



Norwegian University of
Science and Technology

Deep Convolutional Networks for Steering an Off-Road Unmanned Ground Vehicle

End-To-End Learning and Sensor Fusion

Erlend Faxvaag

Master of Science in Cybernetics and Robotics

Submission date: June 2018

Supervisor: Kristin Ytterstad Pettersen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Abstract


Autonomous vehicles have numerous advantages compared to standard vehicles. They can reduce fuel consumption, reduce injuries and death, optimize mobility, and reduce traffic congestion. Most lane assists used in consumer cars today are built up by several modules and can be complex and non-general, and few of these can handle dirt roads. Change in weather and road types can cause drastic and unwanted effects on the performance and safety of the lane assist. The end-to-end approach for autonomous vehicles has shown promising results in the later years. It uses a front-facing camera on a vehicle that feeds images of the road through a Convolutional Neural Network (CNN) which then maps each image directly to a steering angle. End-to-end networks have the advantage that there is no need for manually designing rules. The network is trained using supervised learning by cloning the behavior of human maneuvers. It is trained to generalize its perception of the road so it can predict accurately regardless of weather and road conditions. This thesis proposes a multi-input end-to-end network for dirt roads that combines camera images and Light Detection And Ranging (LiDAR) data in an attempt to outperform single-input end-to-end networks. Using a test set, the experiment proved that combining camera images and LiDAR outperforms camera or LiDAR alone. Multi-input networks can, therefore, improve the local navigation of an off-road autonomous UGV. Additionally, a path verification technique is presented. It uses a segmentation network to segment out the dirt road, and together with the predicted steering angle, it evaluates the local path of the vehicle.

Sammendrag

Selvkjørende biler gir flere fordeler sammenlignet med vanlige kjøretøy. De kan redusere drivstofforbruk, redusere skader og ulykker, optimalisere framkommelighet, og redusere kø på veiene. De fleste kjørebane-assistenter i biler i dag er bygget opp av flere moduler og kan være komplekse og er ikke-generelle, og få er terrenggående. Skiftende vær og underlag kan ha uønskede effekter på ytelsen og sikkerheten til kjørebane-assistenten. Ende-til-ende tilnærmingen for autonome kjøretøy har vist lovende resultater de siste årene. Den bruker et forovervendt kamera på et kjøretøy som mater bilder av veien inn til et konvolusjonsnettverk som direkte tilegner hvert bilde en anbefalt styrevinkel. Ende-til-ende nettverk har fordelen med at man ikke trenger å manuelt designe regler. Nettverket er trent ved hjelp av ledet læring ved å etterligne menneskelige manøvre. Den er trent til å generalisere sin oppfatning av veien uavhengig av kjøreforhold som nedbør og underlag. Denne avhandlingen foreslår et ende-til-ende nettverk med inngang for både kamera bilder og LiDAR, i et forsøk på å forbedre resultatet i forhold til ende-til-ende nettverk med kun en inngang. Ved å bruke et test dataset ble det demonstrert at nettverket med flere innganger oppnår et bedre resultat enn nettverk som kun benytter kamera eller LiDAR alene. Flere innganger kan dermed forbedre den lokale navigeringen til en autonom UGV. I tillegg blir en teknikk for bane-verifisering foreslått. En slik teknikk benytter et segmenteringsnettverk for å segmentere ut veien, og sammen med den predikerte styrevinkelen gjøres det forsøk på å evaluere bilens bane i forhold til veien.

Preface

This master thesis is the concluding work of my Master of Science degree at Department of Engineering Cybernetics (ITK), Norges teknisk-naturvitenskapelige universitet (NTNU). Problem description is to be found in Section 1.2. The background and contributions of the project are described in more detail in Section 1.3 and 1.4. I would like to thank Kristin Y. Pettersen for the opportunity, the provided hardware, and guidance through this project. I would also like to express my gratitude towards Narada Warakagoda and Forsvarets forskningsinstitutt (FFI) for introducing the problem undertaken in this thesis. Your support and software have been vital for the progress of this report. Finally, I would like to thank Johann Diral for the collaboration.



Erlend Faxvaag Johnsen

Trondheim, June 2018

List of Acronyms

API Application Programming Interface

ANN Artificial Neural Network

AP Average Precision

CUDA Compute Unified Device Architecture

CNN Convolutional Neural Network

DL Deep Learning

ELU Exponential Linear Units

FFI Forsvarets Forskningsinstitut

FCN Fully Convolutional Network

GPU Graphics Processing Unit

ILSVRC ImageNet Large Scale Visual Recognition Competition

ICR Instant Centre of Rotation

IU Intersection over Union

LiDAR Light Detection And Ranging

LSTM Long Short Term Memory

MSE Mean Square Error

ReLU Rectified Linear Units

RNN Recurrent Neural Network

ROS Robot Operating System

RMSE Root Mean Square Error

UGV Unmanned Ground Vehicle

Contents

Abstract	i
Sammendrag	ii
Preface	iii
1 Introduction	1
1.1 Motivation	1
1.2 Problem description	2
1.3 Background	4
1.4 Contributions	4
1.5 Related work	5
1.5.1 Single-input end-to-end networks	5
1.5.2 Multi-input end-to-end networks	5
1.5.3 FCNs for semantic segmentation	6
1.6 Outline	6
2 Theory	7
2.1 Artificial Neural Networks (ANNs)	7
2.1.1 Loss functions	9
2.1.2 Optimizers	10
2.1.3 Activation functions	11

2.1.4	Regularization	12
2.2	The end-to-end approach	13
2.3	FCNs for semantic segmentation	13
2.3.1	Evaluation metrics	14
3	Methodology	17
3.1	Tools	17
3.1.1	Workstation	17
3.1.2	TensorFlow	18
3.2	Datasets	18
3.2.1	Dataset for steering angle network	20
3.2.2	Labeling data for segmentation network	22
3.2.3	Existing datasets	23
3.3	Training steering angle network	24
3.3.1	Training camera subnetwork	25
3.3.2	Training LiDAR subnetwork	28
3.3.3	Training fusion network	30
3.4	Training segmentation network	31
3.5	Path verification	34
3.5.1	Pipeline	35
3.5.2	Verification test	37
4	Results and conclusion	39
4.1	Results	39
4.1.1	Steering angle network	39
4.1.2	Segmentation network	42
4.1.3	Path verification	42
4.2	Discussion	42
4.2.1	Steering angle model	42
4.2.2	Segmentation model	45
4.2.3	Path verification	46
4.3	Future work	48

4.4	Conclusion	49
A	Appendix: Media attachment	51
A.1	Script setup	51
A.2	Organization	52
B	Appendix: Steering angle prediction from segmented images	53
	References	57

List of Figures

1.1	System overview	3
2.1	CNN filter example	8
2.2	Simple Artificial Neural Network (ANN)	9
2.3	FCN architecture	14
3.1	Sample from camera dataset	20
3.2	Sample from LiDAR dataset	21
3.3	Sample from segmentation dataset	23
3.4	Steering angle model	25
3.5	Dirt road samples with steering angle	26
3.6	Comparison between activation functions	27
3.7	Images to be segmented	32
3.8	Path verification example	36
4.1	Steering angle predictions	40
4.2	Camera subnetwork loss plot	40
4.3	LiDAR subnetwork loss plot	41
4.4	Steering angle network loss plot	41
4.5	Samples showing less accurate segmentation	43
4.6	Training set steering angle distribution	45
4.7	Path verification samples	47

B.1	Radius of curvature	54
B.2	Steering angle prediction using segmented images	55

List of Tables

- 3.1 Dataset for steering angle network 22
- 3.2 Dataset for segmentation network 22
- 3.3 Hyper parameters steering angle network 28
- 3.4 Steering angle subnetwork 29
- 3.5 Hyper parameters for segmentation network 34

Chapter 1

Introduction

1.1 Motivation

Autonomous vehicles have numerous of advantages compared to standard vehicles. They can reduce fuel consumption, reduce injuries and death, optimize mobility, and reduce traffic congestion. Most lane assists used in consumer cars today are built up by several modules and can be complex and non-general, and few of these can handle dirt roads. Change in weather and road types can cause drastic and unwanted effects on the performance and safety of the lane assist. However, several car manufacturers, startups, and other companies have in recent years started to research and development of self-driving cars using Deep Learning (DL). A common technique used for this is called the end-to-end approach [1, 2, 3]. Instead of making rules on how a car should behave, they use supervised learning with an Artificial Neural Network (ANN) to clone the behavior of human maneuvers accurately. The end-to-end approach for autonomous vehicles has shown promising results [1, 2, 3]. Yet, most papers about self-driving cars present a system meant for public roads and utilizes only camera images in their end-to-end networks.

1.2 Problem description

This thesis will investigate the use of the end-to-end approach with both camera images and LiDAR data as input for steering angle prediction, everything in an off-road environment like dirt roads. The multi-input end-to-end network will be compared to single-input end-to-end networks only using camera images or LiDAR data as input. The end-to-end network will in this thesis be named steering angle network. Additionally, this thesis presents a possible and experimental technique for local path verification using the output of a segmentation network together with the predicted steering angle from the steering angle network. By combining the results from the two models, steering angle, and segmentation, one can verify the vehicles local path. The end-to-end network uses supervised learning to clone and learn from human behavior, while the segmentation network uses supervised learning for road segmentation. Figure 1.1 illustrates the full system. Unlike asphalt roads, colors on the ground on dirt roads and trails do not have clearly defined meanings. Trees, thicket, and poorly defined roads is a messy environment for a computer which makes this task challenging. The task can be divided up into the following objectives:

1. Develop two single-input end-to-end networks for steering angle prediction on dirt roads, one for camera images and one for LiDAR data.
2. Develop a multi-input end-to-end network for steering angle prediction on dirt roads by fusing the two networks mentioned above.
3. Compare the performance of the networks.
4. Develop a segmentation network for segmenting road from non-road.
5. Use the segmented road together with the predicted steering angle from the previous network to present an experimental technique for path verification.

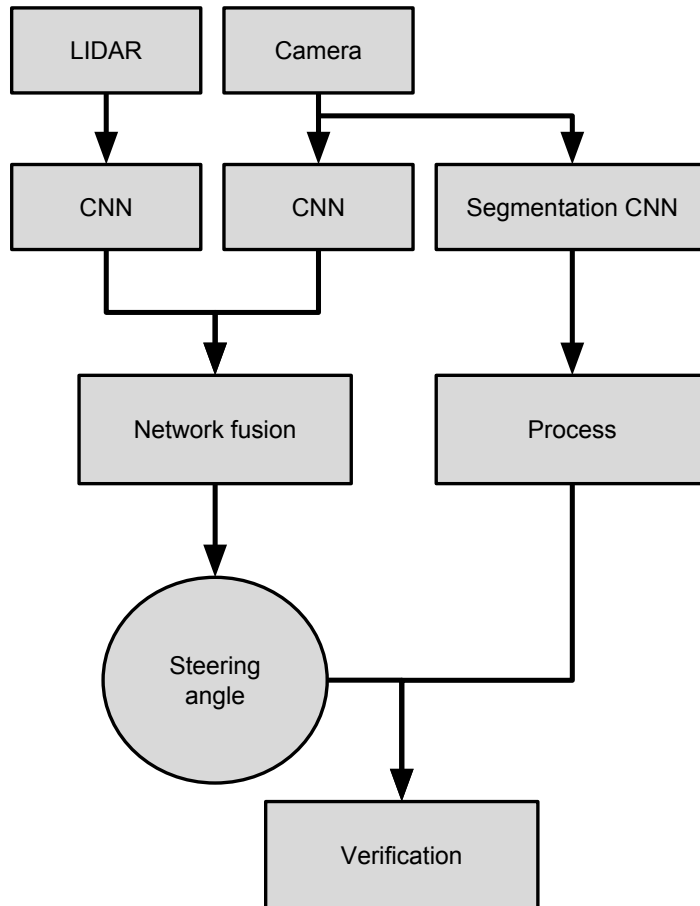


Figure 1.1: The system takes both regular camera images and LiDAR data as input. These are processed through two CNNs before predicting a steering angel. Parallel to this is a segmentation CNN that uses camera images to segment out the road ahead. This segmentation can be used together with the predicted steering angle in a path verification module.

1.3 Background

External supervisor Narada Warakagoda and Forsvarets Forskningsinstitut (FFI) introduced a hypothesis of using multi-input neural networks in an end-to-end approach to steer an Unmanned Ground Vehicle (UGV). This thesis will investigate the use of the end-to-end approach using both camera images and LiDAR data as input, compared to only using camera or LiDAR data. All this in an off-road environment. In addition to providing the task itself, Warakagoda provided information about the fundamentals of the end-to-end approach, and neural network in general. He also provided the TensorFlow software used as a starting point for developing the end-to-end network presented in Chapter 3, along with the unprocessed dataset. This network is covered later in Chapter 3 as the steering angle network. The other network trained and used in this project was a segmentation network. Since this network only was to be used to present a possible path verification technique, a suitable architecture was adopted from gitHub¹ and modified to fit the task initiated in this thesis. Supervisor, Kristin Y. Pettersen equipped the project with a suitable workstation. This workstation was necessary to train the networks in Chapter 3.

1.4 Contributions

- A comparison between single-input end-to-end networks using camera or LiDAR, and multi-input networks using both - on dirt roads. This thesis demonstrates that multi-input end-to-end networks outperform single-input networks, which can give the system better hardware redundancy and along with improved resistance to sensor noise like sun flair, rain and snow, and reflections.
- A path verification technique using a segmentation network is presented. It uses the segmented road and combines it with the predicted steering angle from the end-to-end network to evaluate the local path of the vehicle.

¹Source: <https://github.com/maxritter/SDC-Semantic-Segmentation>

1.5 Related work

This section reviews central literature relevant to the work done during this project.

1.5.1 Single-input end-to-end networks

[1] by Nvidia used the end-to-end approach for steering angle prediction on a real car. Nvidia's paper explains how they normalize all pixels before feeding them into five convolutional layers followed by three fully connected. For better centering of the vehicle, they use left- and right-positioned camera. Images from these cameras are used to mimic an off-center position during training. They perform successful tests on highways. Similar to the Nvidia project is the work done by Comma.ai [2]. They use three convolutional layers followed by three fully connected layers to map images of roads to steering angles. Note that [2] does not have a research paper. Their research contains only the software developed by the company, and no design choices were explained or documented. The CEO Hotz did a more general description of their goal and work during *AI by the bay*, an AI conference [4]. [5] also use the end-to-end approach. They present an architecture base on the VGG-16 model [6] with additional seven convolutional layers, and two max pool layers. It should be noted that they use this architecture for both steering angle prediction and object detection. Furthermore, they add batch normalization in the last layers for faster convergence [7]. Another technique is the use of temporal data, in [8, 9] they present a network with recurrent nodes using Long Short Term Memory (LSTM). In [9] they also compare their model with other models for comparison using a benchmark dataset.

1.5.2 Multi-input end-to-end networks

Combining camera images and LiDAR in an end-to-end matter has been proven to improve autonomous driving performance [3]. In [3], a neural network processes both camera images and LiDAR data for steering angle prediction on public roads.

They experiment with point cloud mapping and the use of PointNet. An architecture initially made for handling segmentation tasks with LiDAR data [10]. The model uses the Pointnet architecture in [10], and combines it with a CNN by using a fully connected fusion network. They also present a similar model with a LSTM instead of a fully connected fusion layers for sequential data handling. A similar model for object detection has been developed in [11] to be used for vehicles.

1.5.3 FCNs for semantic segmentation

Fully Convolutional Neural Networks (FCNs) has been used to segment road from background in Mulitnet [12]. They utilize the architecture described in the theory chapter Section 2.3 for semantic segmentation of public road and background (non-road). The main reason for using a FCN for segmentation was the small computational time a forward iteration uses, together with its accuracy [12, 13]. The paper focuses on fast real-time performance, which is necessary for an autonomous driving system.

1.6 Outline

The report is organized as follows. In Chapter 2 one can find a short introduction to the theory used in the later chapters. Chapter 3 covers the methodology and approach used to tackle the tasks in this project. Chapter 4 includes the results, discussion, and conclusion. Appendix A, includes (as an attachment zip-file) the scripts used in this thesis. Finally, in Appendix B, a possible technique for steering angle prediction only using segmentation is presented.

Chapter 2

Theory

This chapter will briefly cover some of the theory behind the DL technologies used in building and training the networks, and this will establish the foundation of the later networks in Chapter 3.

2.1 Artificial Neural Networks (ANNs)

Artificial Neural Networks (ANNs) have been around since the early 1940s. However, it took decades before they were improved enough to be efficiently used for machine learning tasks. Real breakthroughs did not come into existence before early 2010s [14]. This was a result of further development of techniques and using GPUs for faster calculations [15, p.440]. There are many different variations of the Artificial Neural Network (ANN), both in depth, width, input, output, and general architecture. A typical network used when input data are images is called a Convolutional Neural Network (CNN). To make a new layer in a CNN, one can apply a small filter with weights which slide along the input image. When it slides, it multiplies itself with the corresponding pixel values on the input image. When this is done all over the

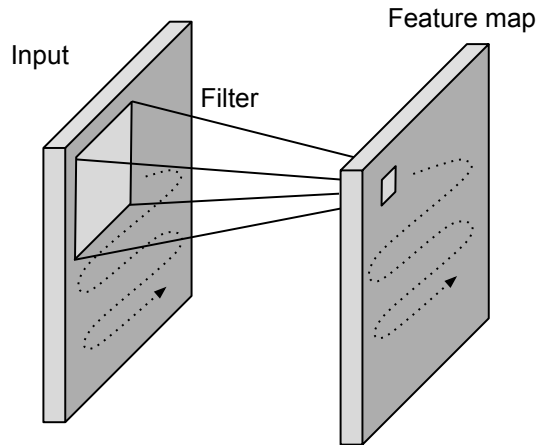


Figure 2.1: Example of a convolution which outputs one feature maps.

input image, the output is called a feature map. This multiplication of input and filter can be done several times to create multiple feature maps. Using filters preserves the spatial relationship between nearby pixels. By doing this, it extracts small but essential characteristics out of the input image. An illustration of a convolution is to be found in Figure 2.1, a patch of nearby pixels are mapped to the next layer. An ANN can output a variety of different tensor shapes. CNNs are often used for classification tasks where the output is a vector, and the highest value of a position indicates the associated class. They can be build up by several convolutions layers, pooling layers which down-samples a patch of the input, and fully connected layers. Fully connected layers are where each node in a layer is connected to all nodes in the next layer. However, ANNs can also be used for regression problems. When dealing with a regression problem, the single output node returns a continuous value. An example figure of an ANN for regression is illustrated in Figure 2.2.

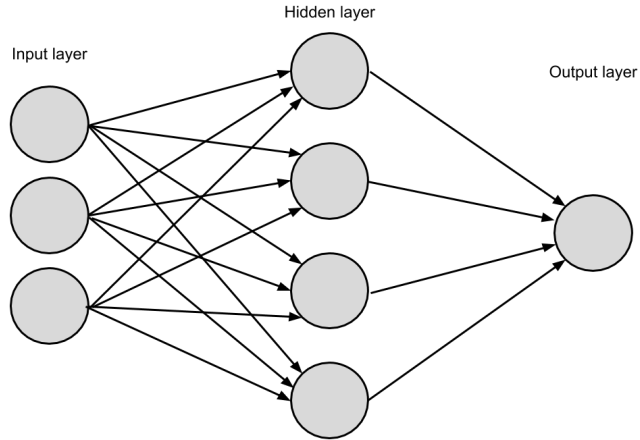


Figure 2.2: Simple ANN with one output node for regression. Last node can output a continuous range of values.

2.1.1 Loss functions

A loss function is used in neural networks to describe how far off the produced result was from the expected result. The error is then used to minimize the loss using an optimizer. Models that deal with a regression problem outputs a value and not a class. To measure the performance of these networks, one can use Mean Square Error (MSE), expressed as

$$MSE = \frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2 \quad (2.1)$$

where \hat{Y} is a vector with the predicted values, and vector Y is the correct and observed values to verify with. The MSE gives the optimizer a measure on how far off it currently is. Similarly, one can also take the root of MSE, and this is called Root Mean Square Error (RMSE). Models for semantic segmentation often use softmax cross-entropy in a pixel-wise fashion as loss function. It calculates the difference between the predicted

class and the label class for each pixel. The cross-entropy function expressed as

$$D(S, L) := - \sum_i \sum_c l_i(c) \log(s_i(c)) \quad (2.2)$$

where S is the two-dimensional array of pixels and classes, where softmax operation (sum across both classes is one) has been applied across the class axis. L is the two-dimensional label array. i iterates over all pixels and c over all classes. $l_i(c)$ is the class label indicating whether the i th pixel belongs to the class c . It is 1 if this is true and 0 otherwise. $s_i(c)$ is the softmax output for class c of the i th pixel.

2.1.2 Optimizers

The key aspect of training a multi-layered network is to find the change in error with respect to a specific weight so that the loss is minimized [16]. By using the concept of gradient descent with an optimizer, the algorithm can propagate backward from the loss function to adjust weights [16]. A simple optimizer that utilizes the gradient descent is Momentum. It adds a fraction of the previous update vector in the current one to accumulate momentum towards the local or global minimal. It is formulated as

$$\begin{aligned} m_t &= \gamma m_{t-1} + \eta \nabla_{\theta} J(\theta) \\ \theta &= \theta - m_t \end{aligned} \quad (2.3)$$

where θ are the parameters of the cost function J , m_t is the current updated vector, γ is the momentum term, and η is the learning rate. A common and more advanced optimizer is the Adam optimizer [17]. Adam stands for adaptive momentum estimate. The algorithm uses the principle of gradient descent to do a step towards ideal minima, similar to Momentum. However, [17] explains how the optimizer calculates individual learning rates for each parameter and the adaptive momentum values for these. It determines the moving average of the gradient and squared gradient and calculates the adaptive learning rates using these based on the parameter. These exponential moving

averages, called the mean and the uncentered variance, are formulated respectively as

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) \nabla_{\theta} J(\theta) \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) (\nabla_{\theta} J(\theta))^2 \end{aligned} \quad (2.4)$$

where β_1 and β_2 control the decay rates. m_t and v_t are initialized as zero which causes them to be biased towards zero. This is corrected with

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \end{aligned} \quad (2.5)$$

which is then used in the updated function

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \quad (2.6)$$

where ϵ is a small number to prevent zero-division. Their study show how Adam gives a better performance than other optimizers in most cases.

2.1.3 Activation functions

Non-linear activation functions enable the neural network to solve non-linear tasks. Activation functions are usually designed to take smaller values closer to zero as inputs to speed up learning [18]. Rectified Linear Units (ReLU) has been the industry standard of activation functions because of its simplicity and speed [19, 20]. It is also replicate the real-life scenario of a neuron firing, which in a nutshell is what inspires the use of ANN. ReLUs pass negative values as inputs, if zero or below, these nodes are deactivated. This can in some cases cause dying neurons which is related to the

vanishing gradient problem. It is expressed as simple as

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0. \end{cases} \quad (2.7)$$

In 2017 a paper was released describing a new and effective activation function called Exponential Linear Units (ELU) [21]. They presented an activation function that reduces the vanishing gradient problem. According to [21], "Mean shifts toward zero speed up learning by bringing the normal gradient closer to the unit natural gradient because of a reduced bias shift effect" (p. 1). The ELU activation function is plotted later in Section 3.3, and is formulated as

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{cases} \quad (2.8)$$

2.1.4 Regularization

A common problem in ANNs are overfitting. Some weights are overstimulated on the training set which causes the poor performance results on the test set. One regularization technique is weight decay, also known as L^2 regularization. It penalizes the loss function depending on the current weight. This gives the new loss function

$$\hat{E}(w_{ij}) = E(w_{ij}) + \frac{\lambda}{2} w_{ij}^2$$

where $E(w_{ij})$ is the old loss function and, λ is the penalty hyperparameter. A similar regularization technique is Dropout. A pre-specified amount of randomly chosen neurons are deactivated for each iteration. This avoids overly stimulated weights and allows for better generalization of ANNs. Another regularization technique is Batch Normalization [7]. It has a regulating effect, and additionally, it helps speed up the learning by hindering explosion of weights and keeping them within the range of the activation function [7]. Batch Normalization is usually placed after activation

functions, and it can either standardize or normalize the values into a specific range. Batch normalization in models enables larger learning rates and allows the user to be less careful about parameter initialization [7].

2.2 The end-to-end approach

The end-to-end approach is used in self-driving systems for predicting steering angles. It usually takes images from a front-facing camera and uses a CNN to map these image frames of roads directly to steering angles. This approach is called end-to-end because it handles everything from feature extraction to path prediction, end-to-end, without any additional modules. The network maps individual camera images directly to steering angles through the single node output. These models, which output a continuous value, are called regression models, and Figure 2.2 illustrates an example of this type of output. End-to-end models can concertize and distinguish essential features of the road ahead because they have been trained to generalize features of the road, such as road boundaries and curvature.

2.3 FCNs for semantic segmentation

Fully Convolutional Network (FCN) can be used for semantic segmentation and has proved powerful with the right techniques [13]. Semantic segmentation is a way of labeling images by giving each pixel a class. When using FCN in semantic segmentation, a CNN extract features just like a regular CNN. A FCN model does not have fully connected layers. Instead, it stacks convolutional layers that produce a smaller and smaller heat map of the target. Since there are no fully connected layers, the spatial representation is kept. When the heat maps are created, de-convolutional layers are used to reconstruct the original size of the input image. De-convolutional layers can learn by backpropagation, similar to a normal CNN. In contrast to a regular CNN, the FCN model uses a pixel-wise loss function like Formula 2.2 presented in Section 2.1.1.

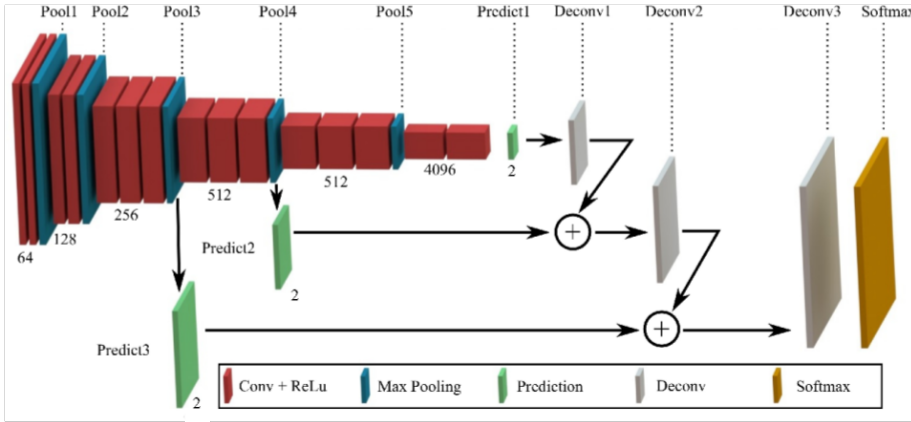


Figure 2.3: Architecture of a Fully Convolutional Network used for semantic segmentation. Width and height of the output are the same as the input image. This model can classify pixels of two classes.¹

De-convolutional layers can also handle non-linearity when up-sampling an image. [13] implement this architecture with skip layers. This is a way of more accurately predicting the segmented areas by merging the up-sampled image with the appearance information of early layers whose heat maps are of higher resolution. The result is a fast segmentation network (because of only using convolutions) that performs well on different segmentation datasets [13]. An example of a FCN architecture for semantic segmentation can be seen in Figure 2.3.

2.3.1 Evaluation metrics

To evaluate the performance of the segmentation model, Average Precision (AP) was used. Segmentation tasks can be evaluated using several different metrics with different characteristics like those used in [13], but AP is used by Multinet in [12], and to easier compare the results it was also applied when evaluating the segmentation network

¹Image source: <https://goo.gl/Fcnxxf>

in the methodology chapter, Section 3.4. AP use a confusion matrix to calculate the precision using True Positives (TP) and False Positives (FP), and the recall using TP and False Negatives (FN). Precision is defined as $\frac{TP}{TP+FP}$ and recall is defined as $\frac{TP}{TP+FN}$. AP calculates the area under the precision/recall curve. In this thesis, it implemented using the scikit-learn² Application Programming Interface (API), and is expressed as

$$AP = \sum_n (R_n - R_{n-1})P_n \quad (2.9)$$

where R_n and P_n are the precision and recall at the n th threshold. It should be noted that Multinet uses a slightly different AP formula called the 11-point interpolating AP. It is used by the Pascal Visual Object Classes evaluation and is defined in their own paper [22]. The small metric difference does not matter in this thesis. The main goal of the segmentation was not to achieve the highest possible score, but to present a possibility of using a segmented road for path verification.

²Documentation Source: <https://goo.gl/WzJZjY>

Chapter 3

Methodology

This chapter consists of five sections. The two first are a description of tools and datasets. Following is a section for the end-to-end model which is named steering angle network. Next, a section of the segmentation network. Finally, a section for the path verification technique. The scripts used in this part of the project can be found as an attached zip-file in Appendix A.

3.1 Tools

This section covers the software and hardware appliances used to develop and utilize the neural networks.

3.1.1 Workstation

The CPU is the main processing unit in a computer. It handles a variety of different tasks and often uses multiple cores to parallelize them, a GPU, on the other hand, is more specialized. It is primarily used to handle heavy graphics. Because it has

significantly more cores, it can handle certain simple tasks fast and parallel. ANNs are computationally heavy with thousands or maybe millions of weights and biases that are handled. ANNs have in the later years dramatically increased in size and forced the need for GPUs in training. The workstation used in this work had four parallel GeForce GTX 1080 Ti graphics cards and a MD RYZEN THREADRIPPER 1900X 4.0GHz. Each GPU has 11 GB of VRAM, and more importantly, they have a memory bandwidth of 484 GB/sec each.

3.1.2 TensorFlow

The models build in this project were build using TensorFlow¹ and TensorFlow Slim², a machine learning API made by Google. It enables developers to implement formulas, DL techniques, and other machine learning features fast and easy. Neural networks are built as computational data flow graphs with TensorFlow, where nodes are computations and edges are multidimensional arrays storing weights and biases [23]. There are other APIs which are easier and simpler to use and build with, but TensorFlow allows for better control of data flow and architecture. GPUs were used for faster computations. NVIDIA has created a platform named Compute Unified Device Architecture (CUDA) which allows developers to use the computational power in the GPUs for general purpose tasks. By installing CUDA on the workstation, TensorFlow could access the GPUs memory and computational features for faster training. As mentioned in Section 3.1.1, because it has multiple cores it can handle certain simple tasks fast and parallel.

3.2 Datasets

A time-consuming part of the project was to prepare the dataset used to train the TensorFlow models. Open and free datasets for self-driving systems exist, but none

¹Documentation source: https://www.tensorflow.org/api_docs/

²Documentation source: <https://goo.gl/cQJJcm>

of them were recordings from a dirt road environment. Therefore, images and data collected by FFI were prepared to be used in the different models.

A vehicle with mounted cameras, a LiDAR and sensors were driven around in different environments recording data to be used in the dataset. Most of the data recorded were dirt roads and trails. While recording, camera images, LiDAR data, and steering angle were continuously saved. The camera rig was made up by three cameras, all pointing forwards. One camera was centered, and the two others were positioned on edge on each side of the vehicle. Both side cameras left and right was originally used for stereo recording and recorded images in grayscale. The data were saved with the help of Robot Operating System (ROS), which saved all data into ROS-bags, including the associated steering angles. Combined, all ROS-bags occupied a total of 2.5 TB. All images captured by the front facing center camera was saved in BAYER format. By using this format, the size of an image was stored in one-third of its original size [24]. These ROS-bags were opened and read with ROS, *rosvbag*, *CvBridge*, and *openCV*. Down below is a code snippet of an example program that reads and saves frames from the */camera/center* topic. The other topics used in this project were */camera/stereo_left*, */camera/stereo_right*, */lidar_sweeps*, and */olav/vehicle_measurements*.

```
import rosvbag
import cv2
from cv_bridge import CvBridge

filenm = 'test.bag'
bag = rosvbag.Bag(filenm)
i=0
img = []

for topic, msg, t in bag.read_messages(topics=['/camera/center']):
    img = CvBridge().imgmsg_to_cv2(msg, desired_encoding="passthrough")
    cv_image_rgb = cv2.cvtColor(img, cv2.COLOR_BAYER_RG2RGB_EA)
    cv2.imwrite('img_%s.png' % i , cv_image_rgb)
    i=i+1
```

3.2.1 Dataset for steering angle network

All data saved in each rosbag were saved with timestamps in nanoseconds. The recording rate was 6 Hz for the cameras, 10 Hz for the LiDAR and 50 Hz for the steering angle sensor, but the timestamps were not synced. This is expected due to minor delays in different processes of hardware and ROS. Therefore, a script was made that would pick out the timestamp of a data and find the timestamp for a steering angle that would be closest to it in time. The script gathered data from the rosbags as explained in the previous section, and saved camera images, LiDAR data, and steering angle labels in folders. To ensure that the dataset only contained data collected from a moving vehicle, all data recorded when the vehicle had a ground speed below one kilometer per hour was discarded. When playing back camera frames one could from time to time observe the vehicle stopping or parking. Data from a stationary vehicle would pollute the dataset which is only meant to contain data of a moving vehicle operated by a human.



Figure 3.1: Before and after applying hot encoding on the camera images.

It was decided to convert all image frames taken by the center camera and converting them to grayscale. This was to match the left and right camera so images taken by all three cameras could be used in the same neural network, interchangeably. Next, all image frames were colored by using hot encoding. Coloring was done to make

sure that the data would fit the pre-trained net used later in this chapter. Figure 3.1 illustrates this colorization. This tripled the size of the dataset. Additionally, steering angles belonging to the left or right camera was modified so the steering angle would tilt towards the center of the road. The idea was to learn the model to keep the vehicle centered, comparable to what [1] did. All image frames were resized to take up less space and fit the CNN. All camera images were divided up into a training set and a test set where the test set made up 20 %. LiDAR data collected from the rosbags were at first of shape $(32,723,2)$, height, width, and the last dimension represent distance and intensity. To get it to fit the input of the CNN, the LiDAR data frames were reshaped with TensorFlow's *resize bilinear* function to $(32,224,2)$. The first channel, representing distance, was duplicated and placed as the third channel. The final shape of all LiDAR data where $(32,224,3)$. The data frames were then divided up into a training set and a test set where the test set made up 20 %.

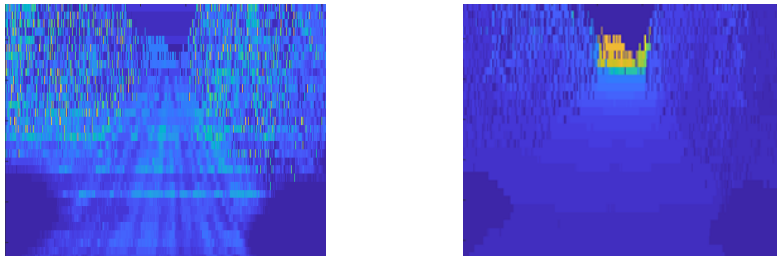


Figure 3.2: Samples from the LiDAR dataset where pixels in the left figure represent the distances to the environment, and the pixels in the right represent reflection intensity.

For fastest possible data handling and pipeline, TFrecord was used. That is a feature of TensorFlow that makes data into binary files, which can be fed directly into the GPU. This enhances the training speed since the CPU no longer have to handle all data before feeding it to the model. Two datasets were made with TFrecord files. One was made from the camera images, and the other one was made from the LiDAR data, each with 142,013 samples and both with synchronized steering angles.

Table 3.1: Dataset for steering angle network

	Training	Testing	Total
Camera images	113,613	28,400	142,013
LiDAR data	113,613	28,400	142,013

3.2.2 Labeling data for segmentation network

For the segmentation network, 500 RGB color images of dirt road were prepared. The images were handpicked from different rosbags to match the expected environment the system most likely is meant to be used in. The intention was to collect a variety of non-asphalt roads in different conditions and with different degree of turns on the road in the image.

Labeling data for the segmentation network was a time-consuming job. A program with a user interface was written to speed up the dataset production, and at the same time make each label as accurate as possible. The program took each of the 500 original images, one by one, and presented them to the user. By using grabCut, an algorithm for foreground extraction [25], one could easier segment road and non-road. After the algorithm was done cutting out the road in the image one could use drawing tools to fine-touch and finalize the image and correct the inaccuracy of grabCut. When done, the dataset would consist of a folder of the original images, and another folder with the label images. All label images in this folder would have binary color, white for road or black for not road. Figure 3.3 illustrate an original image meant to be fed into the segmentation network, and label image meant to be used by the loss function to train the network.

Table 3.2: Dataset for segmentation network

	Training	Testing	Total
FFI	400	100	500
KITTI	398	200	598



Figure 3.3: The original RGB color image to the left, and the associated two-class label image to the right.

3.2.3 Existing datasets

In Section 3.4 the segmentation network is presented. To extend the self-made dataset, the KITTI dataset by Karlsruhe Institute of Technology was included. It consists of 398 training images and 200 test images taken from a city and highway environment [26]. The labels contain multiple classes of segmented objects and areas, but it is only the road label and background (non-road) which is of this thesis' interest. Another dataset used, here indirectly, was the ImageNet from the ImageNet Large Scale Visual Recognition Competition (ILSVRC) 2015. When the VGG-16 model used in Section 3.3 and 3.4 were downloaded, they were pre-trained on ImageNet. They were downloaded pre-trained from TensorFlow's model library³. The dataset contains images of 1000 different classes used in image classification tasks. The pre-trained model has adapted features like edges, colors, and shapes which can give the networks in this thesis a head start when training on their own dataset. This is called transfer learning.

³Download source: <https://github.com/tensorflow/models/tree/master/research/slim>

3.3 Training steering angle network

This section deals with the steering angle network which is built using the end-to-end approach. It handles both camera images and LiDAR data as input and maps it to a steering angle. It consists of two subnetworks, one for the camera images, and one for the LiDAR data. In the first following subsection, the camera subnetwork is covered, followed by a subsection about the LiDAR subnetwork. In the final subsection, the two subnetworks are fused into the full model which handles both inputs parallel. The two subnetworks, camera network, and LiDAR network were first trained individually. When training the subnetworks, both camera images and LiDAR data had the same timestamp. This eliminates the environmental differences in the two datasets. The TensorFlow software provided by Warakagoda was used as a starting point for developing the steering angle networks. It was initially used for classification tasks and had to be modified to fit this problem.

As mentioned in Chapter 2, steering angle prediction with CNNs has shown effective [1, 2, 3]. By using camera images combined with LiDAR data, the objective was to enhance the performance compared to single data networks. Camera images provide patterns and colors, while LiDAR provide depth and object understanding to the model. Here, both data types use its own CNN to extract features before they are fused together for prediction of the correct steering angle. The two prior CNNs are from now on named subnetworks. Figure 3.4 illustrate the principle behind this idea. Ultimately, the goal is to enable the vehicle to drive by itself by *looking* at the road ahead, without explicit lane detection nor path controllers. Doing this on a dirt road or trail is assumed to be a challenge. Unlike with asphalt roads, colors on the ground on dirt roads and trails do not have a clearly defined meaning. Slopes frequently change, and roads are not level. The model is trained by using supervised learning, cloning of human car maneuvering, collected in the datasets presented in section 3.2.

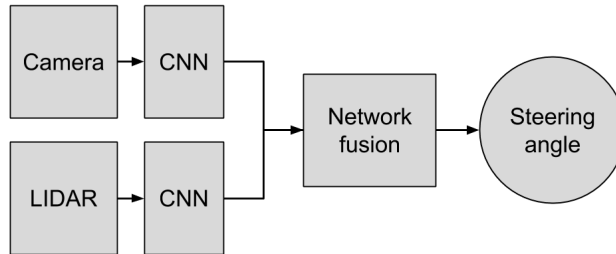


Figure 3.4: Overview of the steering angle prediction model. Camera and LiDAR frames are each fed through a CNN subnetwork before fused together to predict the steering angle. These subnetworks are called camera subnetwork and LiDAR subnetwork.

3.3.1 Training camera subnetwork

The camera subnetwork was first trained by itself, isolated from the rest of the system. The subnetwork was fed with images recorded by the three forward facing cameras which are placed on the roof of the UGV. The grayscale images are color mapped with a hot encoder. More information about this dataset can be found in Chapter 2, Section 3.2.1. Samples from this dataset taken by the front-facing cameras can be seen in Figure 3.7. Each image in 3.7 has a tag with the corresponding steering angle. The goal was to predict the steering angle of similar samples as accurate as possible. As seen in the samples, the subnetwork has to handle different conditions such as changing light and soil type.

Architecture

Since similar work have used fully connected layers [1, 2], the first experiment also included fully connected layers in the last part of the subnetwork. The first experimental architecture was inspired by Nvidia [1] and used a pre-trained VGG-16 model for extraction followed by three fully connected layers. The model started out with the Adam optimizer and the MSE as loss function. However, during testing and

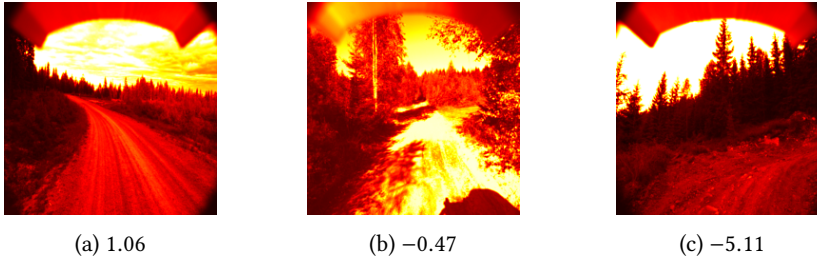


Figure 3.5: Different preprocessed samples from the training set showing variations in weather and lighting conditions, terrain, and soil. The value under each sample indicate it's corresponding steering angle value. Positive values represent a left turn, and negative a right turn.

experimentation, a better result was achieved using the standard Momentum optimizer. After several unsatisfactory trainings with this the architecture it was changed to the to the architecture presented by [5]. The training convergence improved when comparing the first model with the new model. After this small experiment, it was decided to use the model presented in [5] as a base for the camera subnetwork. The first part of the steering angle model was built from a pre-trained VGG-16 architecture. Transfer learning was used to get faster convergence towards acceptable loss values. All layers up to the forth max pool layer of the VGG-16 model was used. Following this is the new prediction section of the model, adapted from [5]. This new section was later connected with the fifth max pool layer, which included three additional convolutional layers. Using these additional layers resulted in a loss drop of 50 %. The final architecture can be found in Table 3.4. It is divided into an upper section made by the pre-trained VGG-16, and the lower section called the predictor part.

An attempt was made to implement ELU for faster learning in the predictor part of the model [21]. It proved difficult to observe a distinct improvement in loss convergence. Tests showed that the final loss was indistinguishable from a model only using ReLU. Since the pre-trained VGG-16 section of the model already uses ReLU, it was not possible to experiment with other activation functions there. In the end, ReLU was

implemented in the last layers as well, so not to mix activation functions within the model. The ReLU's activation function illustrated as blue in Figure 3.6, may result in dying neurons, however, through testing and experimentation ReLU turned out to provide an acceptable result. The last output layer of the model presented in [5] used a tanh activation function. All three activation functions are plotted in Figure 3.6. The model presented here does not include an activation function in the final layer to avoid the predicted steering angle to be bounded by the range of the activation function. To sum up, all activation functions are ReLU except the last node where it is linear.

The model was implemented with two regularization techniques to prevent overfitting. The first regularization technique used was weight decay, also known as L^2 regularization. The L^2 parameter was set to $4e - 5$, The second is batch normalization. Since the

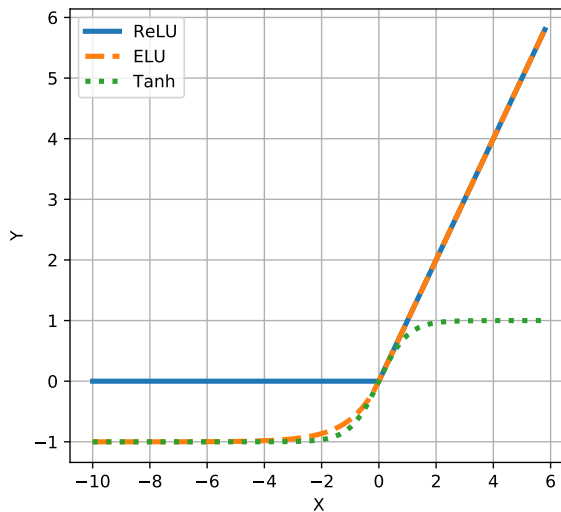


Figure 3.6: Comparison between ELU, ReLU and Tanh. ReLU where used in the steering angle model. Tanh where used as output activation function in [3].

target label is standardized and input images are normalized, batch normalization was implemented after all convolutional layers in the last prediction part of the model. Unfortunately, VGG-16 was created before batch normalization [6, 7]. This means that batch normalization layers could not be added in the upper, and already pre-trained, VGG-16 section of the model.

Data handling and tuning

The images were collected from the TFrecord files and randomly shuffled before preprocessed and fed into the network. The input images were normalized from 0 to 1, as mentioned in Section 3.2, to fit the ReLU activation functions. During training, both training loss and validation loss were plotted to determine easier which parameters to tune. One could also observe the validation curve to find a suitable number of epochs. The final hyperparameters are listed in Table 3.3. When training on the full dataset a batch size of 20 was used. Bigger batch size was preferable because training is faster, but experimentations were limited by GPU resources. Learning rate was first set to $1e - 4$ but triggered a loss of *nan* because of exploding weights. $1e - 6$ proved slightly too small. $1e - 5$ was therefore chosen as the final learning rate.

Table 3.3: Hyper parameters steering angle network

Batch size	Learning rate	Epochs	$L^2Reg.$
20	$1e-5$	100	0.00004

3.3.2 Training LiDAR subnetwork

The LiDAR subnetwork was first trained by itself, isolated from the rest of the system. The subnetwork was fed data recorded by the LiDAR sensor on the roof of the UGV. The first channel of the LiDAR frame, which represent the distances to the objects, where duplicated and stacked behind the two first channels to fit the input shape of the

Table 3.4: Subnetwork used in steering angle network. This architecture were used by the camera subnetwork, and the LiDAR subnetwork. The line separates the pre-trained VGG-16 and the new predictor section. (a) indicates the only difference between the two subnetworks. (b) indicates the fusion network input.

	Layer type	Features	Kernel	Strides	Activation	
VGG-16	Conv2D	64	3x3	1x1	ReLU	
	Conv2D	64	3x3	1x1	ReLU	
	MaxPool2D	-	2x2	2x2	-	
	Conv2D	128	3x3	1x1	ReLU	
	Conv2D	128	3x3	1x1	ReLU	
	MaxPool2D	-	2x2	2x2	-	
	Conv2D	256	3x3	1x1	ReLU	
	Conv2D	256	3x3	1x1	ReLU	
	Conv2D	256	3x3	1x1	ReLU	
	MaxPool2D	-	2x2	2x2	-	
	Conv2D	512	3x3	1x1	ReLU	
	Conv2D	512	3x3	1x1	ReLU	
	Conv2D	512	3x3	1x1	ReLU	
	MaxPool2D	-	2x2	2x2	-	
	Conv2D	512	3x3	1x1	ReLU	
	Conv2D	512	3x3	1x1	ReLU	
	Conv2D	512	3x3	1x1	ReLU	
	MaxPool2D	-	2x2	2x2	-	
Predictor	Conv2D	256	3x3	1x1	ReLU	
	Batch Norm.	-	-	-	-	
	Conv2D	128	3x3	1x1	ReLU	
	Batch Norm.	-	-	-	-	
	Conv2D	256	3x3	1x1	ReLU	
	Batch Norm.	-	-	-	-	
	MaxPool2D	-	2x2	2x2	-	← (a)
	Conv2D	128	3x3	1x1	ReLU	
	Batch Norm.	-	-	-	-	
	MaxPool2D	-	2x2	2x2	-	
	Conv2D	256	3x3	1x1	ReLU	
	Batch Norm.	-	-	-	-	
	Conv2D	512	4x4	1x1	ReLU	← (b)
	Batch Norm.	-	-	-	-	
Conv2D	1	1x1	1x1	Linear		

model. More information about this dataset can be found in Section 3.2.1. An example of LiDAR data is illustrated in Figure 3.2.

Architecture

The network architecture used for the LiDAR subnetwork was almost identical to the model used by the camera subnetwork covered in the preceding section, Section 3.3.1. This means that Table 3.4 also applies to this model. The difference was that the max pool stride was increased to (4,4) in the last max pooling layer marked (a), this was to decrease training time, and reduce the risk of overfitting. Just like regular RGB color images is the LiDAR data built up by a 3-D array of width, height, and depth. Since the camera subnetwork achieved satisfactory results with its architecture, it was adopted for the LiDAR subnetwork. In contrast to [3] is the LiDAR data handled as regular images in the CNN.

Data handling and tuning

Similar to the camera subnetwork, where also the LiDAR data for the LiDAR subnetwork normalized. Similarly, these were collected from TFRecord files and randomly shuffled before trained on. The LiDAR data used to train this network had the same timestamp as the camera images used to train the camera subnetwork. The hyperparameters were set to the same as it was for the camera subnetwork, see Table 3.3. Since reasonable parameters were found in the camera subnetwork, and they worked on this subnetwork, the hyperparameters were kept as is for simplicity. No further tuning was done.

3.3.3 Training fusion network

The subnetworks were first trained separately, isolated from the rest of the system. The two matching datasets with synchronized timestamps, camera images, and LiDAR

data, were simultaneously fed into the two inputs of the full steering angle network in an attempt to improve the results.

Architecture

The two subnetworks were kept as is all the way down to the second last convolutional layer marked as (b) in Table 3.4. The output of the camera subnetwork was a feature vector of size 2078, and the output of the LiDAR subnetwork was of size 1024, a total of 3072. These were then concatenated and fed to a fully connected layer of 100 nodes before ending up in the last single linear output node.

Data handling and tuning

During training of the full steering angle network the subnetworks were frozen, meaning they could not be trained further. The only trainable parameters were the new fully connected layers fusing the two subnetworks together. Camera images and LiDAR data used to train the two subnetworks, with the same timestamp, were collected from the TFRecords files. The data was pushed through the network in with the same batch size as for the two subnetworks. It was trained for 100 epochs, where the 50th epoch achieved the highest score. The fully connected layer and the output node were smaller than the subnetworks and needed fewer epochs of training. It was trained as the two subnetworks, with the same optimizer, weight decay, learning rate and, batch size.

3.4 Training segmentation network

This chapter deals with the segmentation network isolated from the rest of the system. A brief introduction to the theory is described in the background chapter, section 2.3. The TensorFlow script used in this part of the project was adopted from gitHub and

later modified. This is mentioned in Section 1.3

The architecture in segmentation network is based on the work of [13], using FCN for semantic segmentation. [12] used FCN on road segmentation using DL techniques in their Multinet model, which is what this segmentation task is aiming for. Using relatively few training images (289), they manage to accurately segment the road from the rest of the environment. The goal of this segmentation network was not to achieve highest possible score on the test set, but rather get a descent segmentation result which could be used to research the possibilities of a path verification. This experimental technique is covered in Section 3.5. The challenge in this project was to segment a dirt road from the environment, which can be hard because of the slightness of a distinct difference between road and not-road. Figure 3.7 illustrates four randomly selected samples the model was trained on. The model has to tolerate different weather and lighting conditions, terrain, and soil.

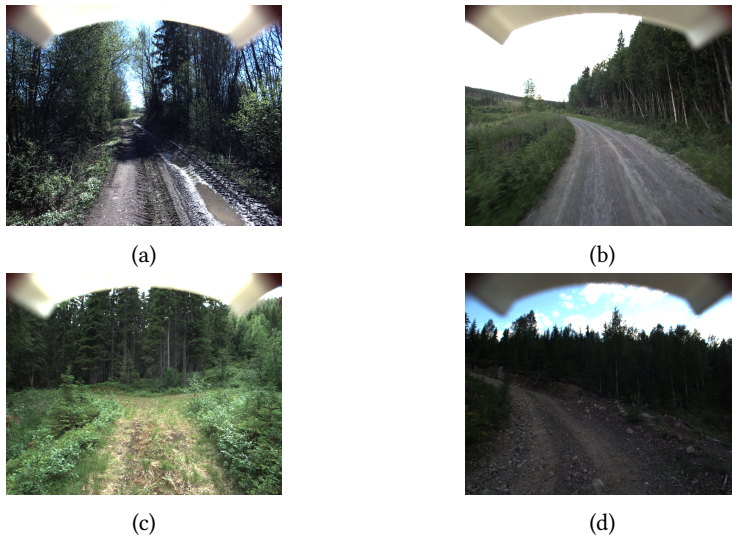


Figure 3.7: The model has to tolerate different weather and lighting conditions, terrain, and soil. Four different samples showing these variations.

Architecture

The FCN model adopted from gitHub was modified to fit the current task. This involved changing feature maps in skip layers to fit two classes, one for the road, and one for non-road, among other smaller modifications. It consists of a VGG-16 with 15 convolutions and three skip layers which de-convolutes the images back up to its original input size. The architecture is identical to the model presented in Figure 2.3. The input of the segmentation network where RGB color images with shape (224, 224, 3) taken from the center front facing camera of the vehicle. The output consists of feature maps consisting of two segments where one represent the background and one the dirt road, the height, and width of the output image are identical to the input image. The outputs were of shapes (224, 224, 2). For regularization, L^2 regularization was used. Considering the input images not being normalized and L^2 being the only regularization technique, the L^2 parameter was set to $1e - 3$.

Data handling and tuning

To handle the input data, a generator was created. The input images and associated label were loaded as PNG-files directly in this generator. In the generator, the samples were augmented with a slight color adjustment to replicate possible camera settings and light conditions. The generator would also 50 % flip the images horizontally to artificially increase the dataset. The 400 self-labeled training images of dirt roads were all slightly augmented. Some time was spent fine-tuning the data augmentation function, which would maximize the data set and help prevent overfitting. After viewing the 400 training images, it was decided to augment them with colors to replicate the various color conditions cause by weather and light. They were also augmented to replicate the color diversity caused by the camera settings of the camera pointing on the road ahead. The colored input images were therefore augmented with Gaussian noise, brightness, hue, and saturation. These augmentations did only change the images within what one can call a reasonable range, all augmented images still looked natural.

Training the segmentation model proved to be fast since it is only constructed of convolutional layers and handles low-resolution input images. Using the hyperparameters listed in 3.5 and the workstation described in Section 3.1.1, one training would take about 30 minutes. Therefore, one could efficiently optimize the hyperparameters by tuning and testing rapidly. Especially the fast forward iterations were one of the reasons for choosing the FCN architecture. It is fast enough to handle real-time image inputs [12]. The segmentation network was first trained on the KITTI dataset and later trained on the dirt road dataset. The idea was to reduce the amount of self-labeled images needed for segmentation, and overall increase the total amount of training data.

[13] used a batch size of 20, while [12] (Multinet) used a batch size of one. A batch size of 20 were chosen for this segmentation network as it was big enough to reduce noise produced by outliers. The learning rate ended up being $1e - 5$, just like Multinet. $1e - 4$ proved to be too big, and $1e - 6$ too small.

Table 3.5: Hyper parameters for segmentation network

Batch size	Learning rate	Epochs	L^2 Reg.
20	$1e-5$	40	$\lambda : 1e - 3$

3.5 Path verification

In this chapter, an experimental path verification technique is presented. The segmented image made from the segmentation model from Section 3.4 is used together with the steering angle from Section 3.3 to verify the local path ahead of the vehicle. The technique presented in this thesis is named path verification. During the research and testing of this technique, another steering angle prediction approach was formed. This was inspired by the Udacity self-driving car program [27] and is given in Appendix B.

The steering angle prediction approach was purely experimental and was only a realization of the idea to demonstrate its feasibility.

3.5.1 Pipeline

The model from the previous section (Section 3.4) returns a prediction of the segmented road. See Figure 3.8a. The binary version of this image is then processed further using computer vision techniques. It is smoothed to get rid of standalone pixels and sharp edges. Next, the perspective transformation is applied which is illustrated in Figure 3.8b. This gives a birds-eye view of the road ahead which makes it easier to apply and extract information from the segmented road. The path verification draws the steering angle from Section 3.3 directly on top of the perspective transformed segmented binary image as illustrated in Figure 3.8c and 3.8d. These figures are meant to demonstrate the idea, and are only imaginative. Considering this technique was for made for an experimental purpose can some parts of this be inaccurate, but accurate enough to demonstrate the idea. This applies to the lack of correction for lens distortion and perspective transformation.

Since the steering angle network outputs a single scalar for each input frame independently from other frames, the steering angle is drawn linearly in its current state. If the drawn predicted steering angle would fall outside the segmented area, it would indicate a poorly predicted steering angle. To adjust the strictness of the path verification, the range of the verification can be moved closer or further away from the vehicle. The verification only verifies the path up to this threshold and ignores the verification beyond this point. As seen in Figure 3.8a, the segmentation continues further up the dirt road, while only a certain distance ahead is used to verify the path. The threshold distance is meant to change dynamically depending on the speed of the vehicle. If the threshold is closer to the vehicle, the path verification is less strict, and if it is far up ahead, it is stricter. In this experimental test, it was set to a constant threshold for the simplicity to demonstrate the possibilities of path verification. Figure 3.8c

portrays a hypothetical verification that passes the test, while Figure 3.8d portrays a verification that fails. The intention of path verification is that this could be used to adjust the current steering angle to fall back into a safe path. By combining the results from each network, the idea is that using two DL techniques could potentially be used to enhance the overall performance of the system. The difference between each predicted steering angle used in the verification and its actual target is compared to the outcome of the path verification. This is to evaluate the path verification. If the distance between a predicted steering angle and the true steering angle were less than six degrees, and at the same time the path is within the associated segmented image, it would be considered successful. So by comparing the result from the test set

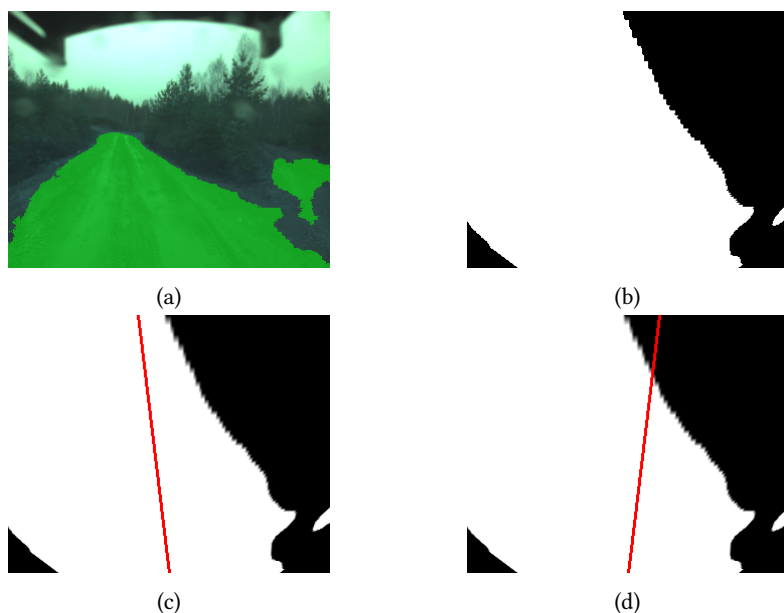


Figure 3.8: Original image with predicted segmentation overlay (a). Binary image of predicted segmented area with birds-eye transformation (b). The predicted steering angle is within the acceptable area (c). The predicted steering angle is outside the acceptable area (d).

of the steering angle network with the result of the path verification, it can indicate the performance of the path verification.

3.5.2 Verification test

A dataset of 270 new samples was sent through the full network from the two previous sections. Included in the dataset were also the associated target label. The multi-input steering angle network predicted the steering angles for each sample, and the segmentation network segmented out the dirt road on each of them. Each segmented image with its associated predicted steering angle were then pushed through the path verification. The verification returned a list of booleans which indicates if the steering angle fell outside of the segmented area or inside. This list, containing 270 elements, were then compared to a list containing booleans from the steering angle network. A boolean would indicate if the predicated steering angle were within or outside of the six-degree range.

Chapter 4

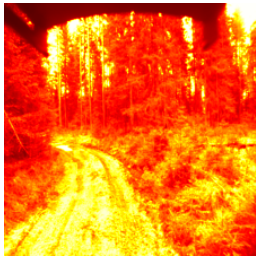
Results and conclusion

4.1 Results

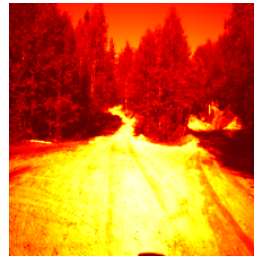
4.1.1 Steering angle network

The steering angle network was as mentioned made up by two subnetworks, one that handled camera images, and another that handled LiDAR data. The two subnetworks were each trained and tested individually before they were fused together and trained and tested again. The same data were used for testing, and the timestamp was synchronized across datatypes. All three trainings used a batch size of 40 over 100 epochs where the best epoch was kept. The networks were trained with MSE, but to evaluate the performance, normalized RMSE was used. This was used in [9], and makes it possible to compare results with their benchmark table. The camera subnetwork scored a RMSE of 0.2244 on the test set. This was slightly higher than the LiDAR subnetwork which scored 0.2236. The fused network, with both camera images and LiDAR data as input, scored 0.1926. Two prediction samples from the test set can be seen in Figure 4.1, where (a) obtained the lowest MSE score, and (b) the highest. The

training and validation losses for the three networks are plotted in Figure 4.2, 4.3 and 4.4. They visualize the training using MSE pr. epoch.



(a) MSE: $3.57e - 5$



(b) MSE: 33.45

Figure 4.1: Two predictions done by the steering angle network. (a) is showing an accurate prediction, and (b) showing an inaccurate prediction.

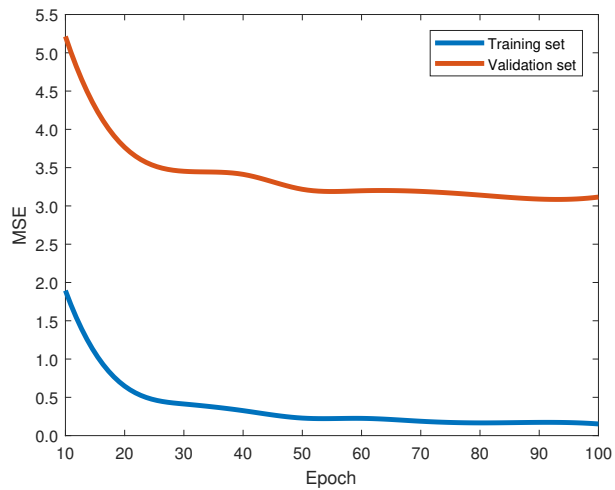


Figure 4.2: Training loss and verification loss for the camera subnetwork

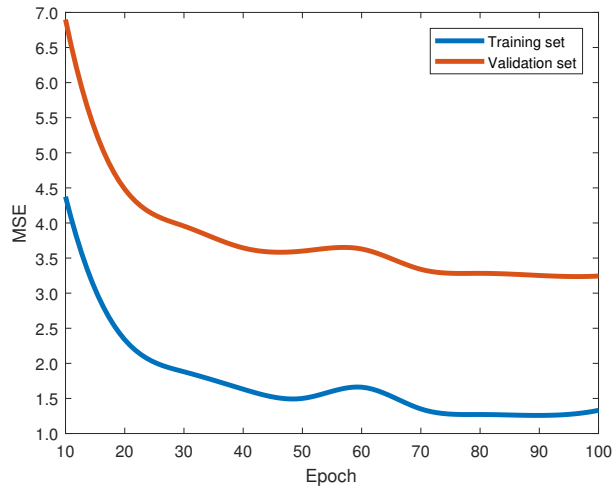


Figure 4.3: Training loss and verification loss for the LiDAR subnetwork

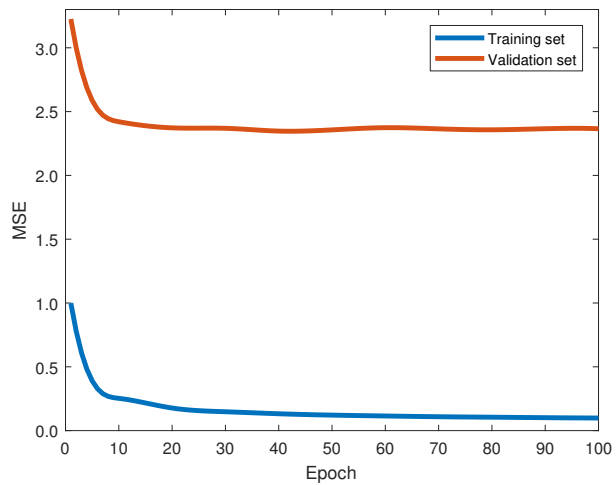


Figure 4.4: Training loss and verification loss for the steering angle network

4.1.2 Segmentation network

As mentioned earlier, the point of segmenting was to reach an acceptable level to illustrate the path verification technique in Section 3.5. The performance was measured quantitatively using the AP described in 2.3.1. The segmentation network scored an AP of 89.11 % on our test set, while Multinet scored 93.51 % on its. The main propose of this comparison is to give an impression of the results achieved. Note that there was a slight difference between AP formulas used, this is covered in Section 2.3.1. Figure 4.5 are four sample frames taken from the test set. The green overlay is the predicted road segmentation. Figure 4.5a and 4.5b represent two image frames with an accurate prediction overlay, both with a score over 90 % AP. The model handles shadows as seen in Figure 4.5a, and also up-coming curves further up the road as seen in 4.5b. Figure 4.5c and 4.5d are examples of less accurate results which can occur when the ground is covered in grass.

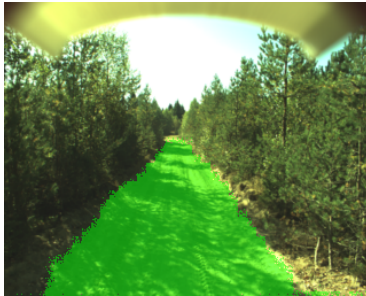
4.1.3 Path verification

A dataset of 270 samples was used to test the path verification technique. Of the 270 images, 225 passed the path verification meaning the path laid within the segmented area 83.33 % of the time. None of the predicted steering angles were more than 6-degrees off from their target value. The 45 samples that failed the test had a steering angle outside the segmented road.

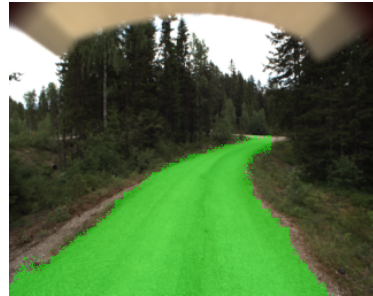
4.2 Discussion

4.2.1 Steering angle model

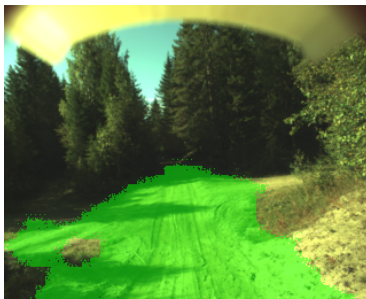
The results from the tests proved that in a dirt road environment, a multi-input end-to-end network using camera and LiDAR could outperform a single-input end-to-end network, whether it is using camera images or LiDAR data. This finding is comparable



(a) Shadow on road (AP: 93.6)



(b) Turn far up ahead (AP: 90.4)



(c) Wide road (AP: 79.4)



(d) Overgrown trail (AP: 79.1)

Figure 4.5: Samples showing less accurate segmentation

to what [3] found, who used a similar approach but on public roads. This indicates that multi-input networks handle the massive amount of data, and manages to exceed the prediction results from single-input networks, also on dirt roads. Both subnetworks extract information the other one does not have, and the last fully connected layers map these right to enhance the performance. LiDAR data also proved to perform slightly better than camera images. Neural networks are challenging to analyze, but an explanation could be the chaotic information in camera images compared to LiDAR data. The camera sensor picks up millions of details, while LiDAR has fewer pixels per meter and only captures the essential, the drivable area. On the contrary, the images used in this project only had 1-channel, and RGB camera images might give a different result.

When comparing the loss plots from the camera subnetwork, Figure 4.2, and LiDAR subnetwork, Figure 4.3, one can see a higher difference between the training and validation loss on the camera subnetwork, than on the LiDAR subnetwork. The LiDAR subnetwork has a more significant loss drop than the camera subnetwork. The reason for the different loss convergences between these two subnetworks can be explained by the different data input and the slight difference in architecture. As seen on Figure 4.4, the loss of the steering angle network stabilizes after about 10 epochs. This can indicate that the last fully connected layers have reached its global or local minima, given the input from the two subnetworks.

The goal of this project was not developing the best possible result compared to other papers, but comparing the results of this experiment indicates how well multi-input networks perform in contrast to single-input. The steering angle network presented here scored lower than other papers. Nvidia [1] did not present their RMSE result in their work, but from [9] it was found to be 0.1604. Camera and LiDAR alone scored respectively 0.2244 and 0.2236, while the fused steering angle network scored 0.1926. This shows a noticeable drop when combining sensors. Performance can possibly be improved to some degree with further tuning and more epochs, however, this was a difficult environment to train on. Imbalanced and unsymmetrical training data are also factors that have an unwanted effect on the testing. The steering angle model has mostly been exposed to straight forward driving. This is visualized in Figure 4.6 where one can see that the majority of the training set consist of steering angles less than 3 degrees left and right. This may mean that the model is overstimulated on forward driving, and it is predicting statistically more forward driving than there really is. A dataset with more turns would be advantageous. Unsymmetrical training data was also an issue [5] discovered to affect the results. The red line in Figure 4.6 indicates the mean of the training set which is slightly off set to the left (positive steering angles imply left turns). This will give a prediction mean slightly to the left of the center. This could have been avoided by flipping images of right turns horizontally.

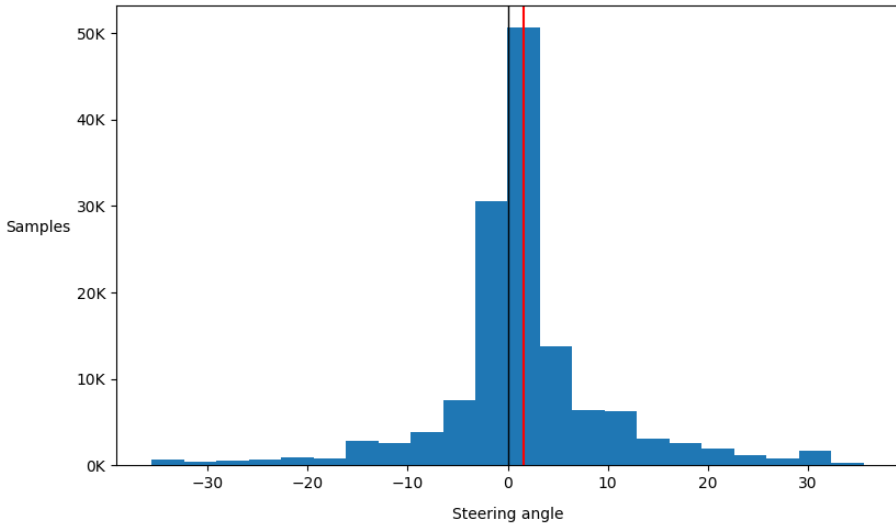


Figure 4.6: Visualization of steering angle distribution. Red line illustrates the mean.

In addition, if the multi-input network is trained correctly, it can withstand sensor failure where one sensor can do the work on behalf of both sensor [28]. This will give a more robust system in the form of hardware redundancy, along with better resistance to sensor noise like sun flair, rain, snow, and reflections.

4.2.2 Segmentation model

Segmentation of dirt road proved to be possible and accurate enough to be used for path verification. As seen in Figure 4.5c and 4.5d, the network is capable of accurately segmenting the drivable area, which then can be used for path verification. Accuracy at this level requires a dirt road without grass on, aside from that, the network segments most of the drivable dirt road from the environment. When the ground is covered in grass and leaves the model has problems segmenting correct, even if it is in the

normal drivable area. This may explain the patch of false negative in the center of Figure 4.5d, which can indicate that the network recognizes grass as something that is generally classified as non-road. Figure 4.5d illustrates another outcome where the road is wider than normal, and the model cannot find the expected road shoulder. A video demonstrating the segmentation network on a test set can be found in the link below¹. In the video, the FCN was given new images in a wet environment which it never was trained on, and then segments the dirt road from the environment.

The data used when training and testing the segmentation model were hand-picked. This was to get samples that would fit the expected location of the future UGV. By doing this, the randomness of the dataset dissipates. It is difficult to find suitable data and at the same time preserve the natural aspect of the dataset. Apart from that, the results proved to be acceptable for the purpose of this task. The segmentation model showed that it was capable of producing results acceptable for the path verification pipeline. In addition, it is difficult to define how accurate a segmentation has to be to be used by path verification.

4.2.3 Path verification

When analyzing the path verification technique, it indicated that it is possible to perform a path verification. This is mainly thanks to the segmentation network which for the most part was able to return segmentations of the full road without holes in it. However, in the test set of 270 samples, the target steering angles were unusually small. The biggest steering angle found in the dataset were 0.45 degrees off zero. So when running the test set through the path verification, the steering angles would appear almost just as vertical lines. Two verifications are visualized in Figure 4.7 showing how the steering angle would appear in all 270 cases. It is not sure why the steering angles from this test set were so small, but it could be an error in the sensor monitoring the steering angle, or an error in the script that collected them from the rosbag. The

¹Video source: <https://youtu.be/3p7P1DdpOqw>

overall result of the path verification was 83.33 % which would have been higher if the correct steering angle were applied. The relatively high score caused by notably more open spaces than regular narrow dirt roads with turns. The video mentioned in the last section is a visualization of the segmentation done on the 270 test images.



Figure 4.7: Two samples from the test set to visualize the path verification. (a) was a sample that passed the verification, while (b) did not. An error in the test set causes the all steering angles to be close to 0 degrees.

The approach used in this project for path verification is debatable, and the technique does have some drawbacks and weak spots. To ensure a correct perspective transformation (birds-eye view) the road ahead has to be a plane. This is rarely the case on dirt-roads. On public roads and especially highways, the level difference between the vehicle and a few meters ahead of the vehicle is minimal. Another negative impact of the performance of the path verification is the processing of inaccurate segmented images. When the steering angle is drawn on the image, it requires a segmented image of the full road. From road shoulder to road shoulder, and from the bumper and all the way up to the threshold. False positive predictions from the path verification can be hazardous. If the system falsely approves a path where it actually is not, the vehicle can potentially crash and harm people. The path verification does need further development to be used in real applications. For example, one can experiment with a curved path instead

of linear, or other completely different approaches using the segmented image and the predicted steering angle. Nevertheless, the thesis presents a proof-of-concept idea for path verification, which indicates possibilities for verification of paths using segmentation and steering angle.

4.3 Future work

Getting a car to drive autonomously, especially on overgrown dirt roads, is a complex task. Suggestions mentioned in the discussion will improve the performance of this system, but there are still pieces missing before this end-to-end approach is capable of driving in such an environment. This system is meant to be implemented in a UGV. To do this the software needs an interface to communicate the UGV used by FFI. Although the end-to-end approach handles all of the local path planning, there still has to be other sensors for detecting special incidents. A controller is also necessary to handle these incidents and enforcing actions parallel to end-to-end module. Sudden and unpredictable incidents an autonomous UGV can encounter are many. People and animals can all of the sudden enter the path ahead of the vehicle. Other vehicles may also cut off the UGV from the side. The UGV also needs an emergency sensor to verify a clear path ahead of the vehicle. There has been done research on reinforcement learning approaches for handling complex tasks like these [29]. A vehicle has to take decisions in a chaotic environment, and modern cars have several sensors which can help understand the situation. Deep reinforcement learning together with convolutional networks can help improve actions when unexpected events happens. Additionally, a route planner is also advantageous, with GPS and maps for longer route preparations. All these elements have to be processed in a controller to calculate the steering angle and activate other actions like braking. Most of these considerations are unavoidable when the UGV ultimately will travel fully autonomous.

4.4 Conclusion

The thesis has demonstrated that in a dirt road environment, multi-input end-to-end networks using camera and LiDAR could outperform single-input end-to-end networks using camera or LiDAR alone. This can help improve the local navigation of an off-road autonomous UGV. Using multiple sensors in an end-to-end fashion allows for redundancy where one sensor can take control if the other fails, along with improved resistance to sensor noise like sun flair, rain, snow, and reflections. Also, experiments on the path verification technique presented here indicate that road segmentation together with the predicted steering angle could possibly be used to verify the local path of an autonomous off-road UGV, in some form or another.

Appendix A

Media attachment

A zip-file with scripts was uploaded as an attachment with this thesis. This zip-file contains scripts used for generating data from the rosbags, building the neural networks, training them, and testing them.

A.1 Script setup

The scripts were all written for python 3.5. The main modules imported in `gen_data.py` were `openCV`, `ROS`, `rosbag`. The rest of the scripts use `TensorFlow 1.4` with `CUDA 8.0`. The datasets were not possible to include as an attachment because of its size.

A.2 Organization

```
./Appendix A
├── /Generate data
│   ├── gen_data.py
│   └── README.txt
├── /Segmentation network
│   ├── eval_segm.py
│   ├── helper.py
│   ├── main.py
│   ├── predict.py
│   ├── README.txt
│   └── test.py
├── /Steering angle network
│   ├── /deployment
│   │   ├── model_deploy.py
│   │   └── model_deploy_test.py
│   ├── /nets
│   │   ├── nets_factory.py
│   │   ├── vgg.py
│   │   └── vgg_test.py
│   ├── /preprocessing
│   │   ├── preprocessing_factory.py
│   │   └── vgg_preprocessing.py
│   ├── infer_fusion_net.py
│   ├── infer_subnetwork.py
│   ├── png2tfrecords.py
│   ├── train_fusion_net.py
│   └── train_subnetwork.py
└── /Verification experiment
    ├── README.txt
    └── verification.py
```

Appendix B

Steering angle prediction from segmented images

In the Udacity Self-driving Car program, one can learn how to build a pipeline of computer vision techniques to extract the lanes from an image and use this to calculate the curvature of the lane [27]. In their course, one assumes a level road ahead of the vehicle, which it mostly is when driving on public roads. A similar approach is presented in this experiment, where the recommended steering angle is approximated using the segmented image from Section 3.4. The segmented binary image is first smoothed, but in contrast to the path verification pipeline in Section 3.5, a canny edge detection is then used to find the road shoulder (see Figure B.2b) [30]. In the same way, as Udacity uses the lane markings for lane curve approximation, this pipeline uses the edge of the segmented image. Next, the image is perspectively transformed to a birds-eye perspective. Finally, a polynomial line is fitted to the edge of each of the two road shoulders (Figure 3.8d). The curvature of the polynomial lines can then be used to calculate the lane curvature. With the lane curvature, one can use the axle distance to calculate the recommended steering angle for the UGV. The geometrical

principle is illustrated in Figure B.1. Instant Centre of Rotation (ICR) is the center of which the vehicle rotates around. L is the length between the front and back axle. α is the steering angle. R is the radius of curvature. Typically on a four-wheel vehicle as the UGV, the inner front wheel turns sharper than the outer one when turning, this is called Ackermann steering geometry. The steering angle stored in the dataset is the average between the two wheels. For the sake of demonstrations, there was no need for an accurate steering model and the curvature of the turn was calculated by the simplified model used here. The angled wheel in this figure represents the average position the two front wheels on the UGV.

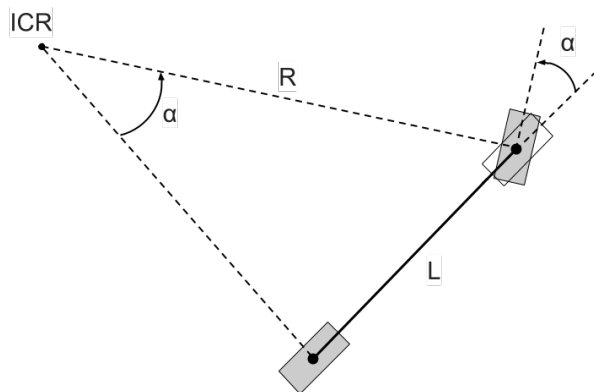


Figure B.1: Illustration of the model used to calculate the radius of curvature to the vehicle when it is turning.

Through experiments like the one illustrated in figure B.2, it proved possible to extract lane curvature from the segmented images. With the lane curvature, one can find the recommended steering angle to the UGV. The segmented image used in Figure B.2 has a AP score of 90.9 % with a straight edge following the road shoulder. This is advantageous when the lane curvature pipeline is fitting the polynomial lines. Throughout the experiments, one could observe which inputs the pipeline would perform better on. These included segmented images of clearly defined dirt roads with high AP scores. On the contrary, when poorly segmented images were tested, the

pipelines approximated the lane curvature inaccurately. These images were mostly segmentations of grassy trails and dirt roads wider than average. The performance was only qualitatively evaluated using the current steering angle and the differences between the left and right lane curvature. Again, this was purely an experimental test to demonstrate a possible steering angle prediction technique.

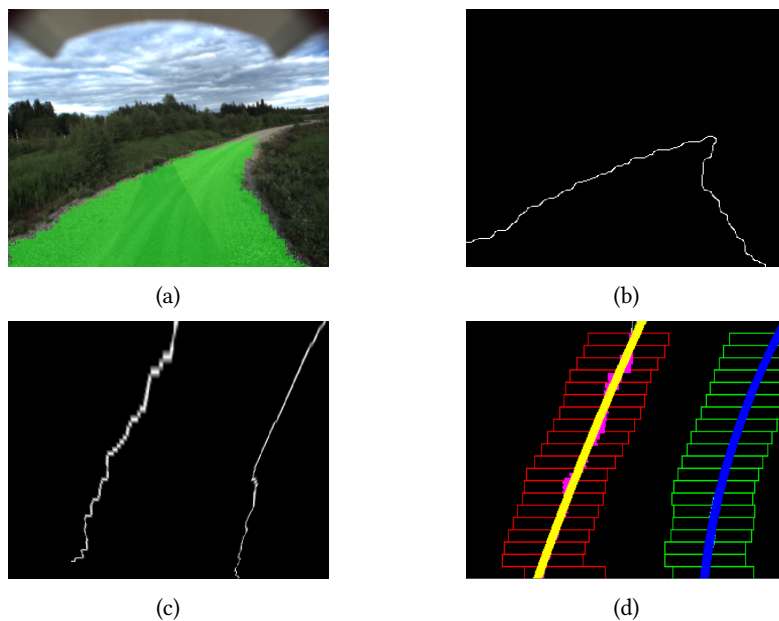


Figure B.2: (a) Original image with predicted segmentation overlay. (b) Binary image of predicted segmented area, smoothed and Canny edge detection. (c) Birds-eye transformation. (d) Lane curvature approximation.

To sum up, it lays a potential for accurately approximate steering angles from segmented roads, and in this case even dirt roads. Throughout this experiment, it proved possible to extract lane curvature from the segmented images, given a typical dirt road where the soil of the drivable area differs from non-drivable areas. As a side note, one could debate that this method of steering angle prediction, in general, potentially could

be more efficient than the end-to-end approach. When predicting with end-to-end approaches, the neural network has to understand the input, map it, and output a specific continuous value. The segmentation network, which is the only critical part of the pipeline, has only one simple task.

References

- [1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Praseon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.
- [2] Hotz and Comma.ai. research, 2016. [Online; Accessed 21. Aug. 2016] <https://github.com/commaai/research>.
- [3] Yiping Chen, Jingkang Wang, Jonathan Li, Cewu Lu, Zhipeng Luo, Han Xue, and Cheng Wang. Lidar-video driving dataset: Learning driving policies effectively. In *IEEE Conference on Computer Vision and Pattern Recognition*, 03 2018.
- [4] Hotz and Comma.ai. AI by the bay, 2017. [Online; Accessed 30. Aug. 2017] <https://www.youtube.com/watch?v=IxuU5L2MEII>.
- [5] Ø. Grimnes. End-to-end steering angle prediction and object detection using convolutional neural networks. Master's thesis, NTNU, Norwegian University of Science and Technology, 2017.
- [6] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.
- [7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

- [8] Zhengyuan Yang, Yixuan Zhang, Jerry Yu, Junjie Cai, and Jiebo Luo. End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perception. *CoRR*, abs/1801.06734, 2018.
- [9] Lu Chi and Yadong Mu. Deep steering: Learning end-to-end driving model from spatial and temporal visual cues. *CoRR*, abs/1708.03798, 2017.
- [10] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.
- [11] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. *CoRR*, abs/1611.07759, 2016.
- [12] M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, and R. Urtasun. MultiNet: Real-time Joint Semantic Reasoning for Autonomous Driving. *ArXiv e-prints*, December 2016.
- [13] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.
- [14] Tim Dettmers. Deep learning in a nutshell history and training, 2015. [Online; Accessed 18. Sep. 2017]
<https://goo.gl/59rgZg>.
- [15] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 5 2015.
- [16] Goodfellow, Bengio, and Courville. *Deep Learning*. MIT Press, 2016.
- [17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.
- [18] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus Robert Muller. *Efficient BackProp*, pages 9–50. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

- [19] Xavier Glorot, Antoine Bordes, and Y Bengio. Deep sparse rectifier neural networks. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS) 2011*, volume 15, pages 315–323, 01 2011.
- [20] Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines vinod nair, 2010.
- [21] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.
- [22] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, Jun 2010.
- [23] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [24] Bruce E. Bayer. Color imaging array. Pat.nr. US3971065A, 1975.
- [25] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. "grabcut": Interactive foreground extraction using iterated graph cuts. *ACM Trans. Graph.*, 23(3):309–314, August 2004.
- [26] Hassan Abu Alhaja, Siva Karthik Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. Augmented reality meets deep learning for car instance segmentation in urban scenes. In *British Machine Vision Conference (BMVC)*, 2017.

- [27] Udacity. Self-driving car nanodegree, 2016. [Online; Accessed 12. Oct. 2017] <https://goo.gl/J6KSjk>.
- [28] N. Patel, A. Choromanska, P. Krishnamurthy, and F. Khorrami. Sensor modality fusion with cnns for ugv autonomous driving in indoor environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1531–1536, Sept 2017.
- [29] Changkun Ye, Huimin Ma, Xiaoqin Zhang, Kai Zhang, and Shaodi You. Survival-oriented reinforcement learning model: An efficient and robust deep reinforcement learning algorithm for autonomous driving problem. In Yao Zhao, Xiangwei Kong, and David Taubman, editors, *Image and Graphics*, pages 417–429, Cham, 2017. Springer International Publishing.
- [30] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, Nov 1986.