually consists of three different types of layers: input, hidden and output layer. The neurons in each layer are called nodes. In an input layer, the nodes do not usually perform any computations but associate the data. Nodes in the hidden layers get data from the input layer and associate it with different weights, then the activation function generates a response to the next layer. These nodes in the hidden layers are isolated from the environment during deployment of the net, and therefore called "hidden". The last layer is the output layer, which is associated with the output data. Unlike the input layer, the nodes in an output layer may perform computations as well. It is worth mentioning that when there are more that one hidden layer, the network is called deep network, corresponding to a subset of machine learning which is called deep learning.

2.3.2 Convolutional neural networks

One popular type of feed-forward neural network is called convolutional neural network (CNN). It assumes that the input has spatial relations which drastically reduces the number of parameters in the network.

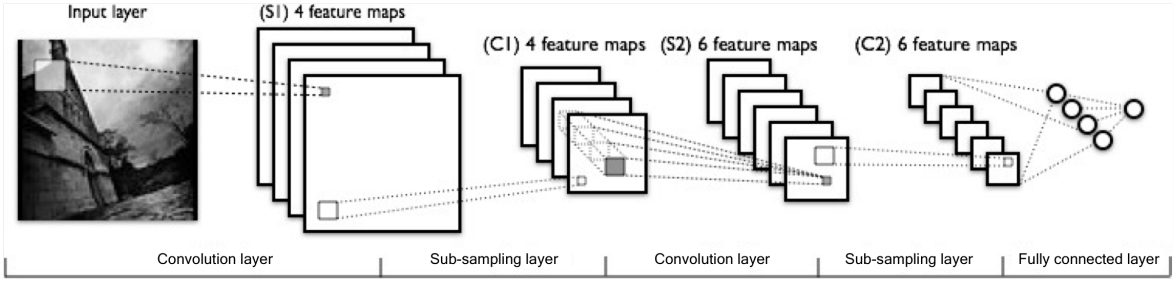


Figure 2.4: *The lenet [33]*

An example of convolutional neural network is shown in Figure 2.4. As seen in the example, a CNN commonly has three types of main layers: convolution layer, sub-sampling layer and fully connected layer. In addition, it also has an auxiliary layer that works as an interface to the data, the input layer.

Convolution layer In a convolution layer, the neurons work like filters. These filters are applied to the input data convolutely and generate feature maps, containing matrices of the neuron activation. The convolutional layer has many hyperparameters controlling the output from the layer, including the number of features, kernel size, stride and padding. The number of features sets the number of filters we want to use on the input. Each filter in the layer will be used to abstract one different feature of the input. The kernel size sets the size of the filter that will slide over the input. By setting the stride, the step length that the filter is moved over the input is changed. Stride is used to reduce the output volume from the layer. In many cases, it is useful to pad the input volume with zeros to enable preserving of the spatial size of the input. By changing the padding parameter the border of zeros padded around the input is changed.

Sub-sampling layer The sub-sampling layer, also known as pooling layer, down-samples the output from the convolutional layer to reduce the spatial size of the representation, thereby reducing complexity and preventing overfitting. This is done by sliding over the input as in the convolutional layer to downsize the input by a certain factor. The difference for the sub-sampling layer is that the kernel of a pooling filter use various functions aiming at reducing the matrix size with static filter parameters as opposed to the filters in the convolutional layers, which are changed during training. The most general kernel is called max pooling. For example, a 2×2 max pooling kernel takes a 2×2 slice of the input in every step, then finds the maximum value of the four input elements and passes it to the next layer. In this case, the operation reduces size of the input by a factor of four.

Fully connected layer After several convolution and sub-sampling layers, the features extracted are sent to a fully connected layer to generate the final output. As the name suggests, in a fully connected layer, each node connects to all the output in the previous layer.

In short, even though there are various models of CNN, the general major steps are:

- Apply several filters to the input data convolutely to form feature maps.
- Subsample the input data or feature maps.
- Connect the last feature maps to a fully connected feed-forward neural network.

The first and second step can be arranged repeatedly until enough high level features are extracted.

2.3.3 Training of neural network

The training of a neural network is done by finding the correct weights in each neuron. Some mathematical methods to optimize the weights are briefly introduced below. It is worth noting that training is not limited to only adjust the weights, there are also methods that can change the topology of the networks, like Neuroevolution [34].

Stochastic gradient descent

One of the basic weight optimization method is the stochastic gradient descent. It uses the gradient information of the error function $\mathrm{E}(w_n)$ to find the direction with the fastest error decrease. It can be described by Eq. 2.7,

$$w_n(k+1) = w_n(k) - \alpha \frac{\partial C}{\partial w_n} \tag{2.7}$$

where C is the cost function and α is called learning rate. It controls the amount of the current gradient to be used for correction, hence adjusts the step length. To reduce the noise level, the average gradient of a batch of training data is usually used (Eq. 2.8),

$$w_n(k+1) = w_n(k) - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial C_i}{\partial w_n} \tag{2.8}$$

where m is the mini-batch size. Another useful technique to attenuate the oscillation is to add a term called momentum (Eq. 2.9),

$$w_n(k+1) = w_n(k) - \frac{\alpha}{m} \sum_{i=1}^{m} \frac{\partial C_i}{\partial w_n} + \mu \Delta w_n(k) \tag{2.9}$$

where μ is a coefficient used to determine the magnitude of the previous gradient and $\Delta w_n(k)$ is the gradient in the previous iteration. In each iteration, the previous gradients can be used by the algorithm to find the minimum of the objective function more efficiently.

Backpropagation

The Backpropagation algorithm is used to find the optimal weights in a neural network. Together with a gradient descent algorithm it makes up the backbone used to train a network. The backpropagation algorithm is described below:

- Input. The input x is sent in through the input layer $l = 1$. An activation function g^1 is applied to the input and calculates a set of activations a^1.

- Feedforward. In each layer $l = 1, 2, 3, \cdots, N$, the data is propagated $s^l = w^l a^{l-1} + b^l$ and fed through the activation function $a^l = g(s^l)$.

- Calculate output error E. The error at the output layer denoted E^N can be expressed as

$$E^N = \nabla_a C \odot \mathrm{g}^{'}(s^N), \tag{2.10}$$

  where C, $\odot$ and g are the cost function, Hadamard product and the activation function respectively.

- Backpropagate the error. For each layer l the error can be calculated as

$$E^l = ((w^{l+1})^T E^{l+1}) \odot \mathrm{g}^{'}(s^l). \tag{2.11}$$

  By multiplying the error from the subsequent layer $l+1$ and the weights between the layers l and $l+1$, the error at layer l is calculated. In other words, the error at layer $l+1$th is propagated backwards.

- Generate gradients. The gradients of the objective function at each layer for the weights and the bias terms are expressed as

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k E^l_j \quad \text{and} \quad \frac{\partial C}{\partial b^l_{jk}} = a^{l-1}_k, \tag{2.12}$$

  where j is the index of the neuron in layer $l-1$ while k is the kth neuron in layer l.

From the gradients, the weights can be updated using the techniques described in Section 2.3.3.

2.4 Current development of neural networks

Since the advent of deep learning in 2012, the method has gained a lot of attention and many believe that the number of AI inspired algorithms will keep increasing in many applications. This has lead to new discoveries, development of specific deep learning hardware and attempts to enhance older ideas of network structures as well. Examples include the studies of different types of recurrent neural networks, fully convolutional networks and neuro-evolutional networks, to name a few. Many of the techniques could be applied to the problem approached by this project and probably yield good results as well. However, the computational power for these deeper nets, both in the training stage and deployment stage has not been available. Despite the success of these complex methods, it is also interesting to investigate the bare minimums required to solve the problem at hand.

3 Method

The focus of this thesis was to investigate how neural networks could be used to generate short local paths using LIDAR sensors. Consequently, the workflow revolved around sequentially collection of training data, construction of neural networks, training and evaluation. The workflow is illustrated in Figure 3.1. Instead of pursuing the final purpose directly, i.e. generating a short local path from a 3D laser scan, it was deconstructed into smaller problems. This was achieved by creating a modular development environment where each module could be substituted by a new one. This approach also made it easier to pivot between different ideas that we wanted to try out during the project and gave us a better understanding of the different parts that the main problem consists of. The remains of this chapter will introduce the tools and frameworks that were used to create the short path generators.

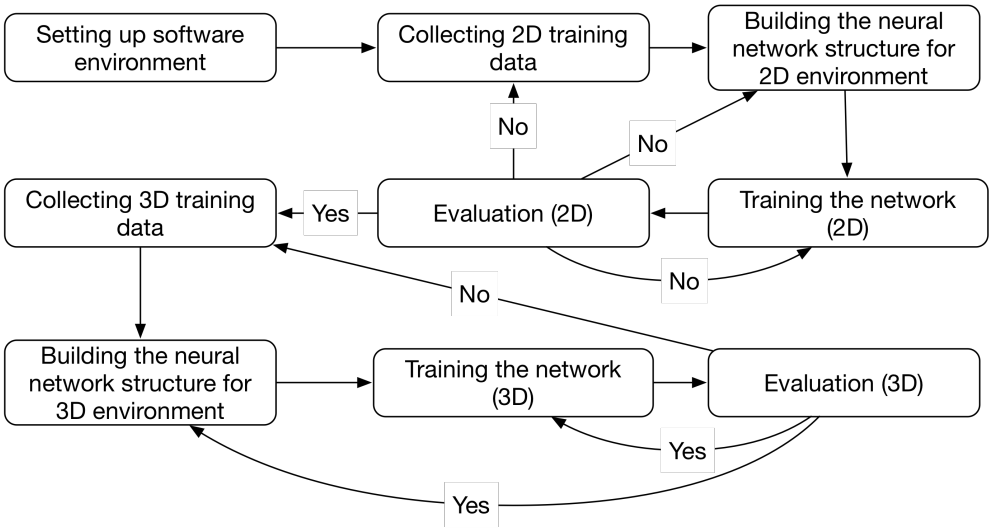


Figure 3.1: *The principal workflow used to create the short path generator*

3.1 Development environment

The development environment was required to allow for easy integration of the different algorithms that were developed, and have the ability to carry out simulations to evaluate them. These requirements were found in Robot Operating System (ROS) [35] and Gazebo[36]. ROS serves as an interface between the developed modules and the simulation environment in Gazebo. It is able to handle the communication between the modules needed for both collecting training data, deployment of the neural networks and could also be used in the field experiments. As seen in Figure 3.1, the project started with setting up this software environment. After the setup of the environment was completed, 2D LIDARs were used to develop the first networks. Using only 2D sensor data initially, had the advantage of creating faster development loops, since the size of the neural networks would be reduced because of the smaller input size, resulting in shorter training times. It also had the benefit of lower performance requirements for sensor simulations, easier data manipulation on training data etc. It was also presumed that an successful approach using 2D sensor data probably would success using 3D LIDAR data as well, since similar information could be extracted from the 3D scan that is present in the 2D scan.

3.1.1 Simulation environment

As mentioned above, Gazebo provides a simulation environment and enables accurate and efficient simulations in a 3D environment, and it also has good support for simulations for the types of sensors used in this project. To mimic the environment in a mine, tracks are created in Blender [37] as illustrated in Figure 3.2.

The vehicle in the simulations were a four wheeled truck using Ackermann steering, the same type used in a

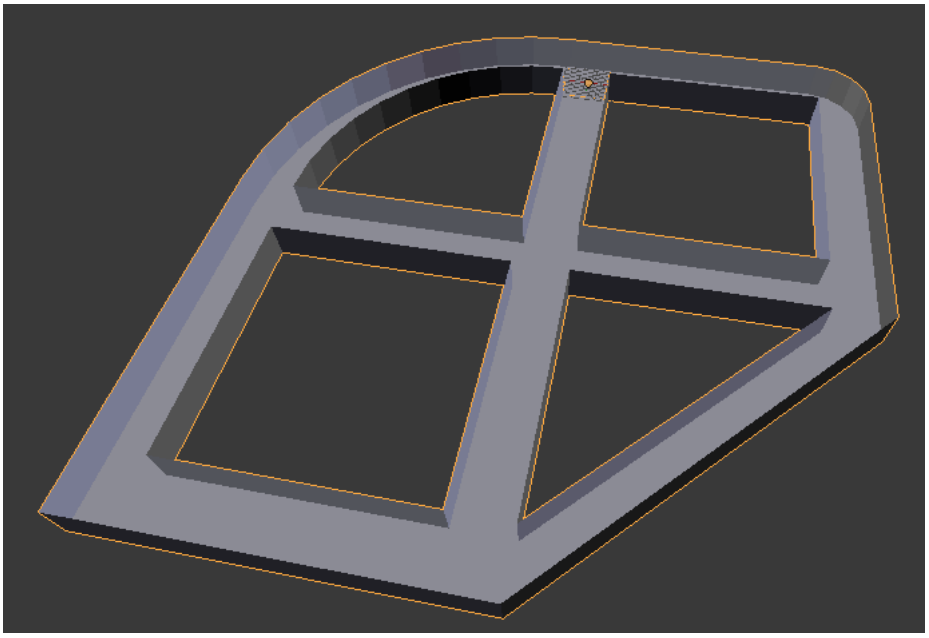


Figure 3.2: *3D model of the training track used in the project.*

previous project [38]. Since neural network uses labeled training data to learn a function, quality data is key in order to succeed. In this case that meant examples of sensor data need to be paired with proper steering angles or similar. Therefore, simulated driving recordings had to be created. These recordings were created by first driving the truck along a track using a gamepad. However, this approach does not scale very well, and since the learning is dependent on the amount of data, especially for complex functions, driving the truck manually took too much time and effort. To solve this issue, path followers were developed to follow recordings generated from manual driving. Hence the process was automated, and training data could be generated once one manual recording was created. The simulation environment was also used during the evaluation of the different networks. The flowchart in Figure 3.3 shows the data flow during simulations for manual driving with the gamepad, collecting training data and evaluation of the neural networks.

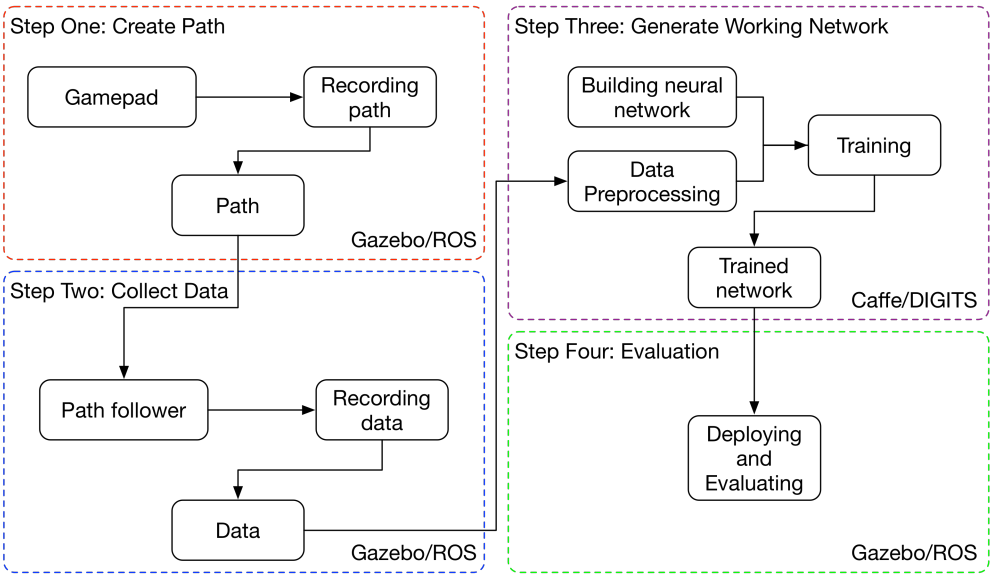
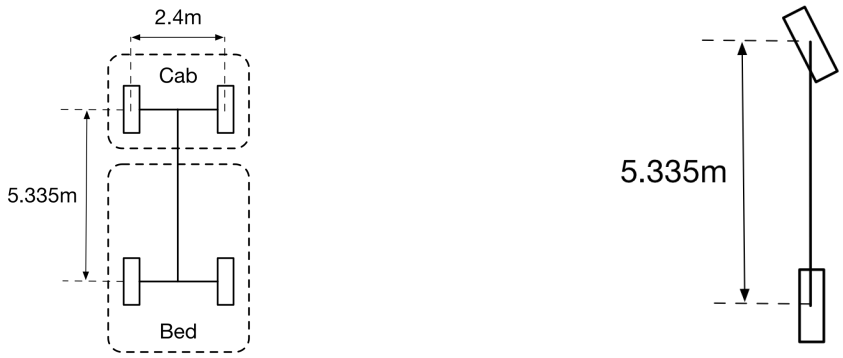


Figure 3.3: *The data flow for development*

3.1.2 Truck model and sensor placements

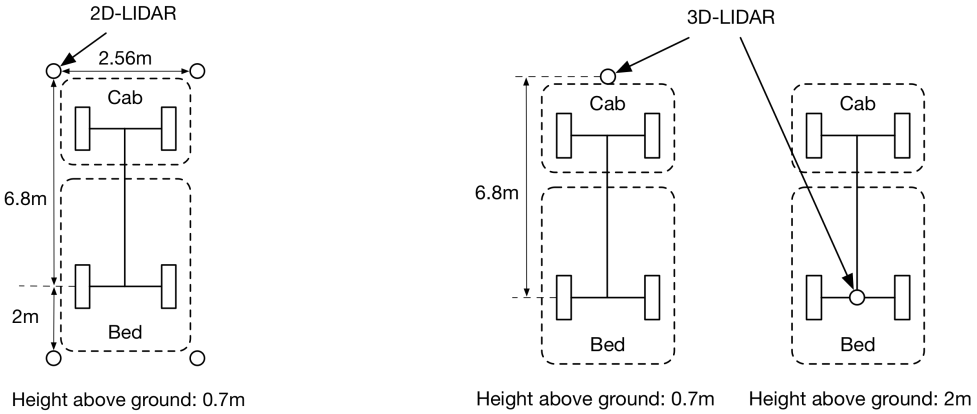
There were two different truck models used in this project. The one with Ackermann steering, as shown in Figure 3.4a, was used in the simulation environment. However, a bicycle model shown in Figure 3.4b was used in calculations for its simplicity. Every local short path is a curvature calculated from the current steering wheel angle using the bicycle model. The neural network was expected to be insensitive to small derivations between vehicle model and real vehicle, since the fast update rate yielded quick feedback to the network.



(a) *Ackermann model* (b) *Bicycle model*

Figure 3.4: *Truck models*

In terms of sensors, various sensor configurations were experimented with in this project. As seen in Figure 3.5a, in 2D cases, the 2D LIDARs were located in the four corners of the truck and merged together using [39]. The merging of these sensor resulted in one 360-degree scan with its origin in the center of the rear wheel axle. In 3D cases, two different configurations were used, as seen in Figure 3.5b. In the first configuration on the left, the 3D LIDAR was located in front of the cab center. The sensor covered 180 degrees of the front view, but did not have any information on both sides and the rear of the truck. The second configuration on the right used one 3D LIDAR 2m above the center of the rear wheel axle. It was not feasible in the real truck, but it resembled the merger used in the 2D case. One could obtain the same coverage by merging four 3D sensors placed in the four corners of the truck. This configuration was used mainly because the simulation of four 3D LIDARs took a lot of computational power and was therefore unfeasible on our computing platform.



(a) *2D sensor* (b) *3D sensor*

Figure 3.5: *Sensor placement on the truck model*

To enhance the realism of the simulations, sensor noise was added during the simulations according to the specifications of the simulated sensors. The added noise was Gaussian distributed with a mean of zero for both of the sensors. The standard deviation was set to 0.01 for all the LIDAR configurations.

3.1.3 Neural Network Frameworks

In order to build and train the network as described in Section 2.3.3, the deep learning framework Caffe[40] was used. It is widely used in both academia and industry, and is said to be a deep learning framework made with expression, speed and modularity in mind, and its BSD 2-Clause license [41] allows for further industrial applications. On top of Caffe, the Deep Learning GPU Training System (DIGITS) was used [42]. DIGITS simplifies management of training data and network models. It also enables real time monitoring of training and visualizations of the neural network structure.

3.2 Collecting training data

Collecting training data was done by first creating reference paths in the Gazebo simulation environment using a gamepad to control the vehicle and record the position of the truck in global coordinates as mentioned in Section 3.1.1. These recordings could then be used as references to create training data using a path follower. Two different path followers were developed with different characteristics.

3.2.1 Simple path follower

The first path follower takes two consecutive points on the reference path and the truck's current position as input in each iteration. The distance from one of the points and the truck's current position is then calculated. This distance is used as an error and control the steering angle of the wheels of the truck. Thus the path follower will generate a large steering angle if the truck is far away from the points on the reference path. To determine if the truck should steer to the left or right, a line between the two consecutive points are drawn and from this information, it can be concluded if the truck is either on the left or the right side of this line. If the error between the truck and the points is under a certain threshold, two new consecutive points are selected. The error is then passed through a P-controller, which is tuned to give reasonable steering angles. This type of path follower gives rise to oscillations around the reference path, since even if the truck is right on the reference path, it will generate a steering angle greater than zero, therefore it will always have a distance between the truck and the reference point. The magnitude of the oscillations can be changed by using different error thresholds, where the new point should be selected. The reason for using this path follower was to introduce deviations in the training data, since it is important to have a diverse training set to prevent overfitting, which could be a problem since the amount of recorded reference paths were low.

3.2.2 Pure Pursuit path follower

The second path follower used was pure pursuit. Pure pursuit is a geometric path follower that calculates the steering angle by creating the curvature of an arc that connects the rear axle with the current reference point on the reference path. It simplifies the four wheel Ackermann steered vehicle in Gazebo to a geometric bicycle model using the geometric relationship

$$\tan\delta = \frac{L}{R} \tag{3.1}$$

where L is the wheelbase, R is the radius of the circle that the rear wheel will follow for a given steering angle δ as seen in Figure 3.6.

The current reference point is determined by a look ahead distance l_d ensuring that the truck is following a point that is in front of the truck. By changing the look ahead distance, the characteristic of the path follower will change from unstable (short look ahead distance) to poor tracking (large look ahead distance). If the coordinates of the vehicle's base axle, front axle and the reference point are known, the angle α between the reference vector and the heading vector can be calculated. When α is known, the steering angle needed to intersect the reference point can be determined from the curvature κ, which is calculated by applying the law of sines to the geometrical relationships seen in Figure 3.7.

$$\kappa = \frac{2\sin(\alpha)}{l_d} \tag{3.2}$$

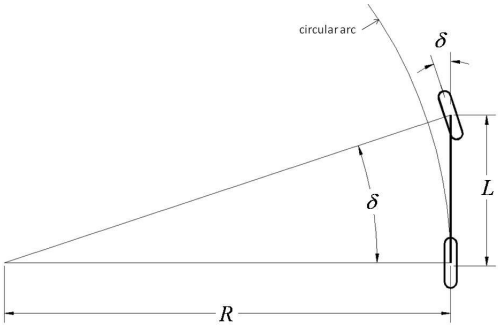


Figure 3.6: *Bicycle model*

By combining Equation 3.1 and Equation 3.2, the control law is given as

$$\delta = \text{atan}\left(\frac{2L\sin(\alpha)}{l_d}\right) \tag{3.3}$$

At each iteration, the path follower searches for the next point and updates the reference point if it is in the look ahead distance.

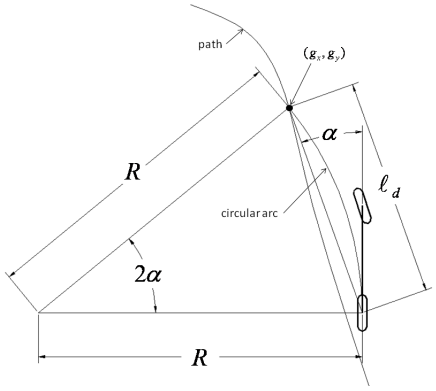
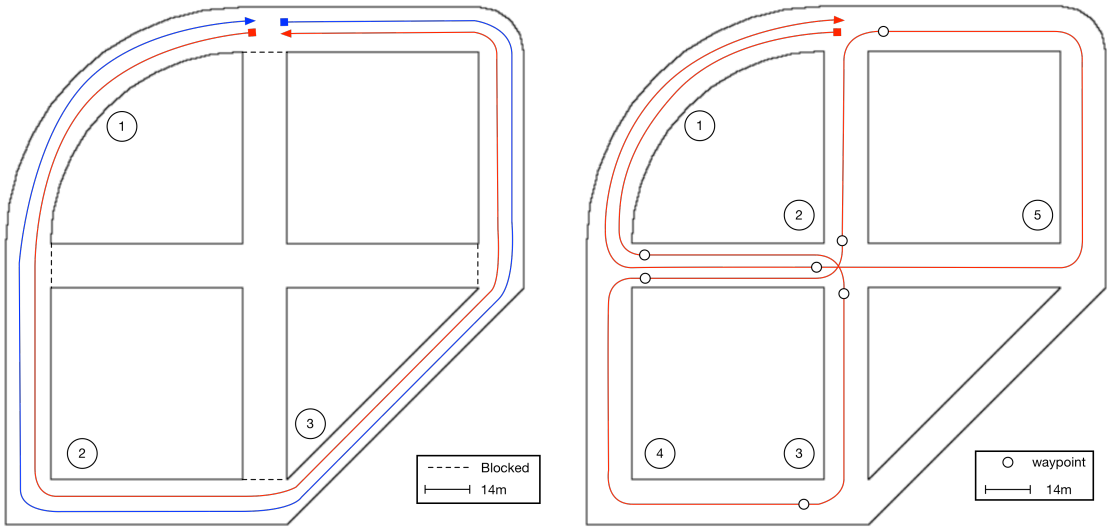


Figure 3.7: *The geometry in pure pursuit*

This path follower was much smoother, hence the natural deviations that the first path follower created from oscillation around the path disappeared. Instead of creating natural deviations in the original data, the recorded data was shifted by rotating the sensors readings and the corresponding labels that was recorded during training. This technique allowed for control of the deviations and assured that all training data expresses correct behaviour in all situations. This could not be said about the first path follower, since its steering commands is more random because of the oscillations.

3.2.3 Training track and path

It is essential to have variations in the training data to ensure good generality in the developed nets. However, the efficiency of collecting the data has to be considered as well. Bearing these two factors in mind, a compact, feature-rich track is designed, shown in Figure 3.8. To accommodate different training requirements, two versions are built. The one with intersection blocked is shown in Figure 3.8a, and the other with open intersections is shown in Figure 3.8b. The features in both track are described in Table 3.1. The red and blue line in the track are the recorded paths, the paths are designed to explore all the features in a short range. There are two opposite paths on Figure 3.8 to reduce the bias of only turning into left or right at corners.



(a) *Training track without intersections* (b) *Training track with intersections*

Figure 3.8: *Training tracks*

Table 3.1: Features for the training tracks

Feature	Description
1	Soft turn
2	90° turn
3	45° turn

(a) Features in the track without intersections

Feature	Description
1	Soft turn
2	X crossing, middle to right
3	T crossing, middle to right
4	90° left turn
5	T crossing, left to middle

(b) Features in the track with intersections

3.2.4 Labels and recording

Since many approaches to find local paths has been tested during the project, different labels for training the networks has been collected during training. These approaches will be described in detail in the following sections. The data was collected in ROS-bags from the path follower.

3.3 Neural network structure and training

The collected data was converted from ROS-bags to a database to meet the requirement that the neural network framework needed. Transforming the data after the recording also provided an easy way to modify the data if needed. In the same way, different approaches required different labels, each investigated approach also explored the results of different network structures. This meant that for each approach, the number of hidden layers, different activation functions and number of neurons was changed and the result observed. During training stochastic gradient descent has been used since it can calculate the loss for a few training examples at a time and still lead to convergence fast. The batch size, learning rate and momentum were adapted as well for the different structures to prevent the optimization during training from ending up on local minimums or slow convergence rate.

3.4 Evaluation

After the training, the network was tested in the simulation environment. The output was evaluated in different ways to compare the performance. The sequence in Figure 3.1 was repeated until an acceptable result was

gained or the approach was deemed as not suitable for further development. The different approaches are also benchmarked against each other in terms of

- Complexity of the network
- Stability of the output from the network
- Driving behaviour of the network

To ensure a fair evaluation among the approaches, all networks has been given the same amount of training data and deployed at an update frequency of 10 Hz. However, it must be noted that due to the available hardware, the simulation of the 3D LIDAR sensor was only running at 3 Hz, while simulation of the 2D LIDAR was at 10Hz. The evaluation track is illustrated in Figure 3.9. Like the training track, it was designed to cover as many as possible different features in a relative short range for efficiency. The list of different features is given as Table. 3.2.

The deployed networks, which have no ability to make decisions at intersections, should finish the first part of the evaluation track represented by the red line. The networks that have ability to navigate at intersections, were supposed to finish the full track.

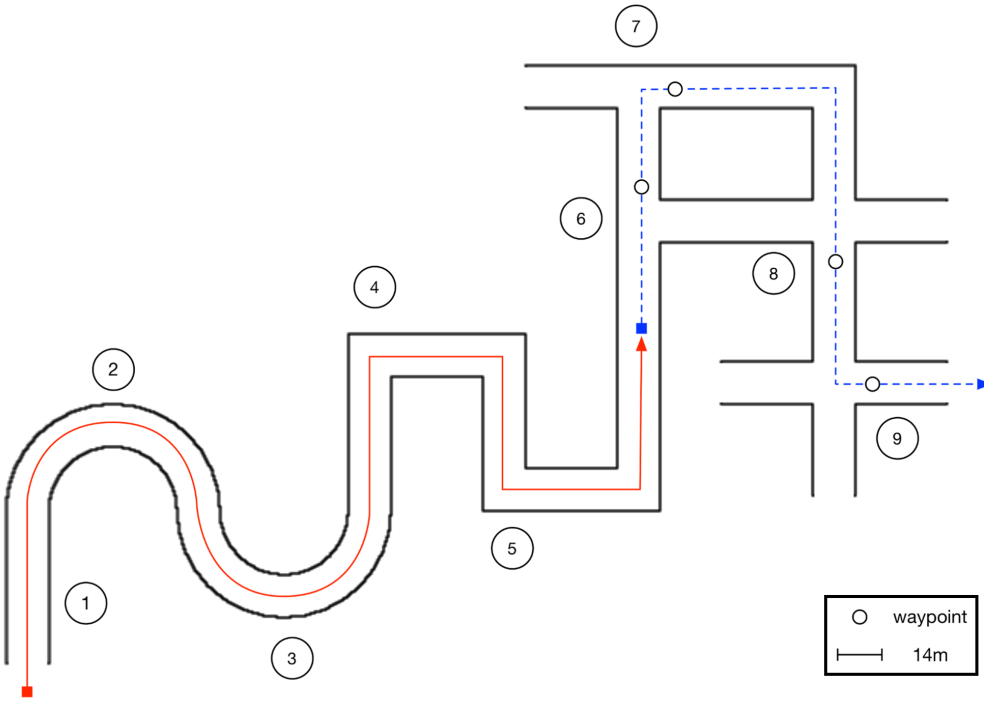


Figure 3.9: *The track used in the evaluation*

Table 3.2: Description of regions with different features.

Feature	Description
1	Straight corridor
2	Right soft turn
3	Left soft turn
4	Right 90° turn
5	Left 90° turn
6	T crossing, left to right
7	T crossing, middle to right
8	X crossing, straight
9	X crossing, turn

4 Approaches

As mentioned in Chapter 3, one of the goals of this work was to explore how and if neural networks are capable of navigation and short path generation using LIDAR sensors. Therefore, multiple approaches have been developed and evaluated. These approaches are described in this chapter to give a general idea on how neural networks can be used to handle these types of problems.

4.1 E2E regression 2D

The first method used an end-to-end regression network that generate a curvature directly from a merged LIDAR sensor reading as described in Section 3.1.2. Hence, the network output could take any real number. The goal of this network was to create a network that could find collision free paths in corridors using 2D LIDAR sensors. Training data was created as described in Section 3.2 using a path follower while recording the merged laserscans and the corresponding curvature, κ. The curvature was calculated from the average of current steering angles of the wheels of the vehicle from

$$\kappa = \frac{\tan(\delta)}{L} \tag{4.1}$$

where

$$\delta = \frac{\delta_{\text{left}} + \delta_{\text{right}}}{2} \tag{4.2}$$

In the merged laserscan, all four sensors was used to utilize all available information around the truck, resulting in a network that is not as prone to cutting corners as the case where only front sensors are used, since the detection of the walls and obstacles in the vicinity of the truck's rear side gets improved in this case as seen in Figure 4.1.

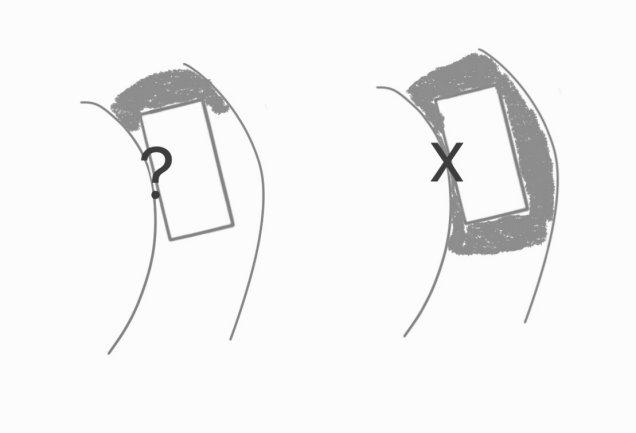


Figure 4.1: *Sketch of merged front sensor readings compared to front and back merged sensor readings*

Since the recorded data solely yielded training examples where the error between the prerecorded path and truck's heading and position is quite small, the data was modified to enhance the result of the network after training as mentioned in Section 3.2, by adding a random yaw rotation, γ, to the recorded yaw angle of the truck. Modifying the data in this way should make the resulting network more inclined to stay in the center of the track and recover when heading against a wall or an obstacle. The rotation was done around the center of the merged laserscans and a new curvature was calculated from that position by adding the yaw angle γ to the steering angle. The added yaw angle, γ, was uniformly selected in the interval -30 degrees to 30 degrees. The interval was selected as large as possible while still making sure that the truck would not hit any walls after the rotation.

When all training data was collected and modified the network was trained and could be deployed in the simulation environment for evaluation. The network consists of two hidden layer with 512 and 23 neurons respectively and uses the ReLU activation function as seen in Figure 4.2.

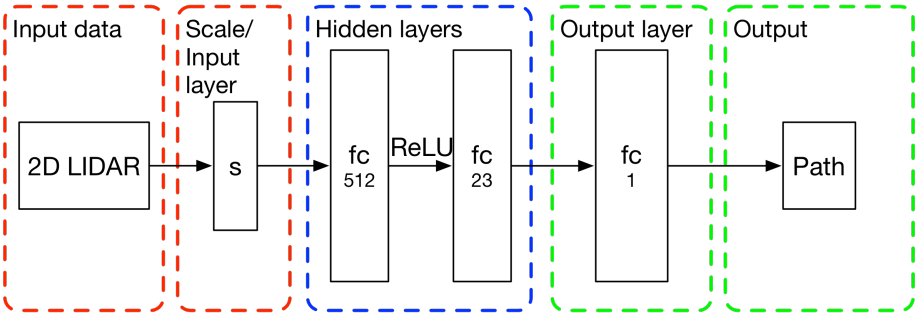


Figure 4.2: *The network structure in the E2E regression approach using 2D LIDAR sensor data as input*

4.2 E2E classification 2D

A drawback of the first network is that the steering commands from the network could be any real number. Hence, it cannot guarantee to only generate curvatures that are feasible for the truck to follow. Another shortcoming of the first proposed neural network structure is that the output size of the network is only one curvature at each time step, which means that a global planner will have no ability to select among multiple different paths in the case of an intersection or similar. In an attempt to solve these shortcomings, the output was changed from one continuous curvature to a set of 23 curvatures corresponding to steering angles from $-\pi/6$ to $\pi/6$. The number of curvatures can be any number which is sufficient. A higher number will result in a more smooth output at the cost of a more complex network. The curvatures were evenly distributed in this range, with the endpoints set as the maximum steering angle of the real truck. The procedure of collecting data was then repeated, but instead of storing the calculated curvature, using Equation 4.1, the curvature was mapped to the closest curvature, κ_i, in the set $\boldsymbol{\kappa}$ where

$$\boldsymbol{\kappa} = \begin{bmatrix} \kappa_0, & \kappa_1, & \dots, & \kappa_{22} \end{bmatrix} \tag{4.3}$$

Thus the labels for the sensors sensor reading were rendered as vectors where all but one entries were zero, which instead was set to one. The change of output meant that the network turned into a classification network instead of a regression network. Consequently, the output of the network could be seen as a probability distribution over the different curvatures. The recorded data was augmented in the same way as in Section 4.1 to enhance the performance of the deployed network.

The network structure in this approach was reduced by one hidden layer as shown in Figure 4.3, and since the output of this network should represent a categorical distribution, the softmax function given as

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \tag{4.4}$$

was applied on the output layer which ensured that the output summed up to one.

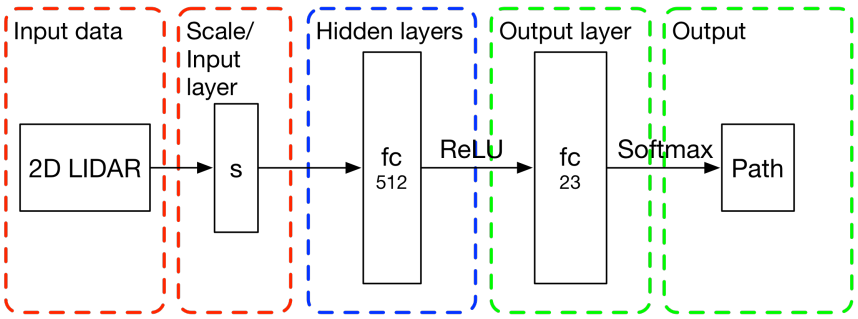


Figure 4.3: *The network structure in the E2E classification approach using 2D LIDAR sensor data as input*

4.3 E2E regression 3D

Even though the simplicity of using 2D LIDAR is appealing, it has its limits, since the sensors are placed horizontally. Therefore, the truck is limited to sensor readings in a 2D plane, which means that it cannot discover obstacles that are outside that plane. The sensors will also be sensitive to roll and pitch perturbations of the truck that can occur on uneven roads. Using a 3D LIDAR, which can discover obstacles in different heights and provide useful data even if the truck is bumping because of the profile of the road surface, could solve these issues.

The same end to end approach as in Section 4.2 was tested using a 16-layer 3D LIDAR sensor. These tests were carried out with two different sensor placements to examine how the sensor placement affects the performance of the network. In the first configuration, the 3D LIDAR was placed in the front with a 180-degree view angle. In the other configuration, it was placed in the center of the truck yielding a 360-degree. The second one will require merging of several sensors, since the truck itself occludes the sensor if it would be placed there. However, these tests were performed in simulations, which means that the visual of the truck could be disabled for the sensor yielding a full 360-degree vision with only one sensor. The structure of the network was the same as in Section 4.2, apart from the change of size of the input layer as seen in Figure 4.4.

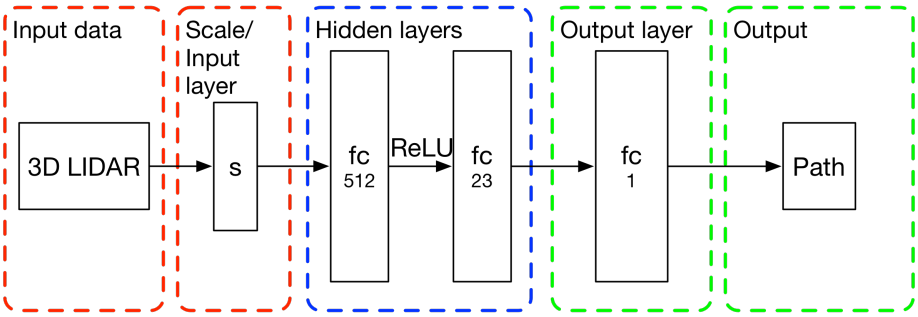


Figure 4.4: *The network structure in the E2E regression approach using 3D LIDAR sensor data as input*

4.4 E2E classification 3D

Similar to Section 4.3, this approach is a repetition of its 2D counterpart introduced in Section 4.2. The aim of testing this approach is to examine whether 3D information would improve the accuracy of classification. Moreover, for practical applications, a 2D scan is not enough since the truck need information from more than a slice of the space to be able to safely move around as mentioned in the previous section. In this approach, only the 360-degree 3D LIDAR configuration were considered.

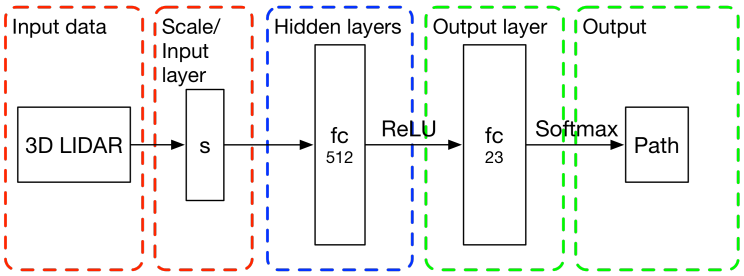


Figure 4.5: *The network structure in the E2E classification approach using 3D LIDAR sensor data as input*

4.5 E2E classification with "Compass" 2D

To approach the decision making problem at intersections, a "compass" was added to the input of the neural networks. As seen in Figure 4.6, a waypoint was placed in the global coordinates at the intersection, from which

the direction of the vector, pointing from the center of rear axle in the truck to the waypoint, is determined. Then along with the heading of the truck, the angle δ between the two directions can be calculated. After that, the angle was fed to the neural network as an image as shown in Figure 4.7. The input image consists of two parts, the "compasses" on the top part and LIDAR readings on the bottom. There are several identical "compasses" instead of just one, since it was an easy way of emphasizing the "compass" so that the neural network could recognize it as an important feature. The LIDAR reading was also duplicated and stacked multiple times vertically on the bottom to balance its importance to the"compasses".

It is worth mentioning that when the truck is far away from next intersection, the compass is "switched off" pointing to the top.

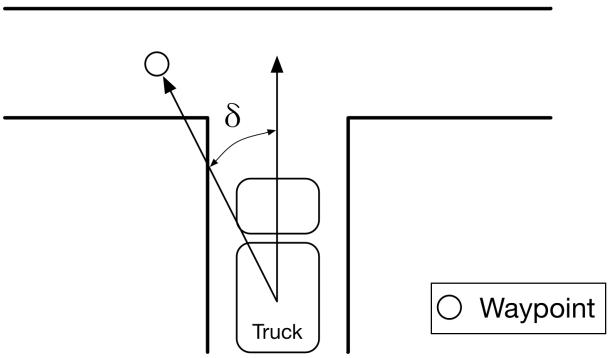


Figure 4.6: *How the "Compass" pointing direction is determined.*

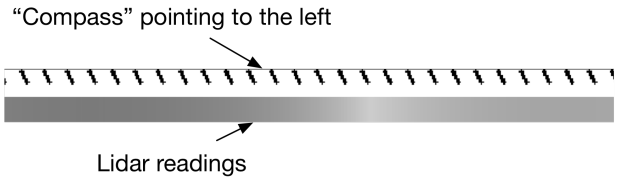


Figure 4.7: *Synthesized input to the neural network*

Since this approach considered the decision making problem, several hidden layers was added in order to handle the more complex task. The structure of the network could be seen in Figure 4.8.

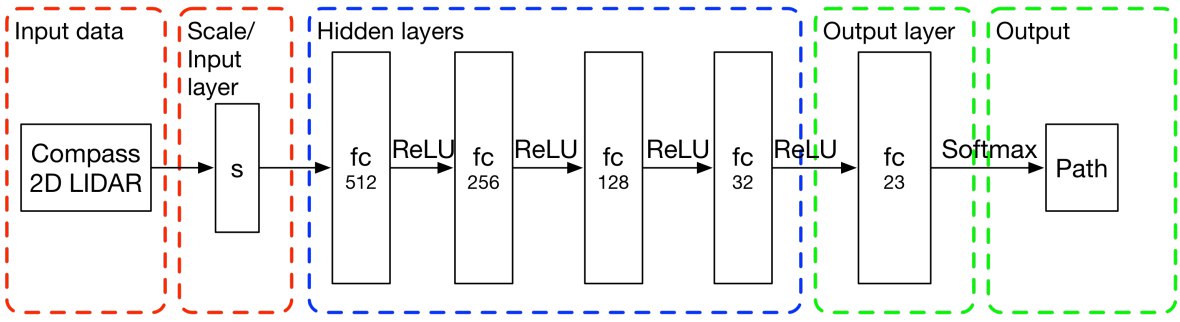


Figure 4.8: *The network structure in the E2E classification with "compass" approach using "compass" and 2D LIDAR sensor data as input*

4.6 E2E classification with "Compass" 3D

Instead of using 2D sensors as in the last approach, 3D LIDAR input was tested in this approach to examine the performance. The "compass" part was identical to the last approach. However, since now one 3D LIDAR reading takes up the same space as the upper part in the input image, it was therefore not needed to duplicate the LIDAR readings. The structure of the network, seen in Figure 4.9, in this approach is identical to the network structure in Section 4.5

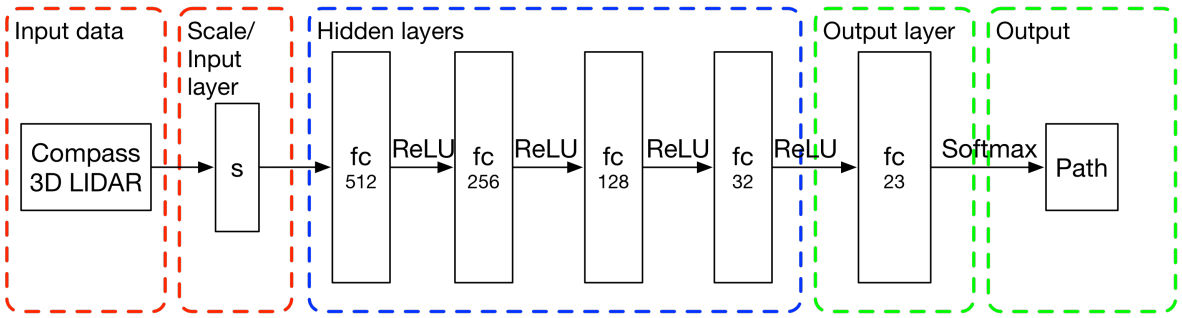


Figure 4.9: *The network structure in the E2E classification with "compass" approach using "compass" and 3D LIDAR sensor data as input*

For evaluation purposes, a convolutional network was created as well. The structure of this network, seen in Figure 4.10, uses three convolutional layers at the start of the network and ends with two fully connected layers, which is inspired by the one seen in [43]. One should bear in mind though that different sensors and different environments were used in this project.

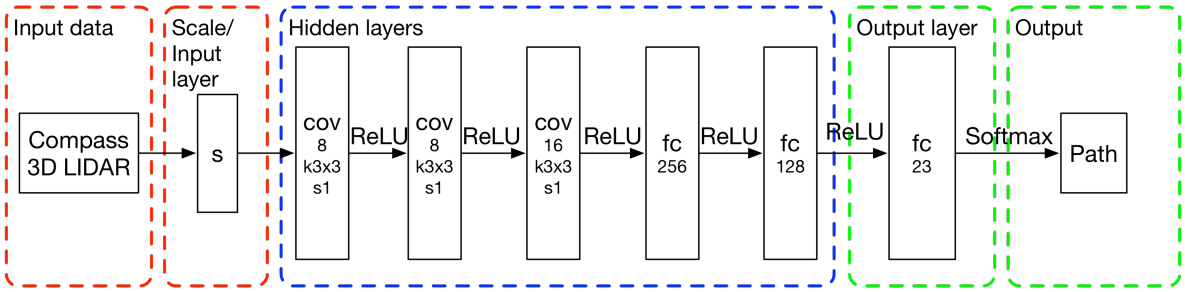


Figure 4.10: *The network structure in the convolutional E2E classification with "compass" approach using "compass" and 3D LIDAR sensor data as input*

5 Results and discussion

In this chapter, the results from the evaluation of the approaches will be presented alongside with general discussions on the project. During the project, a lot of different network structures have been investigated for each approach, but in order to give an easier comparison, the differences in the network structures has been kept at minimum in the evaluation. Instead, these matters will be discussed in a general manner at the end of this chapter. It is also worth mentioning that each deployed network has been tested multiple times at the evaluation track with a large degree repeatability. Only minor deviations can be seen if the starting position is changed.

5.1 E2E regression 2D

The first investigated approach is straightforward, simple and yet it shows promising results even though it fails to pass the last two features in the evaluation track. The green path in Figure 5.1 shows the trajectory of the rear axle center in the truck. From the figure it is evident that it has learned the correct behaviour, and it steer in the right direction. However, in proximity to feature 4 the generated steering angle is too small and the turn is also initiated too late. Further investigation of this net was done by putting the truck at the start of feature 4 pointing towards the end of the right turn. From this position, the truck is able to pass the last two features. Hence the result is considered to be easily improved by increasing the amount of training data. Another solution would be to extend the network with more layers, which would add complexity to the nonlinear function that the net approximates. However, the experience gained during the project tells us that the complexity of this net, using only two hidden layers, is sufficient for these types of tracks given that the volume of the training data is large enough and covers similar scenarios.

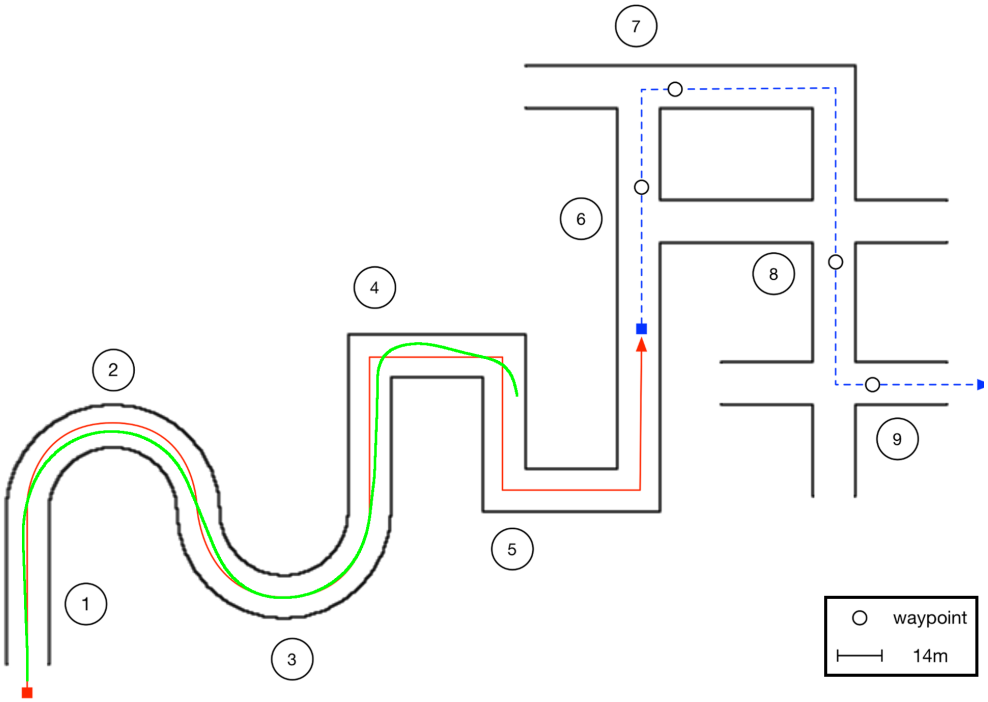


Figure 5.1: *The path created by the truck shown in green during one run at the evaluation track using the E2E regression network with 2D LIDAR sensor data as input.*

From Figure 5.2, two observations can be made. First, the steering commands from the network is noisy and unfiltered, it could wear out steering actuators if not handled properly. The origin of the noise is probably generated from the LIDAR sensors. It might be mitigated by using a deeper net, which could tell apart the noise from the readings. The second observation from Figure 5.2 is the extreme values of the output. It exceeds the limit of what the truck is capable of handling. The behaviour is expected, since the network output can

take any real number as explained in Section 4.1. In the simulation, this signal has been saturated at the steering controller of the truck. Even though none of the labels in the training data had larger values than the truck could cope with, it still generates these extreme values when the truck is getting closer to a wall. This could be due to the fact that since the network is that shallow, the nonlinearites cannot express the intended behaviour which should keep the output between reasonable values. Instead, when the truck gets close to the wall, the extrapolation has low fidelity and diverges from the expected value.

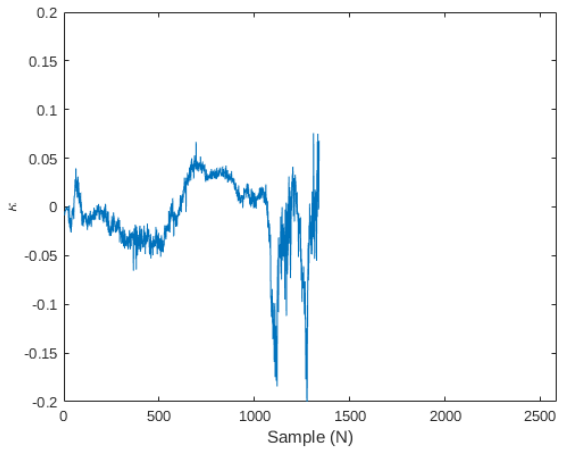


Figure 5.2: *Generated steering commands, κ, using E2E regression approach with 2D LIDAR*

5.2 E2E classification 2D

Moving from the simple regression approach to the classification approach, each steering angle value in the training data was classified to the closest match in the set of different curvatures. Thus, the output changes from one real number to the probability for each curvature in a categorical distribution. The certainty of generated steering command of the net is consequently exposed, which could be used to evaluate how well the network has learned to classify the input. For instance, the network should be confident in inferring a straight curvature in a straight corridor. The top predictions for the straight corridor should thereby be curvatures that are adjacent to the straight one, including the straight curvature itself. Investigations on this network showed that the certainty for the network was usually really high and that the top five curvatures were adjacent to the one selected, which is the intended behaviour. Figure 5.3 shows the generated network output using one example from the evaluation set as input.

Since the output could be seen as a categorical distribution, the certainty of the network for each steering command could be used by a global planner directly. However, since the global planner has been regarded to be outside of the scope in the project, the output with the highest probability was selected at each time step. This is done for all of the classification networks, and since the certainty and accuracy of the network is high it was considered to be sufficient to use the highest peak as output directly. To ensure a smoother steering, one could interpolate between the top predictions, which should reduce flickering between arcs that might occur, since the number of different arcs is limited. Increasing the number of arcs would also be a possible solution to mitigate this issue at the cost of increasing the complexity of the resulting network.

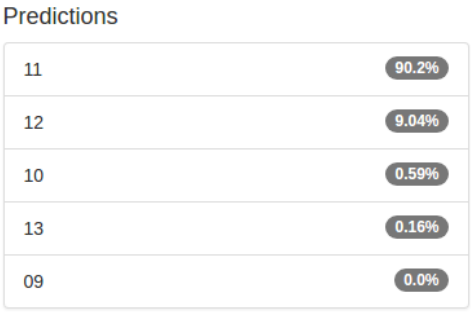


Figure 5.3: *Screenshot from DIGITS where the neural network has inferred the correct arc from sensor input labeled 11 from the evaluation set*

In contrast to the regression network, the logistic regression network is able to navigate through all the features on the evaluation track as seen in Figure 5.4. The same amount of training data is used, but since the output size is changed from just one real number to 23 values, the connections between the hidden layer and the output layer is increased, which makes the network capable of distinguish between different sensor inputs in a better way. Hence, it allowed us to remove one of the hidden layers while getting a better result as well. In the first three features, the truck is able to navigate close to the center of the track as intended. In feature 4 and 5, the truck overshoots the center line in the turns but is still capable of going through them.

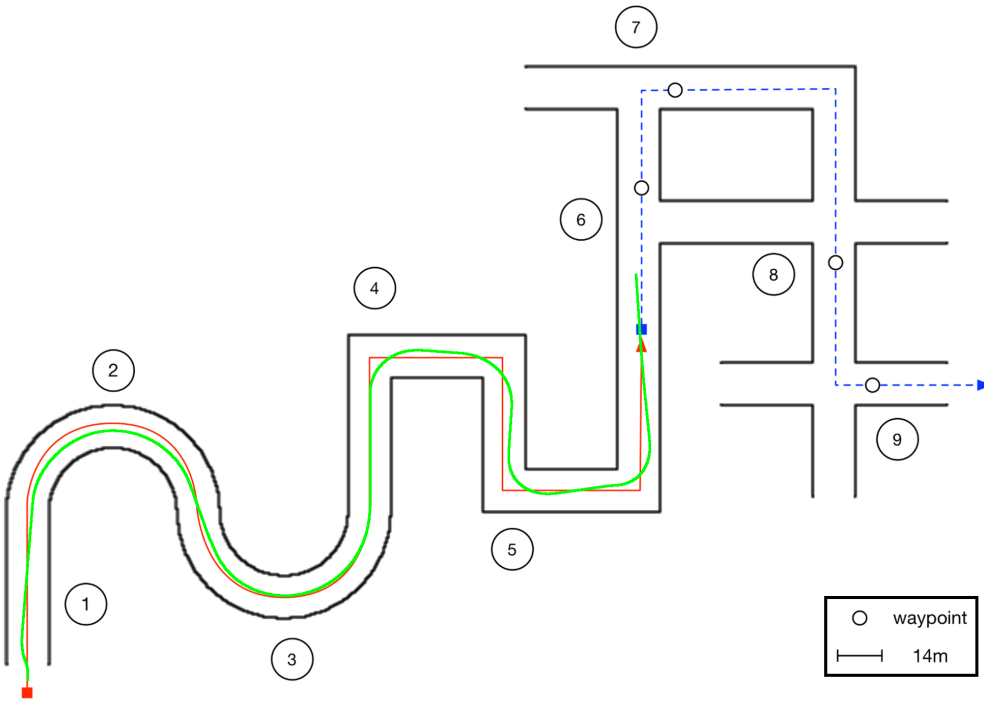


Figure 5.4: *The path created by the truck shown in green during one run at the evaluation track using the E2E classification network with 2D LIDAR sensor data as input*

The generated steering commands from one lap can be seen in Figure 5.5. By investigating the steering output from this network, one could observe that flickering among different curvatures occurs during the first features. However it does not affect the path noticeably, since the difference between two adjacent curvatures is small. The performance could be enhanced either by interpolation or extending the number of curvatures as mentioned previously. There is also a big step between the selected curvatures in the beginning of the lap. This is probably due to the fact the truck is starting outside the evaluation track, where most of the rays from sensor readings of the LIDAR are returning the maximum value, a case that is not present in the training data at all. This makes the behaviour in the beginning inpreditable, but it is still capable of completing feature 1

without any trouble. At the end of the lap (around sample 1500), there is a large peak before the last turn is initiated. This could suggest that the network tries to avoid cutting the corner. However, this large difference in steering output under a small amount of time is not generally preferred.

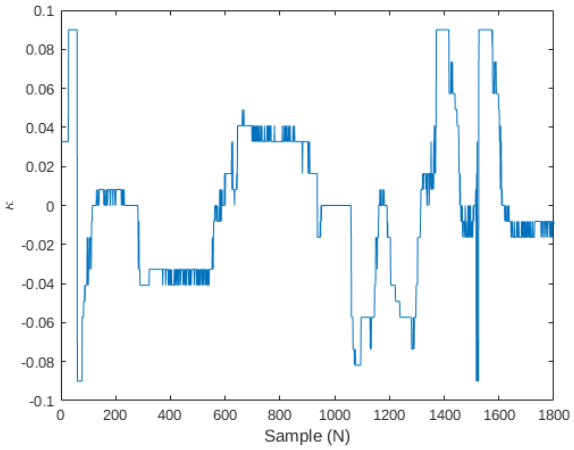


Figure 5.5: *Generated steering commands, κ, using E2E classification approach with 2D LIDAR*

5.3 E2E regression 3D

In this approach, two different sensor configurations are evaluated. The resulting path of front-placed 180-degree 3D sensor trial is shown in Figure 5.6, and steering commands in terms of *kappa* is shown in Figure 5.7. The truck succeeds to pass the full lap. However, unstable behaviour can be seen at the end of feature 2 and it goes too close to the inner wall at feature 3. Therefore, it is believed that using this sensor configuration, the ability of the network to position the truck correctly is limited.

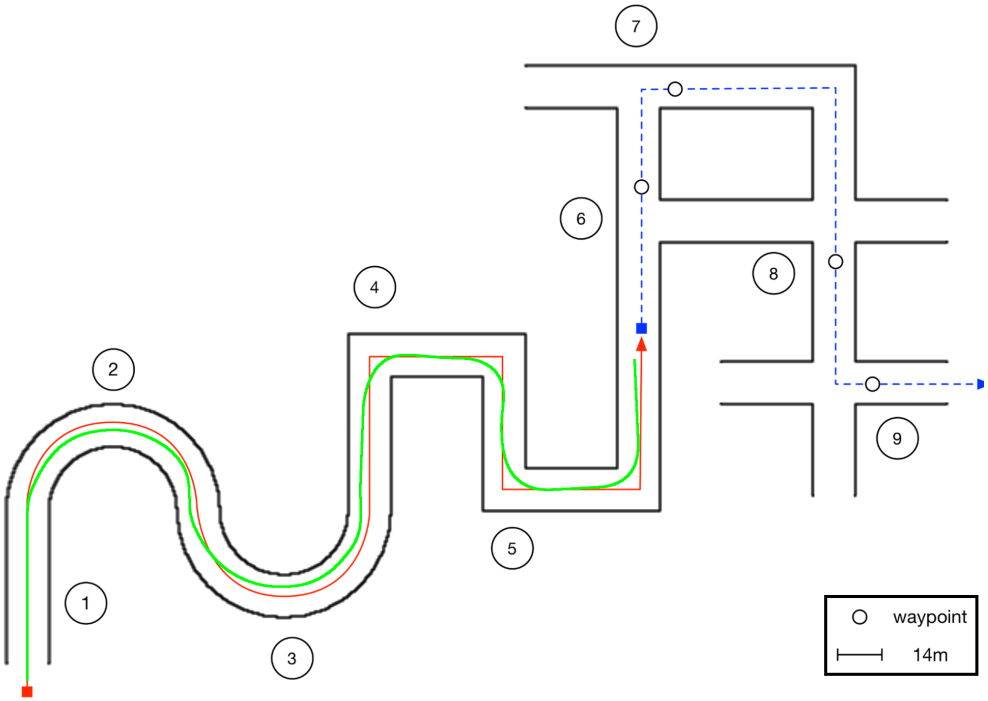


Figure 5.6: *The path created by the truck shown in green during one run at the evaluation track using the E2E regression network with 3D LIDAR sensor placed at the front of the truck*

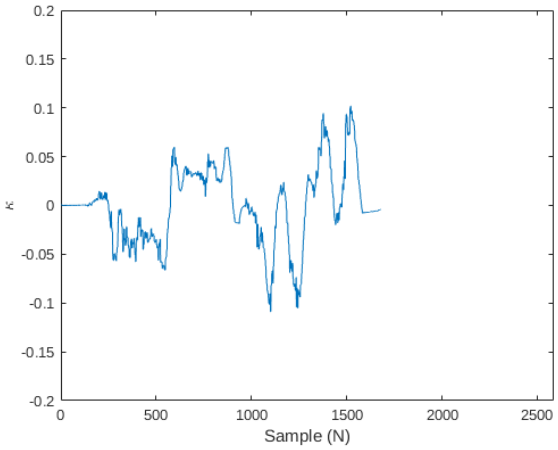


Figure 5.7: *Generated steering commands, κ, using E2E regression approach with 3D LIDAR placed at the front of the truck*

The other configuration with a 360-degree 3D LIDAR at the center passes the full track with a better positioning than front-placed LIDAR configuration at corners. It can be observed at feature 5 in Figure 5.8 and the corresponding area in Figure 5.9 that the truck reduces the turning angle in the middle of the turn, when it sees the potential of cutting corner. It suggests that the center arrangement shows a better performance in corners.

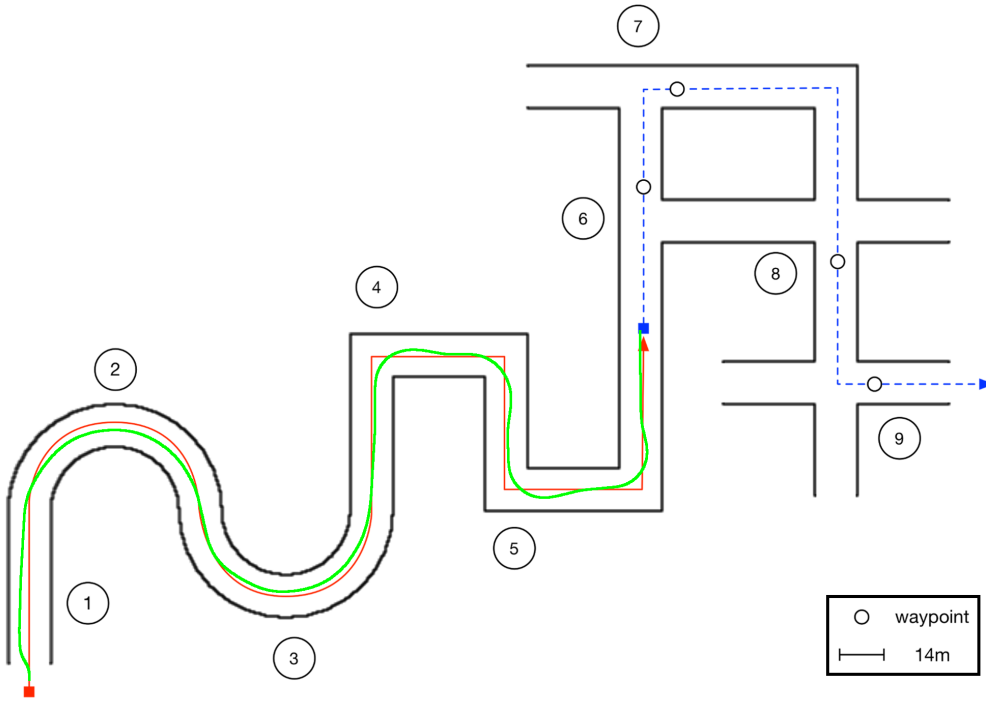


Figure 5.8: *The path created by the truck shown in green during one run at the evaluation track using the E2E regression network with 3D LIDAR sensor placed in the center of the truck*

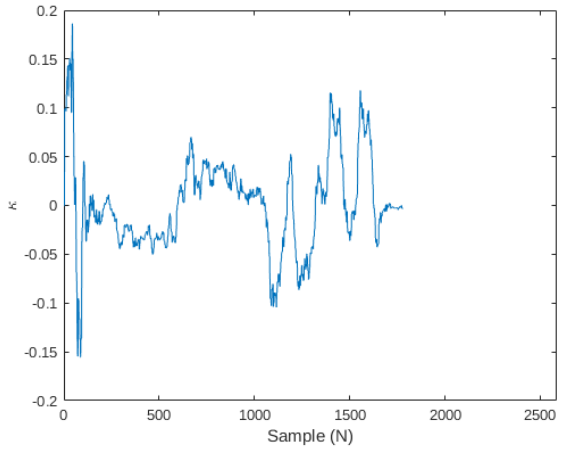


Figure 5.9: *Generated steering commands, κ, using E2E regression approach with 3D LIDAR placed in the center of the truck*

5.4 E2E classification 3D

As shown in Figure 5.10 and Figure 5.11, the configuration using LIDAR in the center of the truck passes the evaluation track successfully while the other, which has the LIDAR placed in the front failed at feature 3, a soft turn to the left.

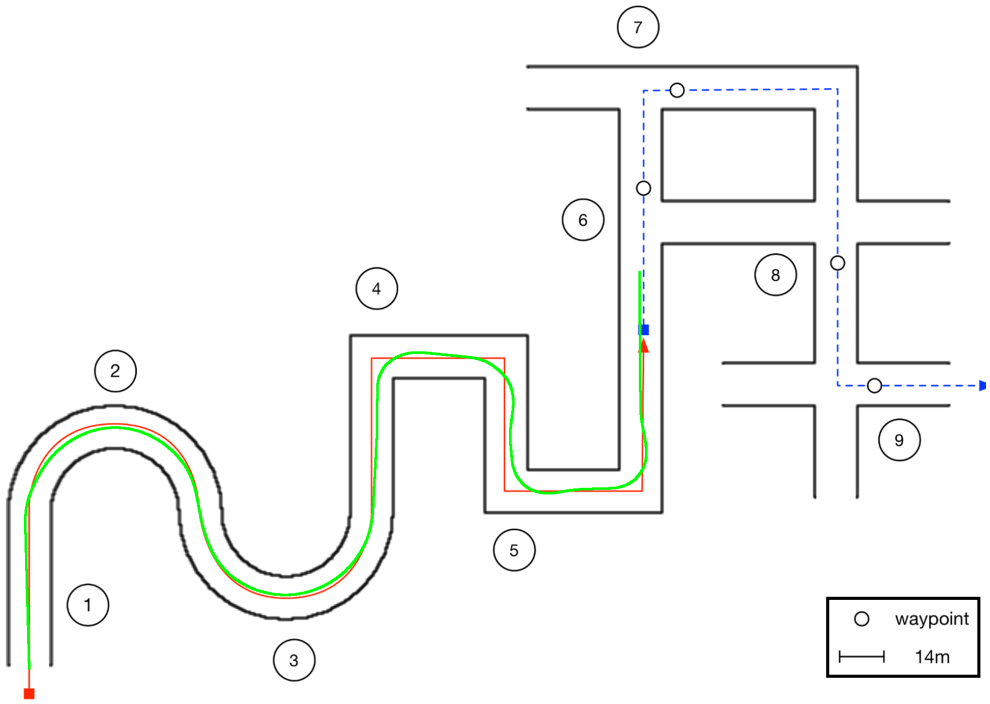


Figure 5.10: *The path created by the truck shown in green during one run at the evaluation track using the E2E classification network with 3D LIDAR sensor placed in the center of the truck*

With the LIDAR at the center, the result is quite similar to that of the E2E classification 2D approach. They share almost identical behaviors at all features in the map. However, it can be seen in Figure 5.12 that compared to the E2E classification 2D approach, the output is more stable which means the network is more confident in predictions utilizing the additional information given using 3D LIDAR.

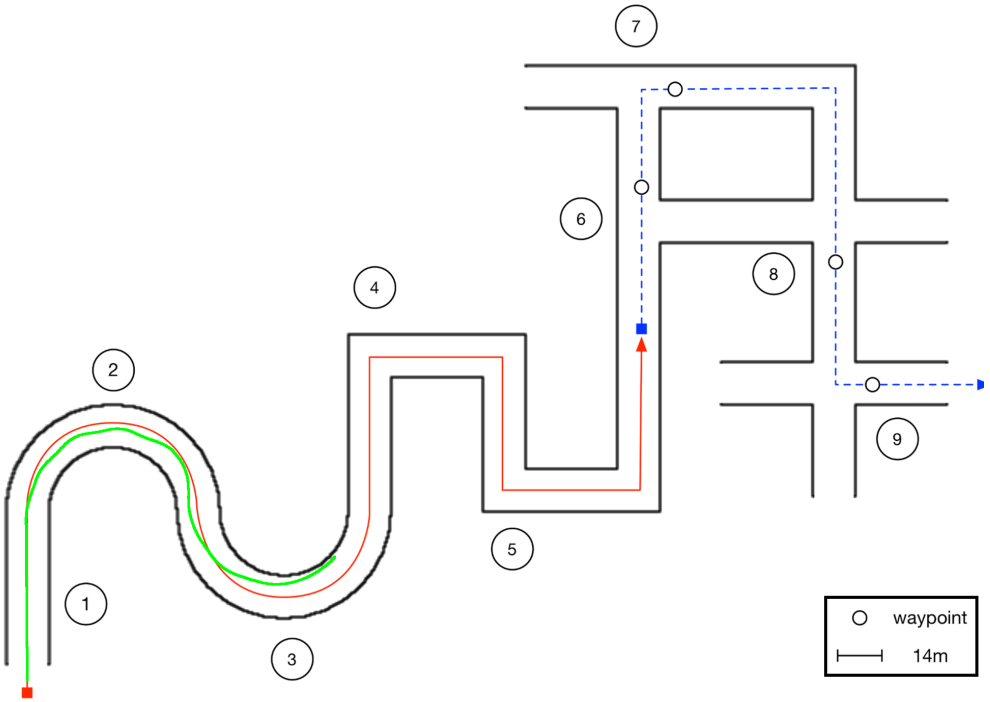


Figure 5.11: *The path created by the truck shown in green during one run at the evaluation track using the E2E classification network with 3D LIDAR sensor placed at the front of the truck*

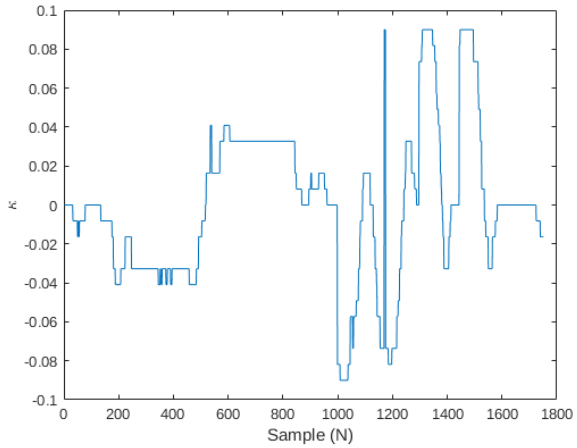


Figure 5.12: *Generated steering commands, κ, using E2E classification approach with 3D LIDAR placed in center of the truck*

Though the configuration with LIDAR at the front failed to complete a full lap, some interesting observations can be drawn. The truck shows oscillatory behaviour at feature 2, which can be seen from both the trajectory and the κ plot, but it still manages to finish this part of the track. At feature 3, the truck hits the wall which is identical to feature 2 apart from the direction. After several trials, it is observed that as long as the truck enters the corners at the center line with a heading pointing to the right angle (parallel to the tangent line of the track curvature), it can pass all the features on the map. This behaviour strengthen the belief that with only 180-degree view, it is difficult for the network to generate paths for the truck to position itself correctly.

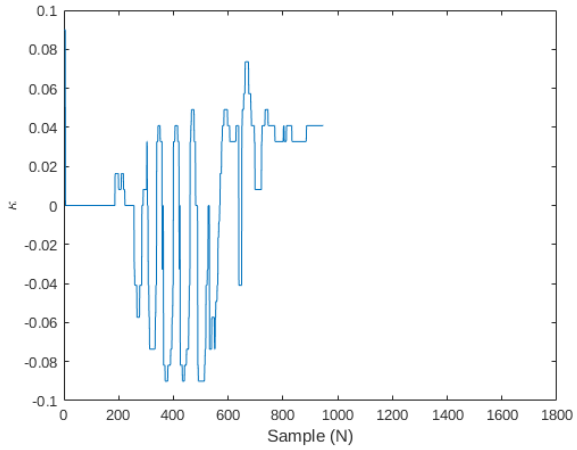


Figure 5.13: *Generated steering commands, κ, using E2E classification approach with 3D LIDAR placed at the front of the truck*

5.5 E2E classification with "compass" 2D

As shown in Figure 5.15, in this approach, the truck successfully navigates through intersections as well as corridors, and is able to finish a full lap without collisions. In the first part of the track, the result is not as good as that of E2E classification 2D in Section 5.2. The truck oscillates at feature 2 and gets too close to the wall at feature 4 as shown in Figure 5.15 and Figure 5.14. It suggests that the balance between the features in 2D LIDAR input and the compass is not handled well. For example, when the compass is "switched off" (pointing upwards), the network may rely on the "compass" part of the input data to much and is therefore reluctant to adjust the positioning of the truck or jumps between decisions, which lead to getting to close to the wall or oscillations respectively.

In the second half of the track, since the compass now points to the next waypoint, the performance of both stability and positioning is much better compared to the first half. The oscillation and corner-cutting behaviour is largely reduced and decisions are made as expected without any hesitation.

It is believed that with more training data, the problems at the first half can be solved, since a more precise balance between features from the LIDAR and the compass can be found.

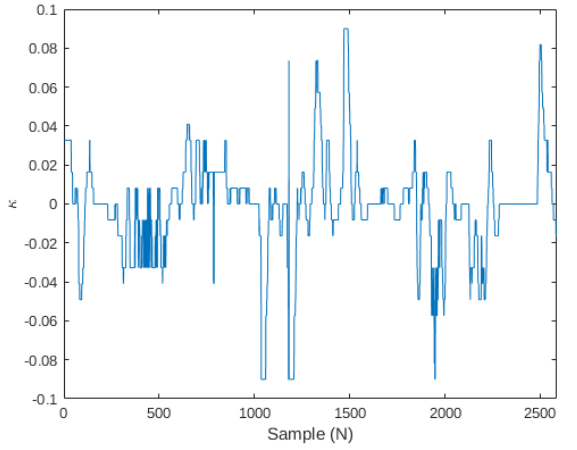


Figure 5.14: *Generated steering commands, κ, using E2E regression approach with "compass" and 2D LIDAR*

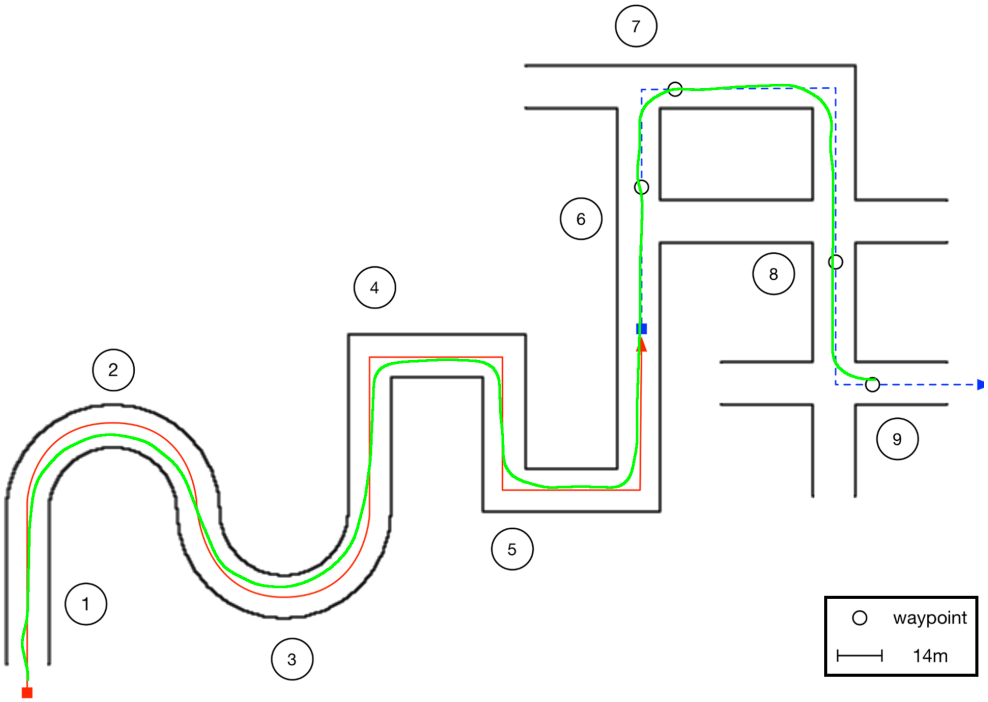


Figure 5.15: *The path created by the truck shown in green during one run at the evaluation track using the E2E classification network with "compass" and 2D LIDAR sensor data as input*

5.6 E2E classification with "compass" 3D

It can be seen in Figure 5.16 that the current approach finishes a full lap with similar behaviour to the 2D result stated above. Nonetheless, a more stable output is gained with 3D information as shown in Figure 5.17. Since there are more features in the 3D input that the network can rely on, the balance between features from the LIDAR sensor and the compass is easier to achieve, resulting in a better performance at the first and second half of the evaluation track than that in the 2D case. However, behaviours like oscillations and getting too close to the wall still exists, leading us to conclude that more training data is needed to gain a better result. With that being said, the result shows the compass approach of dealing with decision making works with 3D input as well.

As mentioned in Section 4.6, a convolutional network was trained for comparission. Figure 5.18 shows the path of the truck from one lap using this network. As seen in the figure, the performance is not as good as in the fully connected network. The truck drives close to the wall in feature 2 and cuts the corners feature 4 and 5. However, when the "compass" is turned on in the later parts of the evaluation track, it is able to make decisions and handles the turns in feature 7 and 9 as well as the fully connected network.

Investigation of the steering commands in Figure 5.19 reveals that the output from the network is not as stable when the "compass" is turned off compared to when it is active. However, compared to the fully connected network, it is evident that the convolutional network is not as stable in general. One reason for this could be that the convolutional neural network puts too much focus on the spatial relation between adjacent rays from the LIDAR sensor, instead of utilizing the range information present in each ray alone.

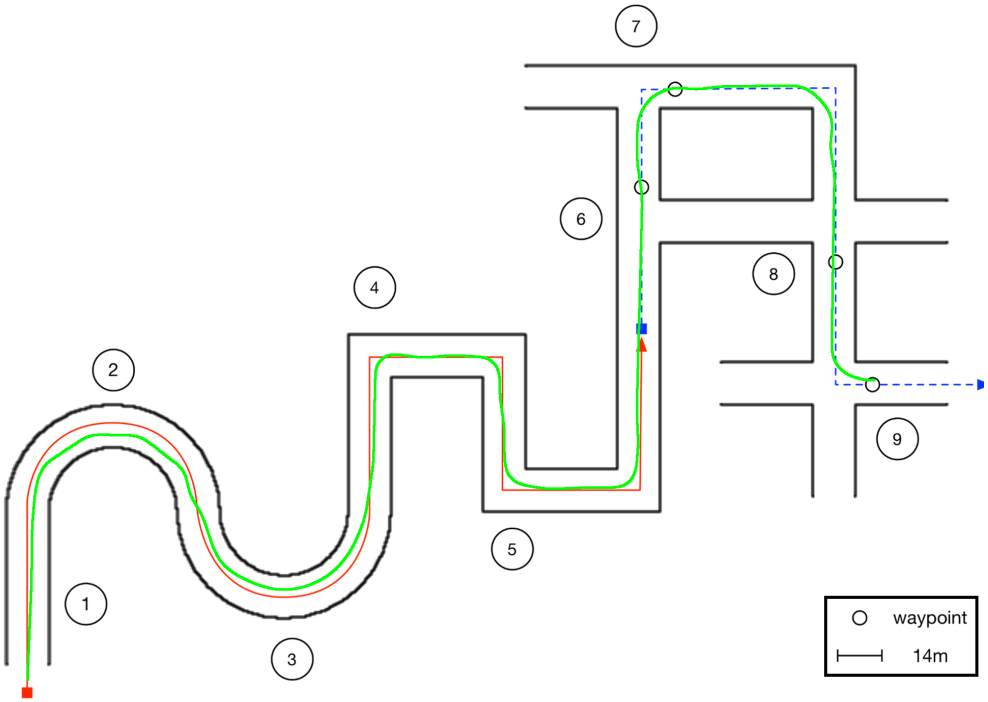


Figure 5.16: *The path created by the truck shown in green during one run at the evaluation track using the E2E classification network with "compass" and 3D LIDAR sensor data as input*

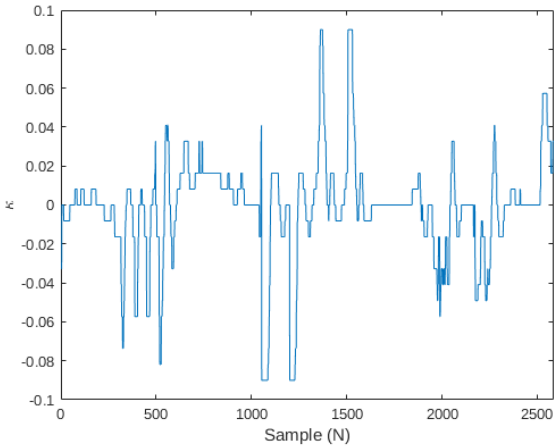


Figure 5.17: *Generated steering commands, κ, using E2E regression approach with "compass" and 3D LIDAR*

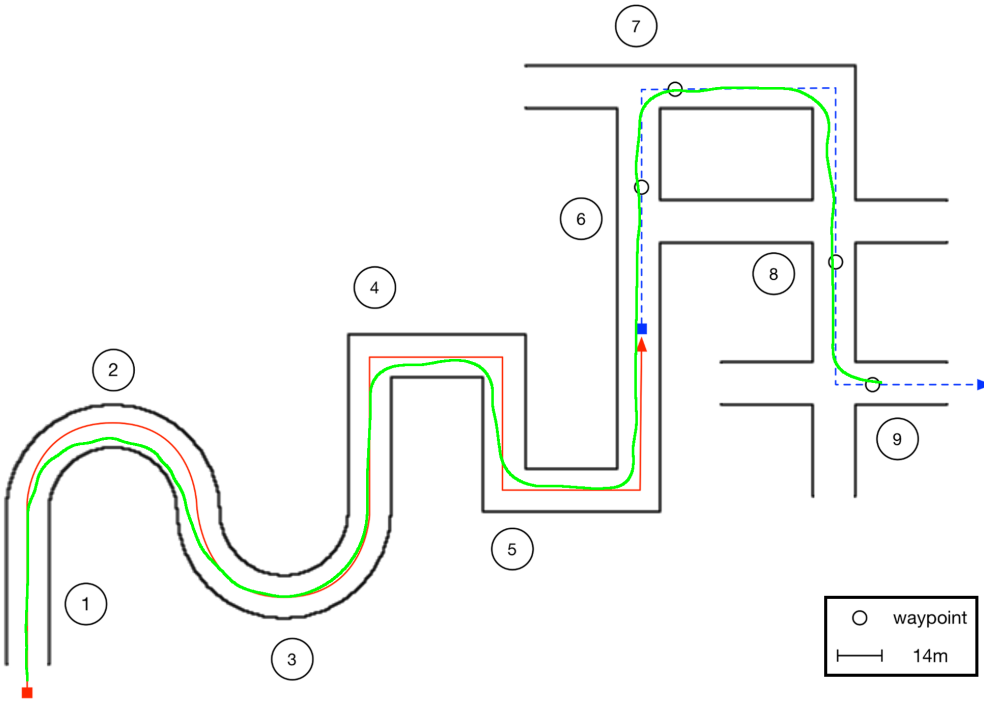


Figure 5.18: *The path created by the truck shown in green during one run at the evaluation track using the convolutional E2E classification network with "compass" and 3D LIDAR sensor data as input*

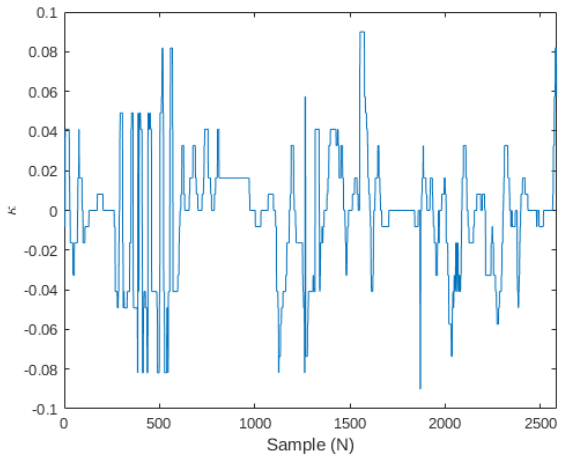


Figure 5.19: *Generated steering commands, κ, using convolutional E2E regression approach with "compass" and 2D LIDAR*

5.7 Discussion

As mentioned in Section 3.4, the network should be evaluated from the point of complexity and driving behaviour. A simplified subjective overview of the different approaches using these metrics has been summarized in Table 5.1. Each approach is listed as the section number in the report and is graded at each feature. It is worth noting that the results are repeatable and the grades are given based on multiple runs of every approach. The grades used for each feature are the following:

- **+** Given if the truck is able to pass the feature with a safe margin from the wall without any oscillations
- **0** Given if the truck is able to pass the feature but is close to the wall and/or show signs of oscillations
- **-** Given if the truck is not able to pass the feature
- **N/A** Given if the truck is not intended to pass that feature during the evaluation

To measure complexity, the execution time of one propagation from input to output in each net is measured. These measurements has been running on a Raspberry Pi 2 model B [44] running at 1Ghz.

Table 5.1: Summary of the approaches

Approach	Feature									Net. exec. time (s)
	1	2	3	4	5	6	7	8	9	
5.1	+	+	+	0	-	N/A	N/A	N/A	N/A	0.0072
5.2	0	+	+	0	0	N/A	N/A	N/A	N/A	0.0072
5.3 front	+	+	0	+	+	N/A	N/A	N/A	N/A	0.0766
5.3 center	0	+	0	0	0	N/A	N/A	N/A	N/A	0.0748
5.4 front	+	0	-	-	-	N/A	N/A	N/A	N/A	0.0740
5.4 center	+	+	+	0	0	N/A	N/A	N/A	N/A	0.0740
5.5	0	0	0	+	+	+	+	+	+	0.1543
5.6	+	0	0	+	+	+	+	+	+	0.1543
5.6 cov	+	0	+	0	0	0	+	+	+	0.9891

Limitations in the experiments

Due to limitations in DIGITS, the training data has to be formatted to 8-bit images. This means the resolution is limited to 0.156m according to Equation 5.1 if the LIDAR range is kept to 40m.

$$Resolution = \frac{LIDAR_{range}}{image_{depth}} \tag{5.1}$$

Since the simulation of 3D LIDAR can be very computational demanding, the update rate of the LIDAR output in the simulation is approximately 3Hz for 3D LIDAR and 10Hz for 2D LIDAR in all the experiments. However, all the networks evaluated can operate at above 10Hz using Nvidia GTX970 GPU , therefore the radar input is fed in repeatedly if it has not been updated before the next iteration of the network, so that the update rate of all the network output can be kept to 10Hz. This delay might introduce oscillations in the simulations that would not appear in a field experiment using real hardware.

Apart from the input and output, the comparison among the structure of neural networks has its limitations as well. As stated above, the structure of the networks are kept as similar as possible in the experiments. However, parameters related to input size and output type are inevitably different. For example, when the input size is 1×720, every node in the first layer of the network has 720 weights while the input size is 16, the number of weights is 11520. Besides, classification networks need to match the number of categories, resulting in 23 nodes, in the output layer, while regression ones need only 1 node.

Performance of the approaches

Bearing the evaluation limitations in mind, the summary of the performance of all approaches is listed below. In addition to discussion within the evaluation results, some observations and experience gained throughout the project are incorporated into the discussion to give a more complete review.

- 2D v.s. 3D
  As stated before, for practical applications, 3D input performs better since more obstacles can be detected and less affected by pitch and roll movement of the vehicle. However, apart from performance, when using 3D input, the number of features in the input grows as well as the complexity of the network. This means more training data is needed and the deployment is more computational demanding, while 2D input is able to provide a similar performance with simpler network structure and less computational demand.

- Regression v.s. Classification
  From the experiments, it is observed that regression networks are more sensitive to small deviations. It leads to a smoother and better positioned trajectory in corners with a small curvature. However, when the size of the training set is limited, the extrapolation of the regression network is not so accurate. Its performance usually drops at hard corners and when the truck is positioned with a large angle against the wall. The classification network, on the other hand, performs well in hard corners, but not as good as regression networks in soft corners given the same training data. It is believed that the classification network does not need to balance between straight corridors and hard corners, since the different labels from the dataset are not relevant to each other in the training and the network can therefore more easily be optimized for different input situations. In contrast, regression networks need high nonlinearities to cope with both situations which makes it harder to optimize, however the interpolation enables fine adjustments to minor changes of the input data.

- Execution time
  In the approaches without "compass" where 2D LIDAR are used, the execution time is significantly faster compared to the 3D LIDAR case. However, both of them can be executed above 10 Hz using the Raspberry Pi, suggesting that they are efficient enough for the application using limited hardware. The approaches with "compass" is able to be executed at a rate that is slightly slower than 10 Hz. We believe that the input size of the network can be reduced by using a more compact representation of the "compass" to get a shorter execution time. As expected, the convolutional network requires significantly longer execution time. It is also worth mentioning that the Raspberry Pi is executing the networks on the CPU. If GPU accelerated hardware is available the execution time could be reduced further.

- Center-configuration v.s. Front-configuration
  As explained in Section 4.1, it is assumed that center-configuration gives the network more information to determine the relative position of the truck in the track. This assumption conforms to the observation in the experiments. The center-configuration decreases corner-cutting behaviours and more importantly, provides more features for classification networks, therefore compensates the low sensitivity of classification networks to input deviations. However, it is worth mentioning that the packaging limitation in real applications makes it difficult to use a center-configuration-like input. It usually involves a merger which processes the data from multiple LIDARs around the vehicle to simulate the output from a center LIDAR. The merging itself takes a lot of computational effort, especially when high resolution 3D LIDARs are used. One way to eliminate merging is to feed all the raw sensor reading to the network and let the network itself to understand how to relate the data from different positions to the output. Actually, experiments to skip the merging were done with promising results, though they are not shown here due to the hardware limitation. Simulating multiple 3D LIDARs is too heavy for

- Decision making
  When only considering corridors, the networks are capable of positioning the truck in the right place and driving along the track. However, when the truck is at intersections, the regression networks drives to a random direction. This behaviour is expected, since the only output from the network is steering angle and it simply lacks the degree-of-freedom to tackle with intersections. For classification networks, it was expected to output several high probability paths at the same time when it is in intersections. However, it becomes, in fact, uncertain for all paths rather than certain for several paths. Hence the "compass" is introduced to explicitly give the instruction of turning direction to the network, and the results show that it works. With that being said, this approach needs the information of truck position and waypoints coordinates, which is not as good as pure relative localization in terms of application, since global localization system is therefore needed. There are several ways to solve this problem, for example, using multi-label-classification network. However, the ways of creating valid data has to be developed. During the project, a lot of effort has been invested to get quality datasets, but the results

were not stable enough. From our point of view, these datasets must be easy to generate in order to be feasible in a real product hence the approaches were limited. Except just using the "compass", other types of information could be also utilized. In a recent report, [45], Google maps, GPS and IMU are used with good results. In the report, a fully convolutional network are developed, which can predict long paths at a high rate on straight roads as well as in intersections. However, it uses a much deeper and complex net that put much higher requirements on the hardware.

Neural networks and safety

The result of the experiments shows that shallow fully-connected feedforward networks are sufficient for end-to-end navigation by generating short local paths in the simulated mining environment. The performance improvement is negligible beyond 5 layers in our studies. Nonetheless, the input variety is much less than that in reality, therefore much more complex networks might be needed for real world implementation.

One observation worth mentioning is that convolutional networks might not be suitable for the application type of this project. We believe that convolutional layers are good at feature identification but lose position resolution of those features. This leads to low performance in relative localization and therefore poor positioning of the truck. During the project, this issue has been observed when convolutional networks had been tested. The fully connected layer always tries to recenter the truck when it is perturbed in corridors, while the convolutional networks has just recovered the truck to be parallel against the walls, indifferent to whether one of the walls is really close to the truck itself.

Since the detailed mechanism behind neural networks is still unknown, safety evaluation by analysis for the network itself is not applicable. Instead, safety checks and restrictions could be applied to the output. Therefore, classification networks might be better, since the safety evaluation can be applied for several high probability paths instead of just one binary decision on the regression output.

6 Conclusion and future work

The development of autonomous vehicles has drawn a lot of attention from both academical and industrial sectors in recent years. Neural networks, another subject undergoing intense study recently, are utilized as a powerful tool to handle problems within autonomous driving. Especially, the ability of neural networks to understand camera image fast and accurately has the potential to make autonomous vehicles more intelligent. However, in terms of end-to-end learning using LIDAR data within automotive applications, it can still be considered as a less explored area. In this project, attempts have been made to find a way to organize data so that even shallow neural networks can interpret the data in a useful way efficiently.

First of all, from the experiments, it is clear that the neural networks are capable of relaying LIDAR range data to generated paths, and more importantly, it is not necessary to implement computational demanding deep networks for end-to-end learning in simple environment like mining sites. It is also found that when dealing with LIDAR data which contains rich depth information, fully-connected neural network shows higher performance than convolutional networks, though convolutional neural network shows a great potential on interpreting camera data in various studies.

Secondly, 360-degree view is necessary for vehicles with long wheelbase. It is crucial to avoid corner-cutting behaviour for large vehicles such as trucks. Without information on both sides of the vehicle, the network is not able to perform high quality relative localization, therefore performs poorly in sharp corners.

At last, with a proper way of giving instructions, the neural network is able to make decisions at intersections according to the given instructions. In this project, a "compass" is formed as visual signal to the networks, so that the network knows which direction it should follow at the intersection. The evaluation shows that this approach is able to inform the network correctly and robustly.

The resulting neural networks of this project are able to drive a truck in simulated tracks with intersections, and a tool-chain of data collecting, neural network training, deploying and testing in simulated environment is developed as well. However, there is still plenty of work left to do.

In terms of neural networks, there are many structures and techniques such as recurrent networks and phase functions that can be implemented to gain a better performance. In addition, the network deploying program can be multithreaded to reduce the execution time in less powerful hardware such as Raspberry Pi. Moreover, the training methods for other driving tasks such as speed regulation, reversing and parking need to be explored to develop a more complete autonomous driving solution. On the other hand, the mathematical explanation of observed behaviours of different networks in the experiments is an interesting and important, though challenging topic to pursuit in the future.

References

[1] P. Stone et al. *Drive Me*. 2017. URL: http://www.volvocars.com/intl/about/our-innovation-brands/intellisafe/autonomous-driving/drive-me (visited on 02/13/2017).

[2] J. D. Crisman and C. E. Thorpe. "SCARF: a color vision system that tracks roads and intersections". In: *IEEE Transactions on Robotics and Automation* 9.1 (Feb. 1993), pp. 49–58. ISSN: 1042-296X. DOI: 10.1109/70.210794.

[3] J. Michels, A. Saxena, and A. Y. Ng. "High Speed Obstacle Avoidance Using Monocular Vision and Reinforcement Learning". In: *Proceedings of the 22Nd International Conference on Machine Learning*. ICML '05. Bonn, Germany: ACM, 2005, pp. 593–600. ISBN: 1-59593-180-5. DOI: 10.1145/1102351.1102426. URL: http://doi.acm.org.proxy.lib.chalmers.se/10.1145/1102351.1102426.

[4] T. Pilutti and A. G. Ulsoy. "Identification of driver state for lane-keeping tasks". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 29.5 (Sept. 1999), pp. 486–502. ISSN: 1083-4427. DOI: 10.1109/3468.784175.

[5] B. Jia, W. Feng, and M. Zhu. "Obstacle detection in single images with deep neural networks". In: *Signal, Image and Video Processing* 10.6 (2016), pp. 1033–1040. ISSN: 1863-1711. DOI: 10.1007/s11760-015-0855-4. URL: http://dx.doi.org/10.1007/s11760-015-0855-4.

[6] D. Gavrila. "The Visual Analysis of Human Movement: A Survey". In: *Computer Vision and Image Understanding* 73.1 (1999), pp. 82–98. ISSN: 1077-3142. DOI: http://dx.doi.org/10.1006/cviu.1998.0716. URL: http://www.sciencedirect.com/science/article/pii/S1077314298907160.

[7] M. Buehler et al. *The DARPA Urban Challenge: autonomous vehicles in city traffic*. English. Vol. 56.;56; Berlin: Springer, 2009.

[8] Tesla Inc. *Autopilot*. 2017. URL: https://www.tesla.com/autopilot (visited on 02/13/2017).

[9] Alphabet Inc. *Waymo*. 2017. URL: https://waymo.com/ (visited on 02/13/2017).

[10] Volvo Cars. *Drive Me*. 2017. URL: http://www.volvocars.com/intl/about/our-innovation-brands/intellisafe/autonomous-driving/drive-me (visited on 02/13/2017).

[11] C. Stergiou and D. Siganos. *Neural Networks*. 2017. URL: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html (visited on 02/14/2017).

[12] Y. Bengio, Y. LeCun, et al. "Scaling learning algorithms towards AI". In: *Large-scale kernel machines* 34.5 (2007), pp. 1–41.

[13] M. Bojarski et al. "End to End Learning for Self-Driving Cars". In: *CoRR* abs/1604.07316 (2016). URL: http://arxiv.org/abs/1604.07316.

[14] H. Xu et al. "End-to-end Learning of Driving Models from Large-scale Video Datasets". In: *arXiv preprint arXiv:1612.01079* (2016).

[15] B. Paden et al. "A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles". In: *IEEE Transactions on Intelligent Vehicles* 1.1 (2016), pp. 33–55.

[16] P. Varaiya. "Smart cars on smart roads: problems of control". In: *IEEE Transactions on automatic control* 38.2 (1993), pp. 195–207.

[17] C. Katrakazas et al. "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions". In: *Transportation Research Part C: Emerging Technologies* 60 (2015), pp. 416–442.

[18] J. Wit, C. D. Crane, and D. Armstrong. "Autonomous ground vehicle path tracking". In: *Journal of Field Robotics* 21.8 (2004), pp. 439–449.

[19] F. Fahimi. *Autonomous robots: modeling, path planning, and control*. Vol. 107. Springer Science & Business Media, 2008.

[20] Y. K. Hwang and N. Ahuja. "Gross motion planning—a survey". In: *ACM Computing Surveys (CSUR)* 24.3 (1992), pp. 219–291.

[21] U. Lee et al. "Local path planning in a complex environment for self-driving car". In: *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014 IEEE 4th Annual International Conference on*. IEEE. 2014, pp. 445–450.

[22] K. H. Sedighi et al. "Autonomous local path planning for a mobile robot using a genetic algorithm". In: *Evolutionary Computation, 2004. CEC2004. Congress on*. Vol. 2. IEEE. 2004, pp. 1338–1345.

[23] H. H. Poole. *Fundamentals of robotics engineering*. Springer Science & Business Media, 2012.

[24] J. J. Leonard and H. F. Durrant-Whyte. "Mobile robot localization by tracking geometric beacons". In: *IEEE transactions on robotics and automation* 7.3 (1991), pp. 376–382.

[25] P. Goel, S. I. Roumeliotis, and G. S. Sukhatme. "Robust localization using relative and absolute position estimates". In: *Intelligent Robots and Systems, 1999. IROS'99. Proceedings. 1999 IEEE/RSJ International Conference on*. Vol. 2. IEEE. 1999, pp. 1134–1140.
[26] N. L. Doh, H. Choset, and W. K. Chung. "Relative localization using path odometry information". In: *Autonomous Robots* 21.2 (2006), pp. 143–154.
[27] J. Borenstein, L. Feng, and H. Everett. *Navigating mobile robots: Systems and techniques*. AK Peters, Ltd., 1996.
[28] Y. LeCun et al. "Off-road obstacle avoidance through end-to-end learning". In: *NIPS*. 2005, pp. 739–746.
[29] W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.
[30] Yaldex. *A basic neuron*. [Online; accessed April 27, 2017]. URL: `http://www.yaldex.com/games-programming/FILES/12fig30.gif`.
[31] M. Wahde. *Biologically inspired optimization methods: an introduction*. WIT press, 2008.
[32] R. Quiza and J. P. Davim. "Computational methods and optimization". In: *Machining of hard materials*. Springer, 2011.
[33] deeplearning.net. *LeNet model*. [Online; accessed April 27, 2017]. URL: `http://deeplearning.net/tutorial/_images/mylenet.png`.
[34] K. O. Stanley and R. Miikkulainen. "Evolving neural networks through augmenting topologies". In: *Evolutionary computation* 10.2 (2002), pp. 99–127.
[35] M. Quigley et al. "ROS: an open-source Robot Operating System". In: *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*. Kobe, Japan, May 2009.
[36] Gazebo. *Gazebo - Robot simulation made easy*. 2017. URL: `http://gazebosim.org/` (visited on 02/27/2017).
[37] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Institute, Amsterdam: Blender Foundation, 2017. URL: `http://www.blender.org` (visited on 02/28/2017).
[38] S. Bagchi and E. Soultani. *Rapid close surrounding evaluation for autonomous commercial vehicles*. 2016.
[39] A. L. Ballardini et al. "ira_laser_tools: a ROS LaserScan manipulation toolbox". In: *arXiv preprint arXiv:1411.1086* (2014).
[40] Y. Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *arXiv preprint arXiv:1408.5093* (2014).
[41] *The 2-Clause BSD License*. Open Source Initiative, 2017. URL: `https://opensource.org/licenses/BSD-2-Clause` (visited on 01/25/2017).
[42] Volvo Cars. *Drive Me*. 2017. URL: `https://developer.nvidia.com/digits` (visited on 04/11/2017).
[43] M. Bojarski et al. *End-to-End Deep Learning for Self-Driving Cars*. 2016. URL: `https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/` (visited on 12/13/2016).
[44] Raspberry Pi Foundation. *RASPBERRY PI DOCUMENTATION*. 2017. URL: `https://www.raspberrypi.org/documentation/` (visited on 06/04/2017).
[45] L. Caltagirone et al. "Simultaneous Perception and Path Generation Using Fully Convolutional Neural Networks". In: *arXiv preprint arXiv:1703.08987* (2017).