Master of Science Thesis in Electrical Engineering
Department of Electrical Engineering, Linköping University, 2018

State Estimation for Truck and Trailer Systems using Deep Learning

Daniel Arnström

LINKÖPING UNIVERSITY

Master of Science Thesis in Electrical Engineering

State Estimation for Truck and Trailer Systems using Deep Learning

Daniel Arnström

LiTH-ISY-EX--18/5150--SE

Supervisor: **Oskar Ljungqvist**
ISY, Linköpings universitet

Examiner: **Daniel Axehill**
ISY, Linköpings universitet

Division of Automatic Control
Department of Electrical Engineering
Linköping University
SE-581 83 Linköping, Sweden

Abstract

High precision control of a truck and trailer system requires accurate and robust state estimation of the system. This thesis work explores the possibility of estimating the states with high accuracy from sensors solely mounted on the truck. The sensors used are a LIDAR sensor, a rear-view camera and a RTK-GNSS receiver.

Information about the angles between the truck and the trailer are extracted from LIDAR scans and camera images through deep learning and through model-based approaches. The estimates are fused together with a model of the dynamics of the system in an Extended Kalman Filter to obtain high precision state estimates. Training data for the deep learning approaches and data to evaluate and compare these methods with the model-based approaches are collected in a simulation environment established in Gazebo.

The deep learning approaches are shown to give decent angle estimations but the model-based approaches are shown to result in more robust and accurate estimates. The flexibility of the deep learning approach to learn any model given sufficient training data has been highlighted and it is shown that a deep learning approach can be viable if the trailer has an irregular shape and a large amount of data is available.

It is also shown that biases in measured lengths of the system can be remedied by estimating the biases online in the filter and this improves the state estimates.

Acknowledgments

First of all, I want to thank my examiner Daniel Axehill and my supervisor Oskar Ljungqvist for their guidance, help and support throughout this thesis work. Their ideas and feedback have been invaluable to me.

I also want to thank my family for their unconditional support and encouragement in all of my endeavors.

Linköping, June 2018
Daniel Arnström

Contents

Notation

Quantities

Quantity	Meaning
(x_1, y_1)	Global position of the rear axle of the truck
(x_3, y_3)	Global position of the rear axle of the trailer
(L_x, L_y)	Position of the middle of the trailer short side relative the truck and dolly hitch
θ_1	Global orientation of the truck
θ_3	Global orientation of the trailer
β_2	Angle between the truck and the dolly
β_3	Angle between the dolly and the trailer
ϕ	Angle between the truck and the trailer
α	Steering wheel angle
v	Speed of the truck
L_1	Length of the truck
L_2	Length of the dolly
L_3	Length of the trailer
L_k	Front overhang of the trailer
M_1	Truck off-hitch
(u_i, v_i)	Pixel position of marker i
d_i	Distance from the middle of the trailer short side and marker i

Acronyms

Acronym	Meaning
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DNN	Deep Neural Network
EKF	Extended Kalman Filter
GNSS	Global Navigation Satellite System
RANSAC	Random Sample Consensus
SGD(M)	Stochastic Gradient Descent (with Momentum)

1 Introduction

State estimation of a truck and trailer system using sensors mounted on the truck is the main concern in this thesis. This chapter gives the background of the thesis, states the problem formulations, relates the thesis to previous works and gives an outline of the thesis.

1.1 Background

In order to control a truck and trailer system automatically with high precision, accurate and robust state estimation of the system is needed. These include the global position and orientation of the system and the angles between the truck and the trailer. One approach to acquire estimates of the angles would be to measure them directly by mounting sensors at the connections of the truck and dolly and the dolly and trailer. A drawback of this approach is that compatibility issues can arise when different dollies and trailers are used, e.g., providers with divergent communication protocols or obsolete trailers. An alternative is to only use sensors that are mounted on the truck. Examples of sensors that are viable for this situation are LIDAR sensors and cameras. These sensors generate data containing a lot of information and the process of extracting the relevant information from these sources often becomes complex if handcrafted methods are employed.

A recent approach to extract information from complex data is to use machine learning techniques, such as deep learning, which circumvents the need for handcrafted feature extractions and instead learns to extract the relevant information in an end-to-end fashion. To use deep learning, a large amount of data has to be

collected and a computer with sufficient computation power is needed. Some factors which have led to the rise of these approaches are the increased availability of these resources.

The estimates of the angles between the truck and trailer can be complemented with measurements from a GNSS receiver mounted on the truck to estimate the global position and orientation of the trailer.

1.2 Problem Formulation

The main purpose of this thesis is to investigate how one can measure the states of a truck and trailer system with high accuracy through sensors solely mounted on the truck. Much of the focus is to investigate if it is possible to obtain high precision angle estimates from LIDAR measurements and camera images using deep learning, how this approach compares with model-based approaches and in which cases one is preferred over the other.

Another question is how the geometry of the system affects the estimates and if errors in the geometric modeling can be remedied using the sensors mounted on the truck.

1.3 Related work

The truck and trailer system treated in this thesis has been investigated earlier in [18] [17] [1] [2]. Of special interest for this thesis is [18] which studies how to estimate the angles between the truck and the trailer, using a LIDAR sensor and a GNSS receiver through a model-based approach. This approach has been used in this thesis as a baseline for state estimates using a LIDAR sensor.

The applications of deep learning on point clouds is often in the context of segmentation and applied on 3-dimensional point clouds. More relevant for this thesis is the use of deep learning on 2-dimensional point clouds for object detection which has been explored in [16] and [3] but in both cases the point cloud is first converted into an image instead of directly feeding the point cloud to a neural network.

In [20] a Bayesian approach is proposed to find the position and orientation of a vehicle from a point cloud using a particle filter. The conditional probability of obtaining each particular point in a point cloud, given the state of the system, is modeled and a Rao–Blackwellized particle filter is used to estimate the vehicle state.

Deep learning is used in [13] to estimate the position and orientation of a camera from camera images. A neural network trained for a classification task is used as a base for training a network for the regression task and this is the same approach

taken in this thesis, where a pretrained network trained for classification is used as a base for training a network performing regression using images.

Model-based approaches to estimate the articulation angle between a truck and a trailer using a rear-view camera has been presented in [4][5][8]. The angle is estimated in [4] by exploiting symmetries of the drawbar in images from a rear-view fisheye camera. In [5] a picture bank of expected appearances of the trailer for different angles is generated and the current image is compared with this bank to determine the angle. Markers located on the trailer are used in [8] and detections of the markers in a camera image are used to estimate the articulation angle, this approach is similar to the model based-approach used in this thesis.

1.4 Outline

The outline of the thesis is

- **Chapter 2 - System Overview and Modeling** gives an overview of the autonomous truck and trailer system of interest in this thesis. The chapter also presents models for the dynamics of the system and how the measurements are related to the system states and geometry.
- **Chapter 3 - Neural Networks** Introduces neural networks and how they are trained.
- **Chapter 4 - State Estimation** presents the techniques used for estimating the states.
- **Chapter 5 - Data Collection and Training** presents how data was collected and how the techniques introduced in chapter 3 were used to obtain state estimates.
- **Chapter 6 - Results** presents results for the estimation methods and discusses their implication.
- **Chapter 7 - Conclusions** presents the conclusions drawn from the results.

2

System Overview and Modeling

This chapter gives an overview of the truck and trailer system and presents a state space model for the system.

2.1 System Overview

There are three main modules that are needed to control an autonomous vehicle, a motion planner, a controller and an observer. An overview of the modules and their connections are shown in Figure 2.1.

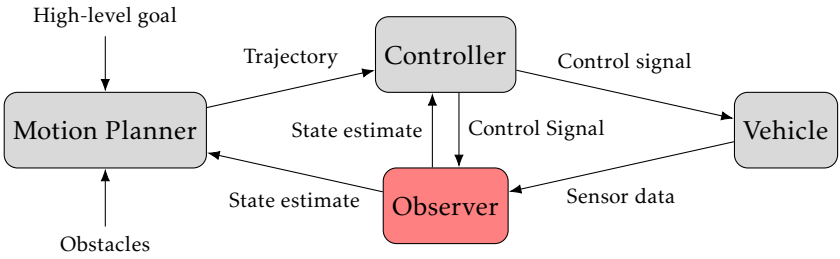


Figure 2.1: *Overlying system architecture for an autonomous truck and trailer system. The subsystem of interest in this thesis is marked in red.*

The motion planner generates a reference trajectory given some high-level goal, e.g to position the trailer according to some waypoints, and the current state of

the system. The generated trajectory is such that it is kinematically possible for the system to travel along it and unwanted collision with obstacles are avoided.

The controller produces control signals, such as throttle and steering wheel angle, to control the vehicle such that it follows the desired trajectory given by the motion planner.

Both the motion planner and the controller use the current state of the system to generate reference trajectories and control signals, respectively. The states of the vehicle are seldom available directly and need to estimated from sensor data. It is the observer that creates these estimates by using information from sensors and control signals. This thesis focuses on the observer and how to achieve high precision estimates given the sensor data from a LIDAR sensor, camera and GNSS receiver for a truck with a dolly-steered trailer.

2.2 Platform

The system consists of a truck, a dolly and a trailer shown in Figure 2.2. The truck is connected to the dolly through a kingpin hitching and the dolly is hitched to the trailer.

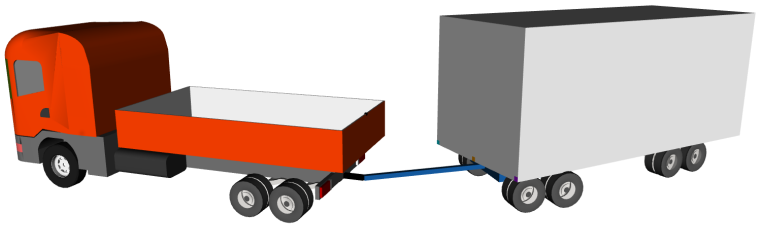


Figure 2.2: *The truck-trailer system under investigation in this thesis.*

All the sensors of the system are mounted on the truck and consist of a LIDAR sensor, a camera and a GNSS receiver.

The LIDAR is mounted on the back of the truck faced backward. 580 evenly spaced rays are sent out from the LIDAR with a field of view of 144° resulting in a point cloud from hits on the trailer. In this thesis, the data are assumed to be preprocessed such that hits on other distant objects have been removed and the hits on the trailer are given as a set of points in cartesian coordinates.

The camera is also mounted on the back of the truck facing the trailer with a field of view of 102°. The specifications of the camera were based on Scanias rear-view cameras from the FAMOS generation [21].

The GNSS receiver provides the position and orientation of the truck in a global coordinate system. A real-time kinematic (RTK) GNSS receiver is used, which improves the precision of the measurements compared with traditional GNSS receivers and thus mitigating drift in estimates.

2.3 Vehicle Model

Figure 2.3 depicts a general N-trailer which consists of N links and $N + 1$ axles. For a general N-trailer, a link does not need to be hitched directly to the preceding axle. The distance between axle j and the hitching point to the previous link is denoted L_j and the distance between axle j and the hitching to the following link is denoted M_j. The global orientation of axle j is denoted θ_j and its velocity is denoted v_j.

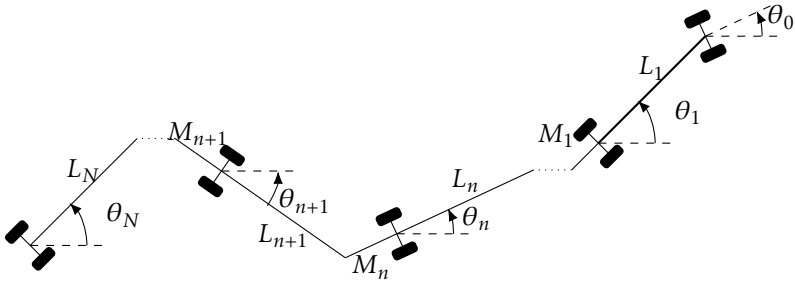


Figure 2.3: *A general N-trailer system.*

A recursive formula for $\dot{\theta}_j$ and $\dot{v}_j$ containing the systems geometry and the global orientation and velocity for the preceding axle has been derived in [1] based on kinematic constraints. The recursive formulas are

$$\dot{\theta}_n = \frac{v_n \tan(\theta_{n-1} - \theta_n)}{L_n} - \frac{M_{n-1}\dot{\theta}_{n-1}}{L_n \cos(\theta_{n-1} - \theta_n)} \tag{2.1a}$$

$$\dot{v}_n = v_{n-1}\cos(\theta_{n-1} - \theta_n) + M_{n-1}\sin(\theta_{n-1} - \theta_n)\dot{\theta}_{n-1} \tag{2.1b}$$

The truck and trailer system of interest in this thesis can be model as a special case when $N = 3$ and $M_2 = 0$. A bird's-eye view of this case is shown in Figure 2.4.

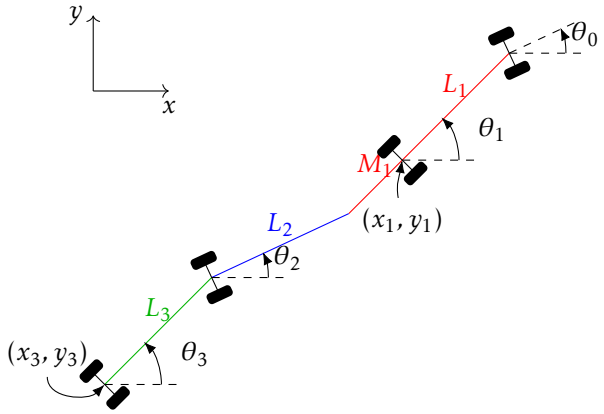


Figure 2.4: *Bird's-eye view of the truck and trailer system where the truck is marked in red, the dolly is marked in blue and the trailer is marked in green.*

Let the angle between the truck and the dolly be denoted β_2, the angle between the dolly and the trailer be denoted β_3 and the steering angle be denoted α. The relations between these angles and the global orientations are

$$\alpha = \theta_0 - \theta_1 \tag{2.2a}$$
$$\beta_2 = \theta_1 - \theta_2 \tag{2.2b}$$
$$\beta_3 = \theta_2 - \theta_3 \tag{2.2c}$$

By combining (2.2) and (2.1) one can derive the following vehicle model for the truck and trailer system, (the entire derivation is carried out in [17])

$$\dot{x}_3 = v \cos\beta_3 \cos\beta_2 \left(1 + \frac{M_1}{L_1} \tan\beta_2 \tan\alpha\right) \cos\theta_3 \tag{2.3a}$$
$$\dot{y}_3 = v \cos\beta_3 \cos\beta_2 \left(1 + \frac{M_1}{L_1} \tan\beta_2 \tan\alpha\right) \sin\theta_3 \tag{2.3b}$$
$$\dot{\theta}_3 = \frac{v \sin\beta_3 \cos\beta_2}{L_3} \left(1 + \frac{M_1}{L_1} \tan\beta_2 \tan\alpha\right) \tag{2.3c}$$
$$\dot{\beta}_3 = v \cos\beta_2 \left(\frac{1}{L_2}\left(\tan\beta_2 - \frac{M_1}{L_1}\tan\alpha\right) - \frac{\sin\beta_3}{L_3}\left(1 + \frac{M_1}{L_1}\tan\beta_2\tan\alpha\right)\right) \tag{2.3d}$$
$$\dot{\beta}_2 = \frac{v}{L_1 L_2}\left(L_2 \tan\alpha - L_1 \sin\beta_2 + M_1 \cos\beta_2 \tan\alpha\right) \tag{2.3e}$$

where v is the speed of the truck and (x_3, y_3) is the global position of the trailer.

This leads to a state space description of the system with the states $(x_3, y_3, \theta_3, \beta_3, \beta_2)$ and the control signals (v, α).

Important to note is that the model assumes sliding without slipping and that the vehicle movement is planar which makes the model imperfect in situations when the vehicle performs rapid maneuvers and drives on uneven surfaces.

2.4 Measurement equations

Measurement equations relate the measured quantities to the state of the system. In this case, the states are $(x_3, y_3, \theta_3, \beta_3, \beta_2)$, where x_3, y_3 and θ_3 describe the global position and orientation of the system and β_3 and β_2 describe the internal state of the system. In this thesis, the main focus is to estimate the internal states.

In the case where β_3 and β_2 are estimated end-to-end from a neural network the measurement simply becomes a direct measurement of the angles and thus the measurement model is

$$\hat{\beta}_2 = \beta_2 \tag{2.4a}$$
$$\hat{\beta}_3 = \beta_3 \tag{2.4b}$$

where $\hat{\beta}_2$ and $\hat{\beta}_3$ are the outputs from the neural network. A drawback of estimating the angles end-to-end is that geometric information about the system is lost since this information is encapsulated inside the neural network. This is a drawback when one wants to estimate possible biases in this geometry as will be further discussed in future chapters.

2.4.1 LIDAR

An alternative measurement model to estimate β_2 and β_3 which retains geometric information about the system is to measure the angle between the truck and the trailer ϕ and the middle point of the trailer's short side (L_x, L_y). L_x and L_y are defined in a local coordinate system with the origin at the truck and dolly hitch and the x-axis aligned with the truck. Figure 2.5 gives an overview of the geometry and the following relationships can be derived

$$\phi = \beta_2 + \beta_3 \tag{2.5a}$$
$$L_x = -L_2 \cos(\beta_2) + L_k \cos(\beta_2 + \beta_3) \tag{2.5b}$$
$$L_y = L_2 \sin(\beta_2) - L_k \sin(\beta_2 + \beta_3) \tag{2.5c}$$

where L_k is the distance from the middle of the dolly axle to the midpoint of the trailer's short side.

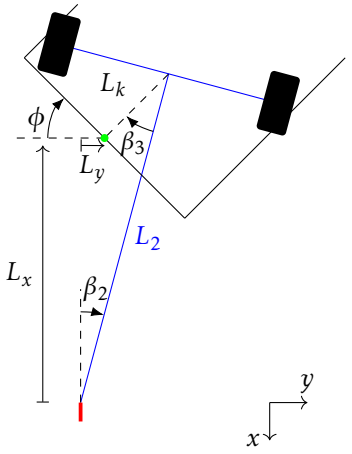


Figure 2.5: *Bird's-eye view of the connection between truck and dolly and the connection between dolly and trailer. The green dot marks the middle point of the trailer's short side at coordinates (L_x, L_y). Blue represents parts of the dolly and red represents parts of the truck.*

ϕ and (L_x, L_y) can be extracted from a LIDAR scan by fitting straight lines to the point cloud. There are two main cases, either the LIDAR hits only the short side or the LIDAR hits both the short side and one of the long sides. ϕ are in both cases related to the slope of the line representing the short side.

The middle point (L_x, L_y) can in the first case be calculated by fitting a line to the point cloud and taking the two outermost points belonging to the line as edges for the line segment representing the short side of the trailer. The coordinate for the middle point is then approximated as the mean of the coordinates of these points. The estimated middle point can be projected onto the fitted line to improve the accuracy of the estimate. The course of action is summarized in Figure 2.6.

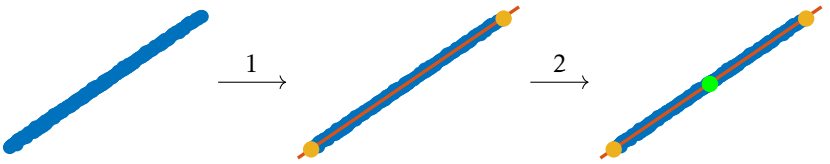


Figure 2.6: *An overview of how the middle point of the trailer's short side can be extracted when one side is in view of the LIDAR. In 1 a line is fitted to the point cloud and the two outermost points belonging to the line are taken as edge points. In 2 the middle point is taken as the mean of the edge points.*

In the second case, the two most dominating lines are extracted from the point cloud. The coordinates of a corner of the trailer can then be decided as the intersection of the two lines. Using the slope of the lines and the fact that the width of the trailer is known one can derive an estimate of the middle point coordinates. The course of action is summarized in Figure 2.7.

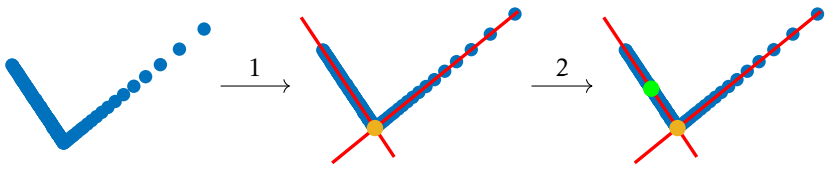


Figure 2.7: *An overview of how the middle point of the trailer's short side can be extracted when two sides are in view of the LIDAR. In 1 the two most dominating lines in the point cloud are extracted and a corner is decided as the intersection of the two lines. In 2 the slope of the lines and the width of the trailer are used to decide the middle point.*

How lines and points on these lines are extracted from the point cloud is described in Section 4.2.1.

2.4.2 Camera

A model-based approach to gain information about β_2 and β_3 using a camera is to detect markers in the picture with known placement on the trailer. With this method, the measurements are the pixel positions (u, v) in the image where the markers are detected and these pixel positions can be related to the markers world coordinates by a projection model. The world coordinates are in turn related to the angles β_2 and β_3, the geometry of the system and the placement of the markers on the trailer. The objective is to find an equation that relates the markers pixel position in the image (u, v) to the states (β_2, β_3). An overview of the transformations between (β_2, β_3) and (u, v) are given in Figure 2.8.

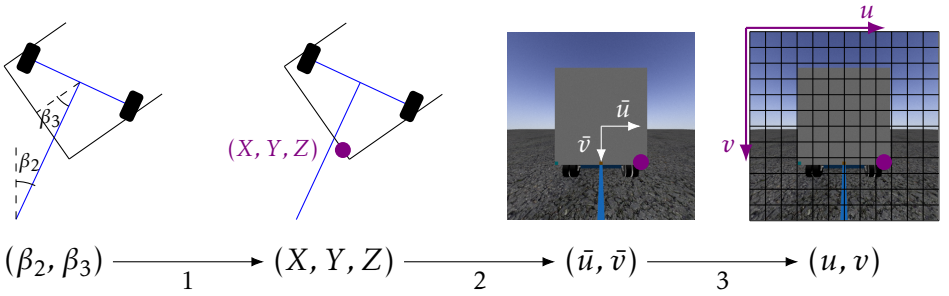


Figure 2.8: *1 given the states* (β_2, β_3) *one can find the coordinates* (X, Y, Z) *of a marker on the trailer in a cartesian coordinates system. 2 The marker on the trailer is projected onto the image plane. 3 the image plane is discretized and pixel* $(0, 0)$ *is in the upper left corner of the image.*

Marker position expressed in terms of the states

If the camera is positioned at the truck and dolly hitch and a marker is placed at a distance d from the middle point of the trailer short side, which is shown in Figure 2.9, one can use trigonometry to describe X and Z in terms of β_2, β_3 and the geometry of the trailer in the following way

$$X = L_2 \sin(\beta_2) - L_k \sin(\beta_2 + \beta_3) + d \cos(\beta_2 + \beta_3) \tag{2.6a}$$
$$Z = L_2 \cos(\beta_2) - L_k \cos(\beta_2 + \beta_3) - d \sin(\beta_2 + \beta_3) \tag{2.6b}$$

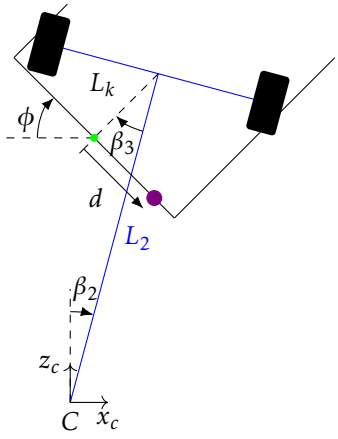


Figure 2.9: *Placement of a marker on the trailer at coordinates* (X, Y, Z).

Note that a translation and rotation needs to be applied to (X, Y, Z) to align the z_c-axis in the direction of the camera if the camera is not mounted at the truck and dolly hitch faced backward.

Worth noting is that the expression for Y has been omitted. The reason for this is to avoid introducing another parameter to the model since Y depends on the markers height relative to the camera. An additional measurement equation for the marker can be obtained by using (2.7b) if this height is known. Another observation is that by only focusing on X and Z one can detect lines, such as the edges of the trailer, instead of just points.

Projection model

The camera is modeled as a pinhole camera which treats the aperture as a point. The image coordinates $(\bar{u}, \bar{v})$ of a point with world coordinates (X, Y, Z) is obtained as the intersection between a straight line connecting the camera and the point (X, Y, Z) and the image plane $z = f$. The projection from world- to image coordinates are shown in Figure 2.10 which also shows how the world coordinate system is defined.

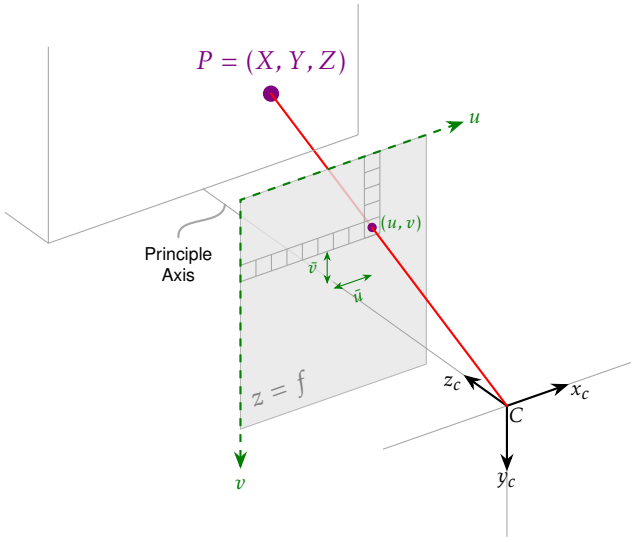


Figure 2.10: *Projections of a point P on the trailer to the image plane when using a pinhole camera model. C is the origin of the world coordinate system and is defined at the position of the camera*

By using the equilateral triangles in Figure 2.10 one can derive the following

relationships between the image coordinates and the world coordinates

$$\bar{u} = f\frac{X}{Z} \tag{2.7a}$$

$$\bar{v} = f\frac{Y}{Z} \tag{2.7b}$$

If high precision is to be achieved in practice, radial and tangential lens distortions need also to be taken into account. However, only the case of an ideal pinhole camera is handled in this thesis.

$(\bar{u}, \bar{v})$ can be related to the pixel coordinates (u, v) if the image dimensions and resolution is known through

$$\begin{aligned} \bar{u} &= s\left(u - \frac{W}{2}\right) \\ \bar{v} &= s\left(v - \frac{H}{2}\right) \end{aligned} \tag{2.8}$$

where H is the pixel height of the image, W is the width of the image and s is a scale factor which is dependent on the resolution.

Measurement model

Combining (2.7a),(2.8) and (2.6) results in the following measurement equation

$$u = \frac{f}{s}\frac{L_2\sin(\beta_2) - L_k\sin(\beta_2+\beta_3) + d\cos(\beta_2+\beta_3)}{L_2\cos(\beta_2) - L_k\cos(\beta_2+\beta_3) - d\sin(\beta_2+\beta_3)} + \frac{W}{2} \tag{2.9}$$

Since the focus in this thesis is on the state estimation of the truck and trailer system rather than image detection, a very simple method for detecting the pixel position (u, v) has been used. The method thresholds color content to locate markers in the image. To avoid false detections in practice, a more sophisticated method needs to be used.

2.4.3 GNSS

A GNSS receiver mounted on the truck provides measurements of the position of the rear axle of the truck (x_1, y_1) and the global orientation θ_1 of the truck. Using standard geometry in Figure 2.4 the following measurement equations can be established:

$$x_1 = x_3 + M_1\cos(\theta_3+\beta_2+\beta_3) + L_2\cos(\theta_3+\beta_3) + L_3\cos(\theta_3) \tag{2.10a}$$

$$y_1 = y_3 + M_1\sin(\theta_3+\beta_2+\beta_3) + L_2\sin(\theta_3+\beta_3) + L_3\sin(\theta_3) \tag{2.10b}$$

$$\theta_1 = \theta_3 + \beta_2 + \beta_3 \tag{2.10c}$$

2.5 State extension

The performance of the observer and the controller can be reduced if there are errors in the geometric parameters of the system. It is shown in [17] that the trajectory following is heavily impaired when a bias in L_3 is present and in [18] biases in L_2 and L_k are shown to induce biases in the estimates of β_2 and β_3 when using LIDAR measurements.

To account for biases in the geometry of the system one can augment the states in the state space model with the lengths L_2, L_3 and L_k. This allows the modeled geometry to be adjusted based on the measurements from the sensors and the dynamics of the system.

2.6 Observability

The trailer position is not unique if L_2, L_k, β_2 and β_3 are allowed to vary at the same time. This means that different combinations of L_2, L_k, β_2 and β_3 can produce identical point clouds, making it impossible to train a neural network end-to-end which is invariant of L_2 and L_k. This can be seen in (2.5) where the position of the trailer is constrained by three equation and if L_2,L_k,β_2 and β_3 are allowed to vary the degree of freedom is four, resulting in an underdetermined system of equations that defines the trailer's position. This also means that the dynamics of the system need to be used to be able to estimate L_2, L_k, β_2 and β_3 at the same time.

There is also a problem with using the GNSS measurements described by (4.14) when L_2 and L_3 are allowed to vary since one cannot distinguish a change in x_3 and y_3 from a change in L_2 and L_3 only based on these measurements. This may lead L_2 and especially L_3 to drift, which has been experienced during tests, resulting in the state estimates to diverge. Thus the GNSS measurements are to be avoided when L_2 and L_3 are seen as states that can vary over time.

3
Neural Networks

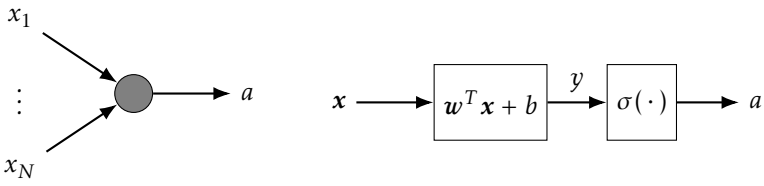
An artificial neural network (ANN) is used to approximate a mapping $f : x \to y$. It has been shown that ANNs can approximate any measurable function with arbitrary accuracy [11] which makes them "universal approximators". The mapping is learned by using a set of input-output pairs (x_t, y_t) that are seen as ground truth, called training data, and parameters of the network are adjusted to make $f(x_t) \approx y_t$ for all these pairs. This way of learning, where example input-output pairs are used, is called supervised learning, in contrast with unsupervised learning where the data is unlabeled.

3.1 Feed-forward neural networks

The atomic entity of a neural network is a neuron. A neuron consists of an affine transformation followed by a nonlinearity, this nonlinearity is often called an activation function and is denoted $\sigma(\cdot)$. The relationship between the inputs $(x_1, ..., x_N)$ and output a is

$$a = \sigma\Big(\sum_{i=1}^{N} w_i x_i + b\Big) \tag{3.1}$$

where $\{w_i\}_{i=1}^{N}$ is called the weights of the neuron and b is called the bias of the neuron, these are the adjustable parameters in a neural network. The output of a neuron is often called the activation of the neuron. Two ways of representing a neuron visually are shown in Figure 3.1.



(a) *Neuron* ***(b)*** *Neuron block diagram*

Figure 3.1: *Visual representations of a neuron.* $\boldsymbol{x}^T = (x_1, ..., x_N)$ *and* $\boldsymbol{w}^T = (w_1, ..., w_N)$.

The nonlinear activation function is important since it allows neural networks to approximate nonlinear mappings. There are several activation functions that are being used in neural networks but one of the most prevalent, and the one that has been used throughout this thesis work is a rectified linear unit (ReLU). A ReLU simply saturates all negative values to zero and leaves positive values unchanged. Thus the ReLU results in the activation function

$$\sigma(x) = \max(0, x) \tag{3.2}$$

An ANN is constituted by layers of stacked neurons as shown in Figure 3.2. The layers are often split into three categories, the input, output and hidden layers. An ANN with more than one hidden layer is called a deep neural network (DNN). DNNs have proven capable to describe very complex mappings which have made it possible to directly learn complicated input-output relationships without the need to employ ad-hoc feature extraction methods on the input before employing machine learning techniques.

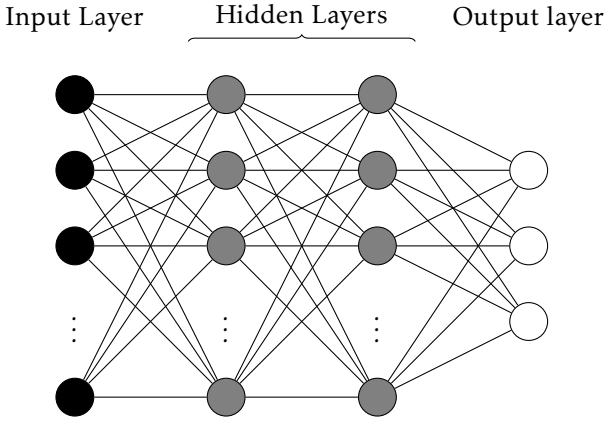


Figure 3.2: *Example of an ANN with two hidden layers and three output neurons.*

The ANN shown in Figure 3.2 is the most fundamental class of ANNs where the input to a neuron is the activations of all neurons in the previous layer. This class is called a fully connected ANN or a dense ANN.

3.2 Convolutional neural networks

Convolutional neural networks (CNN) are a class of ANNs that has found lots of success in image recognition, primarily for classification tasks. In a CNN spatial correlation in data are exploited to reduce the number of tunable weights and biases needed. In other words, data points that are close to each other are assumed to be correlated in some way. Examples of data with this property are images since values of nearby pixels are correlated. The components in a CNN that exploits these spatial correlations are called convolutional layers which fundamentally consists of filters sliding over the input data.

While CNNs have found some application on data in the form of text and audio, they are primarily used on images. This is also the case in this thesis and hence most of the descriptions that follow are geared for CNNs used on image data.

A hierarchical view of CNNs is often presented to motivate why they are so effective. The idea is to stack multiple convolutional layers after each other to extract more and more intricate features. The first convolutional layer trains filters that extract high-level features from images such as edges, corners and colors. Preceding convolutional layers use these high-level features to extract increasingly complex features and the final layers, often fully connected layers, combines complex features to perform the specific task that the CNN should be trained for.

3.2.1 Convolutional layer

In contrast to fully connected layers, convolutional layers arrange neurons in three-dimensional structures and thus both the input and output to a convolutional layer are seen as volumes. The output volume is constituted of activations from neurons arrange in two dimensions, forming so-called feature maps. Each feature map is produced by a filter acting on the input volume and these feature maps are stacked to produce the output volume. Hence the depth of the output volume is decided by the number of filters used in the convolutional layer. An overview of a convolutional layer is shown in Figure 3.3.

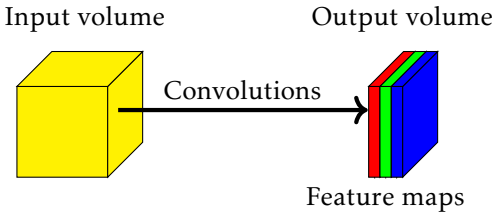


Figure 3.3: *Overview of a convolutional layer. Each filter produces a two-dimensional feature map through convolutions with the input volume. The output volume is produced by stacking the feature maps of all filters.*

The activation of a neuron in a feature map is produced by an affine transformation of the input followed by an activation function, just as in a dense ANN. What differs from an ANN is that the input only consists of the activations of neurons in a specific region in the input volume, not all activations as for ANNs. The region of input neurons which an output neuron uses is called the receptive field of the output neuron. The size of the receptive field is decided by the filter size and how far away the receptive fields are for neighboring output neurons are decided by the filter stride. An example of a 2x2xd filter with stride 1 acting on a 3x3xd input is shown in Figure 3.4. Note that the filters always act through the entire depth of the input volume.

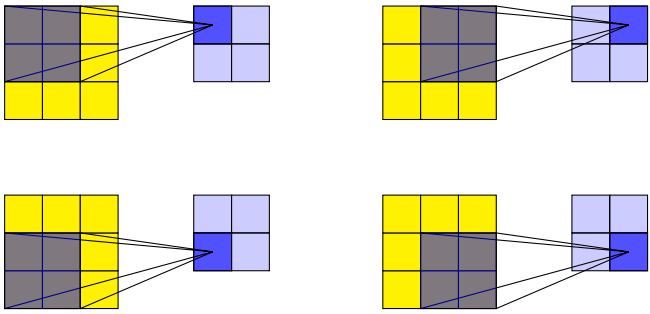


Figure 3.4: *Example of a 2x2xd filter with stride 1 acting on a 3x3xd input volume producing a 2x2 feature map where each cell represents a neuron. Note that the input volume is three dimensional.*

Another thing that distinguishes a neuron in a feature map and a neuron in a fully connected layer is that the affine transformation is the same for all neurons in the same feature map, one often says that the neurons share weights and biases. Thus the feature map can be seen as a convolution between the input volume and a filter followed by an activation function σ. The activations a in a feature

map can be described with the activations in the input volume x and the filter represented by the weights w and bias b by

$$a(n, m) = \sigma\Big(\sum_{k=0}^{D-1} \sum_{j=0}^{F-1} \sum_{i=0}^{F-1} w(i, j, k) \cdot x(i + S \cdot n, j + S \cdot m, k) + b \Big) \tag{3.3}$$

where F is the filter size, S is the stride of the filter and D is the depth of the input.

To connect multiple convolutional layers in series is possible since both the input and output are volumes, this allows for the hierarchical feature extraction described earlier. The first convolutional layer takes an image as the input volume. A normal RGB image can be seen as a three-dimensional volume with depth three, the position of a pixel is two dimensional and each pixel has values over three channels, red, green and blue color content of the pixel.

3.2.2 Dimension reduction

The data passing through the convolutional network is regularly reduced in dimension. This is done to remove redundant information and decrease the number of needed calculations. Two methods to accomplish this are to increase the stride of the filters or add pooling layers.

The stride of the filter determines the spacing with which the filters slide over the data. A larger stride corresponds to a lower resolution in the output volume.

A pooling layer fuses activations from neurons which are nearby each other to a single output. This can be done in multiple ways. Some of the most used ways are by letting the mean, median and maximum value of neighboring neuron activations be the pooled output.

3.2.3 Examples of CNN architectures

Some notable CNN architectures that have been proven useful for image recognition tasks are AlexNet [15], GoogLeNet [23] and ResNet[9]. All have won the software contest ImageNet Large Scale Visual Recognition Challenge, where the task is to detect and classify objects in images.

A breakthrough for CNNs came in 2012 when AlexNet outperformed all other competitors in the ILSVRC. All winners of ILSVRC after 2012 have been based on CNNs. AlexNet has the plain architecture described above with stacked convolutional layers, pooling layers and ending with three fully connected layers. An overview of the architecture is shown in Figure 3.5.

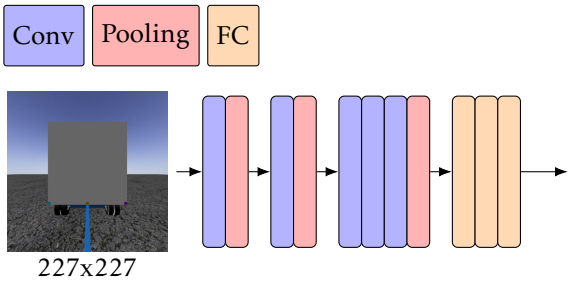


Figure 3.5: *Overview of the architecture for AlexNet.*

GoogleNet won the contest in 2014 and consists of 21 convolutional layers and one fully connected layer. Some of the convolutional layers form a more complex structure than the ones described previously and constitutes so-called "inception modules". An intricate description of an inception module is outside the scope of this thesis but it can crudely be described as a module that trains filters of different sizes, 1x1, 3x3 and 5x5 to be able to cover a large area without exceedingly diminishing the resolution. An overview of the architecture is shown in Figure 3.6.

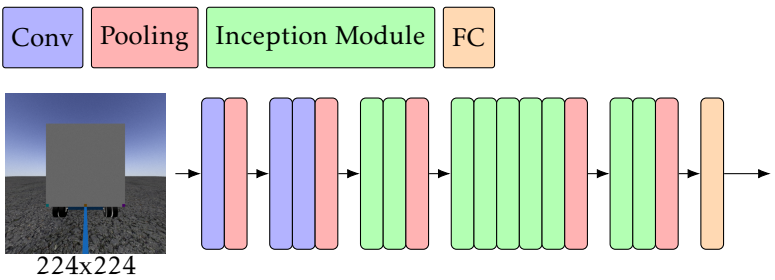


Figure 3.6: *Overview of the architecture for GoogLeNet.*

The winner of the 2015 contest was ResNet or Residual Neural Network. ResNet introduced shortcut connections between layers to tackle the vanishing gradient problem, a problem where the gradients that are used for changing the parameters during training get smaller when neural networks get deeper. These shortcuts made it possible to train deeper networks more efficiently. The version of ResNet that won in 2015 had 152 layers, compare that with the previous winner GoogleNet that consisted of 22 layers. An overview of the architecture is shown in Figure 3.7.

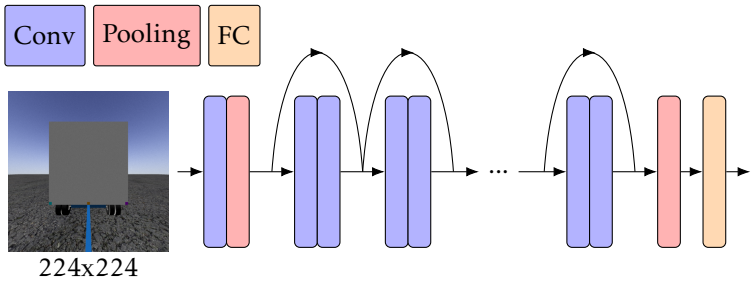


Figure 3.7: *Overview of the architecture of ResNet*

3.3 Training neural networks

This section describes ways to train a neural network to approximate the desired mapping.

3.3.1 Training data

The training data that are used to train a neural network is essential for its performance since the parameters of a neural network are adjusted solely based on the training data. Thus the training data set needs to be representative of data the network will see when applied and a considerate amount of work should be spent on collecting high-quality training data.

3.3.2 Cost function

A performance measure is needed to train an ANN. This performance measure is often called the cost function or objective function and is denoted J. A net that performs well is a network the makes the value J small and the meaning of training a network is adjusting the weights W and biases b of the network to decrease the value of J which can be seen as the optimization problem

$$\min_{W,b} J(W, b, x, y) \tag{3.4}$$

where x and y are the input and the output to the ANN.

When using supervised learning one tries to find W and b to minimize the sum of the cost functions for all training examples, resulting in the optimization problem

$$\min_{W,b} \sum J(W, b, x_t, y_t) \tag{3.5}$$

where the sum is taken over all the training data with x_t and y_t being the input and output of a training example.

The most common cost function for regression tasks, and also the one used in this thesis, is a quadratic cost function given by

$$J(W, b, x_t, y_t) = \frac{1}{2}(y_t - \hat{y})^2 \tag{3.6}$$

where $\hat{y}$ is the estimate from the neural network when the input is x_t and also depends on the weights W and biases b of the neural network.

3.3.3 Adjusting the weights and biases

Gradient descent is often used in practice to determine how much the weights and biases should be changed to decrease J. The idea is to change the parameters in the negative direction of the gradient of J with respect to the parameters. A step in the gradient descent algorithm is given by

$$w_n = w_{n-1} - \eta \nabla J_w \tag{3.7}$$

where η is called the step size or the learning rate and decides how aggressively the parameter should follow the gradient in each step.

A version of gradient descent that is more computationally efficient is stochastic gradient descent (SGD). Instead of taking all the training data into account when calculating ∇J, SGD calculates the gradient from a subset of the training data, called a mini-batch. As an example, if the training data is divided into 10 mini-batches 10 gradient descent steps will be done for each pass through the training data. One pass through the training data is called an epoch and a pass through a mini-batch is called an iteration during training.

A version of SGD is stochastic gradient descent with momentum (SGDM). The difference between SGD and SGDM is that an auxiliary parameter v called velocity is adjusted by the gradient instead of directly adjusting the weights w. The weight adjustment at time-step n is given by

$$v_n = \gamma v_{n-1} - \eta \nabla J_w \tag{3.8a}$$
$$w_n = w_{n-1} - v_n \tag{3.8b}$$

where γ controls how much of the gradient from previous iterations should be used in the current iteration. By taking the gradient from previous iterations

into account, inertia to the change of a parameter is gained. This results in the parameter changing in a reasonable direction even with noisy data.

The gradient with respect to each weight/bias is calculated through a method called backpropagation. The input of the training data x_t is passed through the network, often called a forward pass through the network, and produces the output $\hat{y}$. This estimate is compared with the true output y_t and the difference between y_t and $\hat{y}$ is propagated backward by recursively using the chain rule for derivatives. Through this process, all gradients of J with respects to the trainable parameters can be calculated and can be used for a gradient descent step. For a more thorough description of backpropagation, see [10].

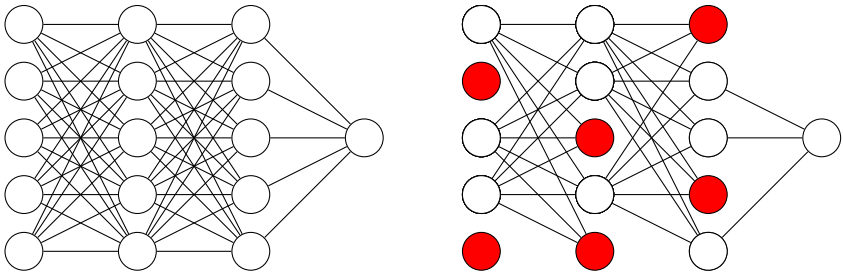
3.3.4 Avoiding overfitting

A problem with neural networks is overfitting of the weights and biases to the training data. The goal of a neural network is to be able to capture the underlying relationship between the input and the output but there is a risk that the network starts to fit unwanted things such as noise in training data since the adjustments of the weights is done solely with respect to the training data. A method to avoid networks from overfitting is dropout [22]. The idea behind dropout is to make neurons less dependent on each other by "deactivating" a random set of neurons during the training phase. More concretely the activation a of a neuron is replaced by $\tilde{a}$ during training where

$$\begin{aligned} \tilde{a} &= R \cdot a \\ R &= \begin{cases} 1, & \text{with probability } 1-p \\ 0, & \text{with probability} p \end{cases} \end{aligned} \tag{3.9}$$

Which results in the neuron being "deactivated" with probability p. The output is normalized with p during test time to account for deactivated neurons during training. An example of dropout is shown in Figure 3.8.

Dropout can be seen as training an ensemble of networks and during test time a sort of average of these networks is used.



(a) Before dropout *(b)* After dropout

Figure 3.8: *Example of a resulting network when using dropout with $p = 0.4$. Neurons in red are deactivated.*

3.3.5 Transfer learning

To use neural networks that have been trained for a specific task to perform another task is called transfer learning. The hierarchical view of CNNs as a feature extractor makes CNNs suitable for transfer learning since the first layers of the networks can be seen as extracting higher level features in a picture such as edges as corners and this information can be useful beyond the specific task that the network has been trained for. It is the last layers of the network that are specific for the particular task where the extracted features are used to perform the desired task [19].

Some examples of the use of transfer learning are presented in [19] and [13]. In [19] a CNN is first trained for classification of images for a certain set of objects and this network is then used as a base for a CNN that classifies objects belonging to entirely different classes. In [13] a CNN trained for classification is used to train a network for regression.

4

State Estimation

This chapter describes how information about the states can be fused in an Extended Kalman Filter and how dominating lines can be extracted from a point cloud using Random Sample Consensus.

4.1 Extended Kalman Filter

The Kalman filter [12] assumes a linear state-space model and estimates the state distribution to be Gaussian. A popular method for handling nonlinear systems is through an extended Kalman filter (EKF) which linearize the model around the latest state estimates. This model is used to propagate the covariance of the state distribution and to weigh information from dynamics and measurements while the mean of the state distribution is propagated through the nonlinear model.

4.1.1 Linearization

A discrete-time dynamical system with state $\mathbf{x}_k$ and control $\mathbf{u}_k$ at time instance k can be modeled by

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + v_k \tag{4.1a}$$

$$\mathbf{y}_k = \mathbf{h}(\mathbf{x}_k) + e_k \tag{4.1b}$$

where (4.1a) describes how the states evolve based on the current state and the current control and (4.1b) describes the measurements $\mathbf{y}_k$ given the current state. v_k is the process noise and e_k is measurement noise and are used to model uncertainty in the dynamics and measurements.

In an EKF the uncertainty of a state estimate is propagated through a linear system and the weighting of information from the dynamics and the measurements is also done based on a linear system. Hence a linear approximation of the dynamical system needs to be used. By calculating the following Jacobians evaluated in the previous state estimate

$$\mathbf{F}_k = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}_{k-1|k-1},\mathbf{u}_k} \tag{4.2a}$$

$$\mathbf{G}_k = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right|_{\mathbf{x}_{k-1|k-1},\mathbf{u}_k} \tag{4.2b}$$

$$\mathbf{H}_k = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\mathbf{x}_{k|k-1}} \tag{4.2c}$$

the following linear time-discrete dynamical system can be obtained to act as a linear approximation of (4.1) in each time-step

$$\mathbf{x}_{k+1} = \mathbf{F}_k \mathbf{x}_k + \mathbf{G}_k \mathbf{u}_k + v_k \tag{4.3a}$$

$$\mathbf{y}_k = \mathbf{H}_k \mathbf{x}_k + e_k \tag{4.3b}$$

4.1.2 Prediction step

In the prediction step, the current state $\hat{\mathbf{x}}_{k-1|k-1}$ and the system dynamics are used to obtain a prediction of the state in the next time instance. The mean of the current state is propagated through the nonlinear dynamics and the covariance is propagated by using the linearized dynamics. Thus, the predicted state $\hat{\mathbf{x}}_{k|k-1}$ and its covariance $\mathbf{P}_{k|k-1}$ are given by

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{f}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) \tag{4.4a}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \tag{4.4b}$$

where $\mathbf{Q}_k$ is the covariance of the process noise and is supposed to capture uncertainty in the dynamics.

4.1.3 Correction step

In the correction step, the predicted state and measurements are fused together to obtain an estimate of the state. The weighting between the predicted state and the measurement is done with the Kalman gain K_k. The corrected state estimate $\hat{\mathbf{x}}_{k|k}$ and the corresponding covariance $\hat{\mathbf{P}}_{k|k}$ are given by

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \left(\mathbf{R}_k + \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T \right) \tag{4.5a}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \left(\mathbf{y}_k - \mathbf{h}(\hat{\mathbf{x}}_{k|k-1}) \right) \tag{4.5b}$$

$$\mathbf{P}_{k|k} = \left(\mathbf{I} - \mathbf{K}_k \mathbf{H}_k \right) \mathbf{P}_{k|k-1} \tag{4.5c}$$

where $\mathbf{R}_k$ is the measurement noise and is suppose to capture uncertainty in the measurements.

4.1.4 Discretization of dynamics

Since the dynamics given in (2.3) is given as a differential equation and the dynamics in (4.1a) is a difference equation, the dynamics need to be discretized. This can be done using Euler sampling with sample time T_s, resulting in the approximation

$$\dot{\mathbf{x}} \approx \frac{\mathbf{x}_{k+1} - \mathbf{x}_k}{T_s} \tag{4.6}$$

The continuous dynamics can then be discretized the following way

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) + \mathbf{v} \xrightarrow{\text{Euler sampling}} \mathbf{x}_{k+1} = \mathbf{x}_k + T_s \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{v}_k \tag{4.7}$$

4.1.5 Motion and measurement models

Three different types of EKFs were used to estimate the states of the system. First, a filter only used the dynamics of β_2 and β_3, a second filter used the full dynamics given by (2.3) and finally a filter where the lengths L_2,L_3 and L_k were augmented as states and the dynamics of β_2 and β_3 were used.

The motion models and measurement models used in these filters are presented below. The control signal u for all models is $u = (v, \alpha)^T$.

Angle dynamics

The states x in this case are $(\beta_3, \beta_2)^T$ and dynamics are given by

$$f^{\text{ang}}(x, u) = \begin{pmatrix} v \cos\beta_2 \left(\frac{1}{L_2} \left(\tan\beta_2 - \frac{M_1}{L_1} \tan\alpha \right) - \frac{\sin\beta_3}{L_3} \left(1 + \frac{M_1}{L_1} \tan\beta_2 \tan\alpha \right) \right) \\ \frac{v}{L_1 L_2} \left(L_2 \tan\alpha - L_1 \sin\beta_2 + M_1 \cos\beta_2 \tan\alpha \right) \end{pmatrix} \tag{4.8}$$

Full dynamics

The states x in this case are $(x_3, y_3, \theta_3, \beta_3, \beta_2)^T$ and the dynamics are given by

$$f^{\text{full}}(x,u) = \begin{pmatrix} v\cos\beta_3\cos\beta_2\Big(1+\frac{M_1}{L_1}\tan\beta_2\tan\alpha\Big)\cos\theta_3 \\ v\cos\beta_3\cos\beta_2\Big(1+\frac{M_1}{L_1}\tan\beta_2\tan\alpha\Big)\sin\theta_3 \\ \frac{v\sin\beta_3\cos\beta_2}{L_3}\Big(1+\frac{M_1}{L_1}\tan\beta_2\tan\alpha\Big) \\ v\cos\beta_2\Big(\frac{1}{L_2}\Big(\tan\beta_2-\frac{M_1}{L_1}\tan\alpha\Big)-\frac{\sin\beta_3}{L_3}\Big(1+\frac{M_1}{L_1}\tan\beta_2\tan\alpha\Big)\Big) \\ \frac{v}{L_1L_2}\Big(L_2\tan\alpha-L_1\sin\beta_2+M_1\cos\beta_2\tan\alpha\Big) \end{pmatrix} \tag{4.9}$$

Augmented dynamics

The states x in this case are $(\beta_3, \beta_2, L_2, L_3, L_k)^T$ and the dynamics are given by

$$f^{\text{aug}}(x,u) = \begin{pmatrix} v\cos\beta_2\Big(\frac{1}{L_2}\Big(\tan\beta_2-\frac{M_1}{L_1}\tan\alpha\Big)-\frac{\sin\beta_3}{L_3}\Big(1+\frac{M_1}{L_1}\tan\beta_2\tan\alpha\Big)\Big) \\ \frac{v}{L_1L_2}\Big(L_2\tan\alpha-L_1\sin\beta_2+M_1\cos\beta_2\tan\alpha\Big) \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{4.10}$$

Measurements

Four different types of measurements were used in the filters

$$y^{\text{RANSAC}} = (\phi, L_x, L_y)^T \tag{4.11a}$$
$$y^{\text{NN}} = (\hat{\beta}_2^{\text{NN}}, \hat{\beta}_3^{\text{NN}})^T \tag{4.11b}$$
$$y^{\text{marker}} = (u_1, u_2, u_3)^T \tag{4.11c}$$
$$y^{\text{GNSS}} = (x_1, y_1, \theta_1)^T \tag{4.11d}$$

with the following measurement models

$$h^{\text{RANSAC}}(x) = \begin{pmatrix} \beta_2 + \beta_3 \\ -L_2\cos(\beta_2) + L_k\cos(\beta_2+\beta_3) \\ L_2\sin(\beta_2) - L_k\sin(\beta_2+\beta_3) \end{pmatrix} \tag{4.12a}$$

$$h^{\text{NN}}(x) = \begin{pmatrix} \beta_2 \\ \beta_3 \end{pmatrix} \tag{4.12b}$$

$$h^{\text{markers}}(x) = \begin{pmatrix} \frac{f}{s}\frac{L_2\sin(\beta_2)-L_k\sin(\beta_2+\beta_3)+d_1\cos(\beta_2+\beta_3)}{L_2\cos(\beta_2)-L_k\cos(\beta_2+\beta_3)-d_1\sin(\beta_2+\beta_3)} + \frac{W}{2} \\ \frac{f}{s}\frac{L_2\sin(\beta_2)-L_k\sin(\beta_2+\beta_3)+d_2\cos(\beta_2+\beta_3)}{L_2\cos(\beta_2)-L_k\cos(\beta_2+\beta_3)-d_2\sin(\beta_2+\beta_3)} + \frac{W}{2} \\ \frac{f}{s}\frac{L_2\sin(\beta_2)-L_k\sin(\beta_2+\beta_3)+d_3\cos(\beta_2+\beta_3)}{L_2\cos(\beta_2)-L_k\cos(\beta_2+\beta_3)-d_3\sin(\beta_2+\beta_3)} + \frac{W}{2} \end{pmatrix} \tag{4.12c}$$

$$h^{\text{GNSS}}(x) = \begin{pmatrix} x_3 + M_1\cos(\theta_3+\beta_2+\beta_3) + L_2\cos(\theta_3+\beta_3) + L_3\cos(\theta_3) \\ y_3 + M_1\sin(\theta_3+\beta_2+\beta_3) + L_2\sin(\theta_3+\beta_3) + L_3\sin(\theta_3) \\ \theta_3+\beta_2+\beta_3 \end{pmatrix} \tag{4.12d}$$

y^{GNSS} was only used for the filter using the full dynamics and the augmented filter only used measurement from y^{RANSAC}.

4.1.6 Initialization of EKF

Since a linearized model is used when propagating the state distribution there are no guarantees for an EKF to converge to the right estimate. Hence it is important to initialize the filter with a reasonable start estimate to mitigate divergence of the estimates. One way of obtaining reasonable start guesses is to use measurements from the LIDAR and the GNSS before starting the filter.

The mapping from β_2 and β_3 to ϕ and L_y given by (2.5) can be inverted to initialize β_2 and β_3. The following relationship holds

$$\beta_2^0 = \arcsin\left(\frac{L_y^0 + L_k\sin(\phi^0)}{L_2}\right) \tag{4.13a}$$

$$\beta_3^0 = \phi^0 - \beta_2^0 \tag{4.13b}$$

where ϕ^0 and L_y^0 are extracted from the initial LIDAR scan through the procedure described in Section 2.4.1.

A measurement from the GNSS can then be used together with β_2^0 and β_3^0 to get

an initialization of the global orientation and position of the trailer

$$x_3^0 = x_1^0 - M_1 \cos(\theta_3^0 + \beta_2^0 + \beta_3^0) - L_2 \cos(\theta_3^0 + \beta_3^0) - L_3 \cos(\theta_3^0) \tag{4.14a}$$
$$y_3^0 = y_1^0 - M_1 \sin(\theta_3^0 + \beta_2^0 + \beta_3^0) - L_2 \sin(\theta_3^0 + \beta_3^0) - L_3 \sin(\theta_3^0) \tag{4.14b}$$
$$\theta_3^0 = \theta_1^0 - \beta_2^0 - \beta_3^0 \tag{4.14c}$$

4.2 Random Sample Consensus

Random sample consensus (RANSAC) [7] is a robust method for fitting a set of data point to a model M iteratively. The main idea is to generate multiple model candidates from a randomly selected subset of the data D and pick the model that best fits all data points in D.

More formally the models are picked from a model set $\mathcal{M}(\theta)$ with model parameters θ. In iteration n the model parameters θ_n is generated from a model that fits best to a randomly selected subset of the data which gives the model candidate $M_n = \mathcal{M}(\theta_n)$. Each data point $d \in D$ is then decided to agree or to disagree with the model M_n. This is decided from a consensus function C which is defined as

$$C(d, M_n) = \begin{cases} 1, & \text{if } d \text{ agrees with } M_n \\ 0, & \text{if } d \text{ disagrees with } M_n \end{cases} \tag{4.15}$$

The consensus score S_c for M_n is then calculated and given by

$$S_c(D, M_n) = \sum_{d \in D} C(d, M_n) \tag{4.16}$$

A high value of S_c means that the data fits well with the model and thus the model candidate with the highest S_c is taken as the model prediction.

To summarise the steps taken in an iteration of RANSAC

1. Generate a model candidate M_n from a random subset of D
2. Score the model candidate by calculating $S_c(D, M_n)$
3. If $S_c(D, M_n)$ is the highest score so far set M_n as the model prediction, otherwise dismiss M_n.

4.2.1 Line fitting using RANSAC

In this thesis, RANSAC is used to fit lines to a 2-dimensional point cloud. Hence D consists of points in $\mathbb{R}^2$ and $\mathcal{M}$ consists of lines in $\mathbb{R}^2$. Let the model candidate,

which is a line, be called L_n, the consensus function used is

$$C(d, L_n) = \begin{cases} 1, & \text{if distance}(L_n, d) < \epsilon \\ 0, & \text{if distance}(L_n, d) \geq \epsilon \end{cases} \tag{4.17}$$

where the distance intended is the perpendicular distance between a point and a line and ϵ is a design parameter to decide how far from the line a point needs to be to be considered an outlier.

The algorithm can be employed iteratively where the line with the inliers of the most dominating line is dismissed and a line is fitted to remaining points. Multiple lines can be extracted from a point cloud using this technique.

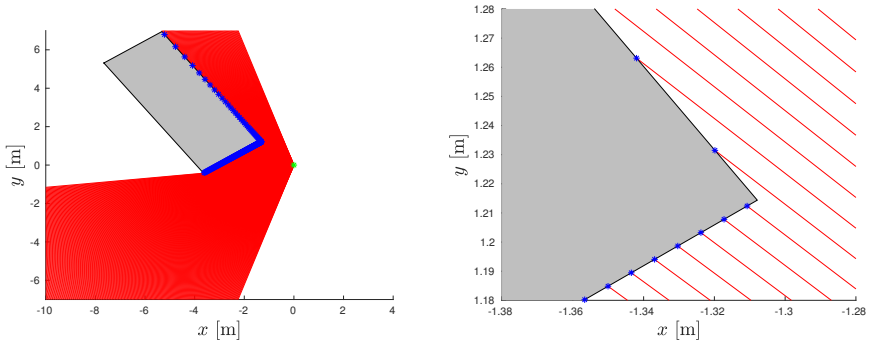
5
Data Collection and Training

This section describes how the data sets that were used for training and testing were obtained and how the training procedure of the neural networks was carried through. It also describes which neural network architectures were used and how these were decided upon.

5.1 Simulation

Simulation environments in MATLAB and Gazebo [14] were established to get access to the large amount of data needed to train neural networks and to test the different estimation methods. The purpose of the MATLAB simulation environment was to generate LIDAR scans for a rectangular trailer while the purpose of the Gazebo simulation environment was to generate camera images and more realistic point clouds.

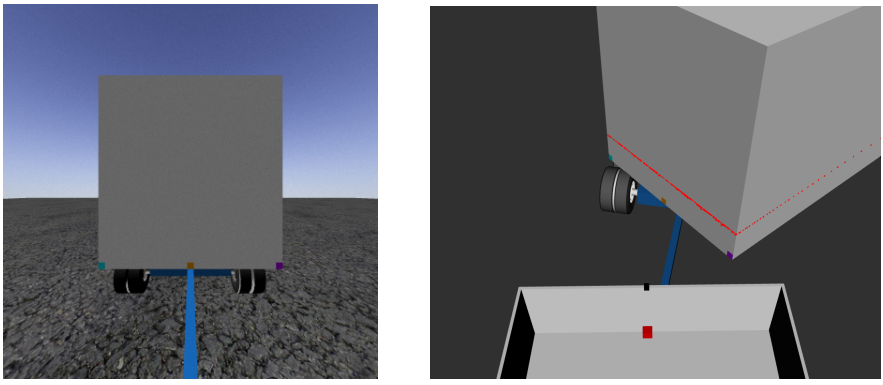
Point clouds in MATLAB were generated by calculating the intersection between rays originating from the position of the LIDAR sensor and a box representing the trailer. An example of a generated LIDAR scan without noise in MATLAB is shown in Figure 5.1



(a) *(b)*

Figure 5.1: *Example of a generated point cloud in MATLAB. The trailer is marked in gray, the rays from the LIDAR is marked in red and the resulting point cloud is marked in blue.*

In Gazebo, the trailer was modeled as a box and a LIDAR sensor and a camera were mounted on the back of the truck. A sample image from the camera is shown in Figure 5.2a and a sample scan from the LIDAR is shown in Figure 5.2b. Gaussian noise was added to both the LIDAR measurements and the camera images to make the simulated data better correspond with real-world data.



(a) *(b)*

Figure 5.2: *(a) Example of a generated image from the rear camera in Gazebo. (b) A scan from the LIDAR in Gazebo, the red dots are the generated point cloud.*

Two types of data sets were generated. The first type was used for training the neural networks and was generated by taking combinations of β_2 and β_3 on a grid with the angles varying on the interval $[-\frac{\pi}{4}, \frac{\pi}{4}]$. This covered a wide range

of the possible orientations that the system can be in when driving. The intervals were discretized with 180 points resulting in an angle resolution of 0.5°.

Neural networks only handle fixed sized inputs and to make sure a scan always had the same length, non-hits were treated as a hit in the origin, i.e., coordinates $(0, 0)$. Setting a miss as a hit in the origin was deemed reasonable since this would result in the point not contributing to the activation of the neurons.

For the camera data, objects such as buildings and cars were put in the background to train the network in more realistic situations. For each data point taken, the global yaw of the system was randomized to get variation in which background objects were seen and to get different lighting in the image. This was done to avoid the CNN from overfitting the background. A montage of some images in the generated training data set is shown in Figure 5.3

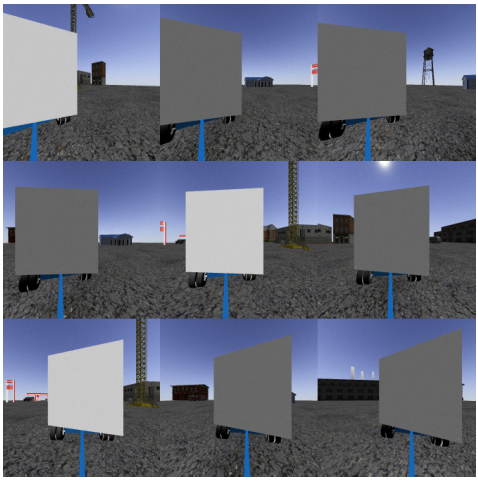


Figure 5.3: *Selection of images in the data set used for training the CNN.*

The second type of data set was used for evaluating the estimation methods and was generated by simulating the system reversing along an eight-shaped trajectory. The dynamics in (2.3) was used to simulate the motion of the system.

5.2 Training

The collected data set for learning was split into training data and validation data with the partition 75% training data 25% validation data. The neural networks were implemented using the MATLAB Neural Network Toolbox and were trained by using a GeForce GTX 970 graphics card. Training was continued until significant improvements on the validation data stopped. The training was done with

SGDM with the learning rate $\eta = 0.001$, momentum $\gamma = 0.9$, dropout probability $p = 0.5$ and mini-batch size 128.

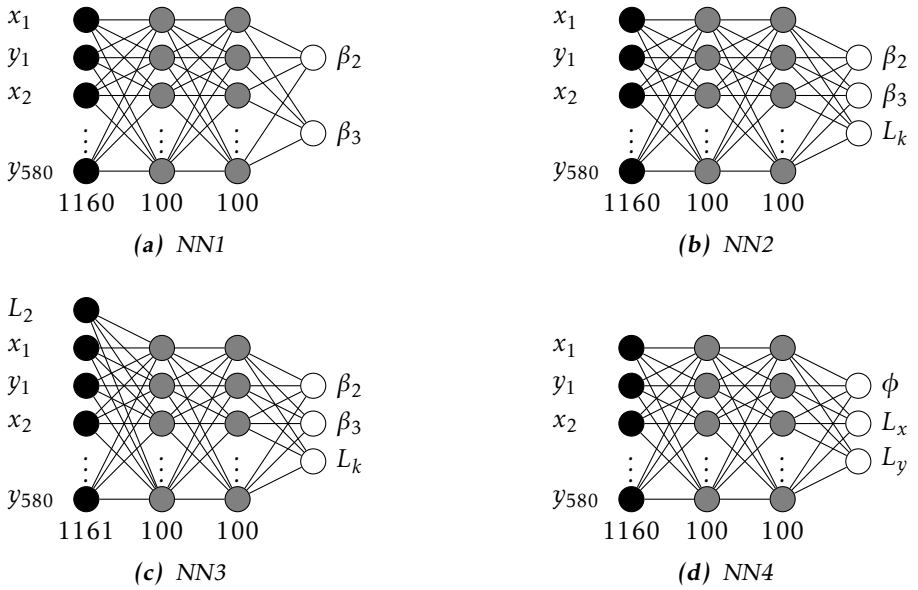
5.3 Network architectures

5.3.1 LIDAR

Four different fully connected neural networks, each with two hidden layers containing 100 neurons, were trained on the LIDAR data. Three networks were designed for end-to-end estimation under different circumstances. The networks were called NN1, NN2 and NN3 and the different cases underlying the design of each network were

- **NN1** - L_k and L_2 are completely known and fixed.
- **NN2** - L_k is unknown and L_2 is known and fixed.
- **NN3** - L_k is unknown and L_2 is known but allowed to vary.

A fourth network was trained to estimate the same intermediary quantities ϕ, L_x and L_y as RANSAC estimates. This network was called NN4. All the networks are shown in Figure 5.4.



(a) NN1 *(b) NN2* *(c) NN3* *(d) NN4*

Figure 5.4: *Network architectures used on LIDAR data.*

Preliminary tests were done with training CNNs on the point clouds. This seemed reasonable since there is a correlation between nearby data points in the point cloud. However, these tests did not yield any better results than feeding the point clouds directly into a fully connected neural network. Thus fully connected networks were picked in favor of CNNs for the LIDAR data.

5.3.2 Camera

A CNN was used to estimate β_2 and β_3 from an image. The architecture of the CNN was based on GoogleNet [23]. Googlenet was chosen because of its balance of obtaining good accuracy in combination with not being too computationally demanding. The CNN had been pretrained for classification on the ImageNet Dataset [6] and was used for initializing the weights of the convolutional layers. The last fully connected layer was replaced with two fully connected layers of size 1024 with dropout to suit the regression task better. A very similar course of action was taken in [13] which proved successful for a regression task, this was another reason for picking a Googlenet architecture. The used architecture is shown in Figure 5.5.

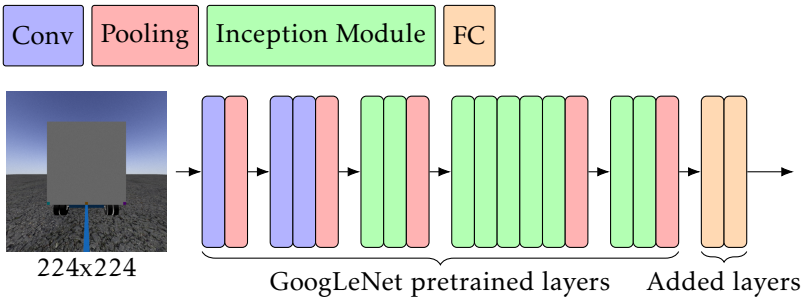


Figure 5.5: *The architecture of the CNN used for angle estimation.*

Images were down-sampled to 224x224 before being fed to the network since the pretrained network was trained on images with that size.

6 Result

This chapter presents the resulting estimates when using the methods presented in previous chapters on data generated in simulation. First, the raw estimates based on LIDAR and camera data are presented and then these estimates are fused together with the dynamics of the system in an Extended Kalman Filter to improve the estimates.

6.1 LIDAR

In this section results of the estimations based on LIDAR-data are presented and interpreted. First, results from the ideal case when the trailer is a perfect box are presented and the results from RANSAC and the different neural networks are compared. Later, results from cases when the shape of the trailer is less regular is shown which leads to some discussion considering robustness and flexibility of the methods.

6.1.1 RANSAC

Figure 6.1 shows the estimates of ϕ, L_x and L_y when using RANSAC to extract lines from the LIDAR-data and using the methods described in Section 2.4.1 to extract the relevant quantities. 100 hypotheses were generated in each iteration and a point was deemed an outlier if it was farther away than 0.05 meters from the generated line.

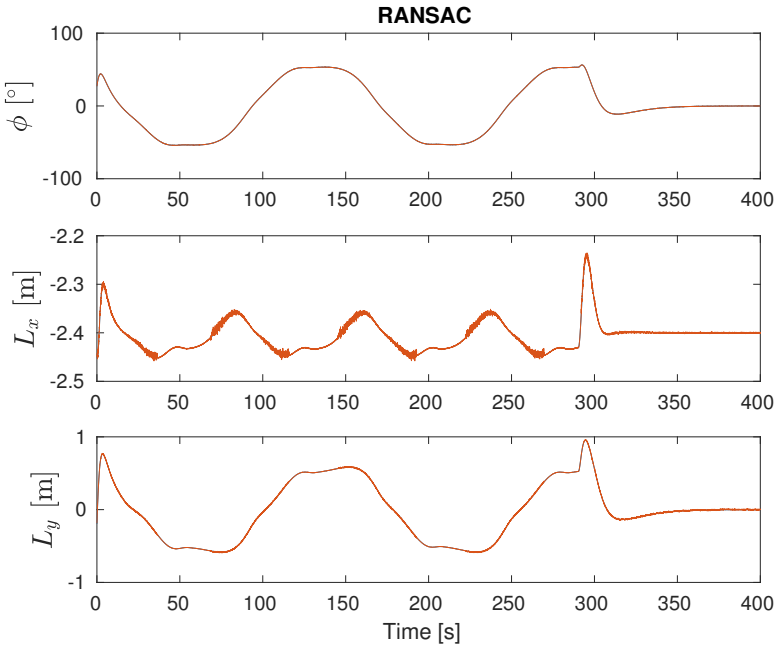


Figure 6.1: *Estimation of ϕ, L_x and L_y using RANSAC compared with ground truth.*

The estimate of ϕ is very close to the ground truth since the line fitted to the point cloud is done very accurately. There are some small biases for both L_x and L_y due to the fact that when the LIDAR only hits the back of the trailer the estimate is decided as the mean of the two outermost points on the fitted line segment. This mean would be the middle point if the two outermost points were exactly at the corners of the trailer, but since this is not the case in practice there will be a small bias.

One can also see that the variances of the estimates of L_x and L_y vary in different time intervals. This is because two lines which are perpendicular to each other have been found by the algorithm and thus one corner of the trailer can be decided with very high accuracy.

6.1.2 Neural Networks

End-to-end

Three different networks were trained end-to-end. One network, called NN1, was trained on training data when L_2 and L_k were fixed. A second network, called NN2, was trained on training data when L_2 was fixed but L_k was allowed to vary. Finally, a third network, called NN3, was trained on training data with varying L_2 and L_k. NN3 also received L_2 as an input since a point cloud is not uniquely defined when β_2, β_3, L_2 and L_k are free simultaneously, which was shown in Section 2.6. A summary of these cases is given in Table 6.1.

Table 6.1: *Lengths that the neural networks were trained on.*

	L_2 [m]	L_k [m]
NN1	2.8	0.4
NN2	2.8	$[0.1, 1]$
NN3	$[2, 4]$	$[0.1, 1]$

The networks were then tested in three different situations. One nominal case when $L_2 = 2.8$ and $L_k = 0.4$, this result is shown in Figure 6.2. A second case when $L_2 = 2.8$ and $L_k = 0.625$, shown in Figure 6.3. Finally, a third case when $L_2 = 3.25$ and $L_k = 0.625$, shown in Figure 6.4.

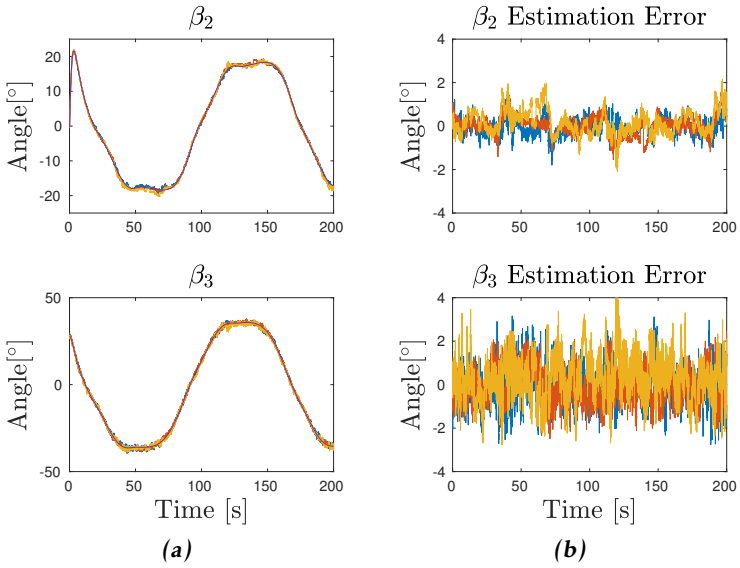


Figure 6.2: *End-to-end estimation of β_2 and β_3 with $L_2 = 2.8$ meters and $L_k = 0.4$ meters for NN1,NN2 and NN3 compared with ground truth.*

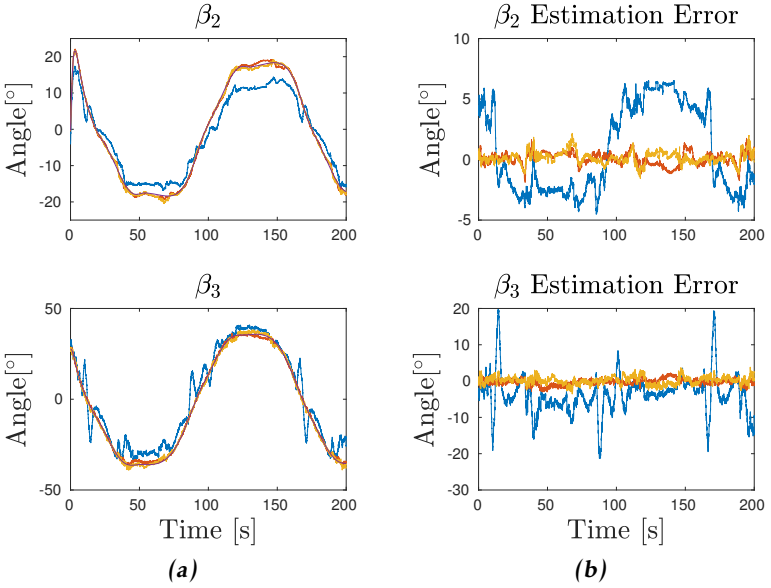


Figure 6.3: *End-to-end estimation of β_2 and β_3 with $L_2 = 2.8$ meters and $L_k = 0.625$ meters for NN1,NN2 and NN3 compared with ground truth.*

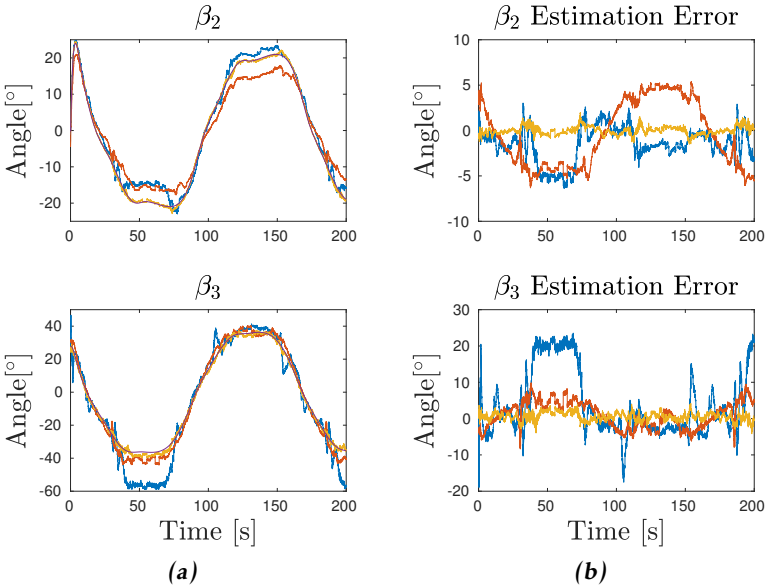
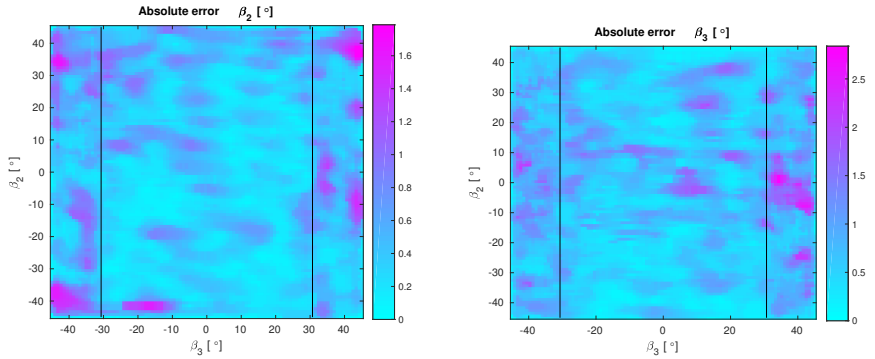


Figure 6.4: *End-to-end estimation of β_2 and β_3 with $L_2 = 3.25$ meters and $L_k = 0.625$ meters for NN1,NN2 and NN3 compared with ground truth.*

These results highlight the well-known fact that neural networks perform well on data that is similar to the training data but are bad at extrapolating outside of that realm. It is important to note that the amount of training data needed for training a more flexible network such as NN3 is much larger compared with the training data needed for NN2 which in turn needs more training than NN1. So the needed amount of training data increases together with the degrees of freedom that the network should be able to handle.

NN3 was also tested on data on multiple combinations of β_2 and β_3 to investigate which combinations of β_2 and β_3 were most problematic. These results are shown in Figure 6.5 which shows the absolute error for β_2 and β_3 for different combinations of angles. The figures also mark critical points where the LIDAR starts to hit both sides of the trailer.



(a) *(b)*

Figure 6.5: *Absolute error of β_2 and β_3 estimates from NN3. The solid black lines marks where the LIDAR starts to hit two sides.*

The network seems to be performing well for relatively small angles and worse at the edges. Another interesting point is that the plots, especially 6.5b, is not symmetric which suggest that the performance of the network could probably be improved with more training. The reason for this is that if the network performs well for (β_2, β_3) it should be able to perform with the same precision for $(-\beta_2, -\beta_3)$ since the point cloud generated for this combination should be the same except mirrored in the x-axis.

Indirect estimation

A neural network was also trained to predict the same parameters as RANSAC, i.e. ϕ, L_x and L_y. The result of this network is shown in Figure 6.6 and training data which the network was trained on covered the same cases as NN3 was trained for. Note the scale difference on the y-axis for L_x compared with L_y.

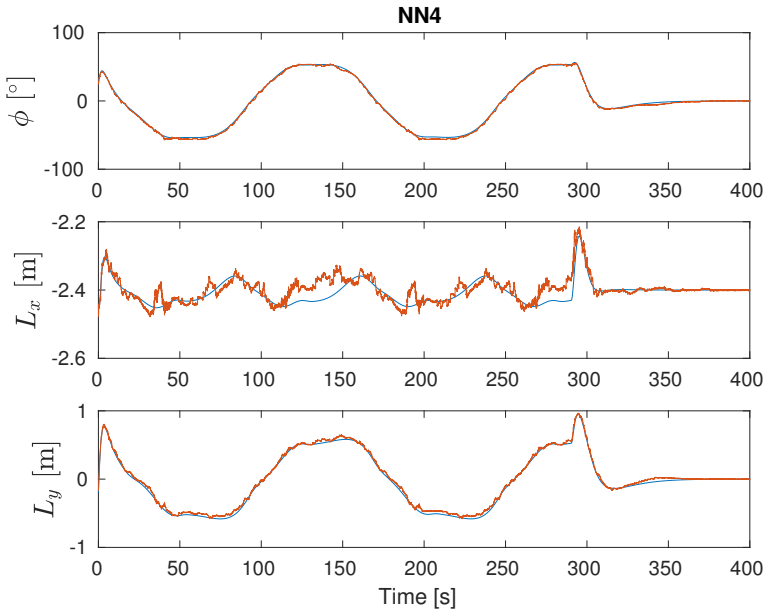


Figure 6.6: *Estimation of ϕ, L_x and L_y using NN4 compared with ground truth.*

6.1.3 Comparison of RANSAC and Neural Networks

The mapping from β_2 and β_3 to ϕ and L_y can be inverted to be able to compare the results from RANSAC and NN4 with the end-to-end estimation. The following relationship holds

$$\beta_2 = \arcsin\left(\frac{L_y + L_k \sin(\phi)}{L_2}\right) \tag{6.1a}$$

$$\beta_3 = \phi - \beta_2 \tag{6.1b}$$

The estimates from NN3 and the transformed estimates from RANSAC and NN4 using (6.1) are shown together in Figure 6.7 and the RMSE of the estimates are presented in Table 6.2.

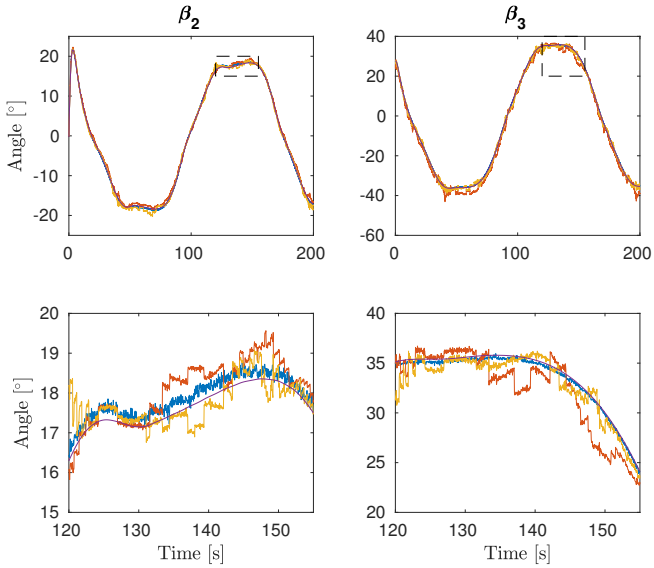


Figure 6.7: *Resulting estimations of β_2 and β_3 from NN3, NN4 and RANSAC compared with ground truth.*

Table 6.2: *RMSE for estimates using different methods.*

	β_2	β_3
NN3	0.55°	0.99°
NN4	0.50°	1.47°
RANSAC	0.18°	0.26°

All methods give reasonable estimates of the angles. However, RANSAC seems to be more accurate than both the neural networks. It also suggests that β_2 is easier to estimate accurately compared with β_3. One thing to keep in mind is that the networks were trained on data with an angular resolution of 0.5° and both NN3 and NN4 seem to have an RMSE of β_2 that is fairly close to the error one can expect from discretization noise.

6.1.4 Robustness

Prior tests have been done assuming the trailer can be modeled as a box. This is a reasonable model in practice but some artifacts such as cables or tubes hanging from the trailer can make the shape of the point cloud deviate from that of a perfect box. Data was generated with protrusions on the trailer to investigate the robustness of the estimation methods when cables and tubes are present. An example of such a scan is shown in Figure 6.8

Figure 6.8: *Example of a LIDAR scan of a trailer with protrusions.*

The resulting estimations from NN3 and RANSAC with and without protrusions are shown in Figure 6.9 and the RMSE are shown in Table 6.3.

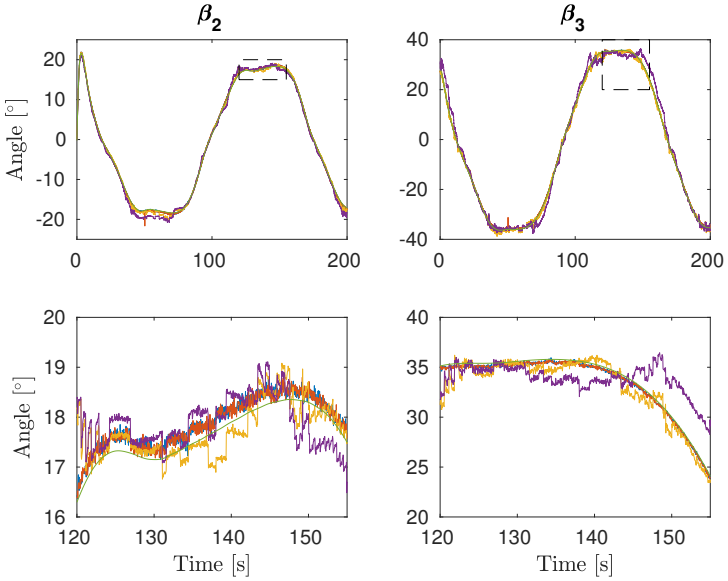


Figure 6.9: *Resulting estimations of β_2 and β_3 for RANSAC, RANSAC with protrusions, NN3 and NN3 with protrusions compared with ground truth.*

Table 6.3: *RMSE for estimates using different methods.*

	β_2	β_3
NN3	0.55°	0.99°
NN3 with protrusions	0.92°	2.16°
RANSAC	0.18°	0.26°
RANSAC with protrusions	0.19°	0.26°

The results show that RANSAC handles protrusions better than neural networks. The reason for this is that RANSAC efficiently omits outliers which means that the points belonging to the protrusions do not affect the estimates. Neural networks, on the other hand, treat each point as an input to a continuous function and hence a small change in one point will produce a slight change in the output. The estimates are barely affected when points are perturbed by white noise with zero mean since the deviations cancel out. However, if multiple points are perturbed in the same direction, as is the case when there are protrusions, there will be a noticeable bias in the output which was shown in the result above.

A possible way of making the neural networks handling protrusions better would be to include this case in the training data.

6.1.5 Non-rectangular Trailer

The results from previous sections suggest that RANSAC performs better than the trained networks both in accuracy and robustness. This is expected since RANSAC uses a predetermined model, that the trailer is rectangular, and produces angle estimates by fitting the data to the model. The neural networks try to learn this model through training but it does not use the model directly. Thus the neural network learns to approximate the model that RANSAC uses which suggest that RANSAC will do a better job compared to a neural network. The strength of neural networks is the flexibility to learn an arbitrary model through training, all that is needed is sufficient training data. To train the networks on point clouds from a rectangular trailer is therefore the same as for a trailer with arbitrary shape, as the one producing the scan in Figure 6.10.

Figure 6.10: *An example of a LIDAR scan of a trailer with a non-rectangular shape.*

To show the flexibility of neural networks, point clouds from a LIDAR hitting a trailer with a non-rectangular shape was generated and a neural network with the same structure as NN1 was trained to estimate the angles. The network was then tested on this new truck and trailer system reversing in an eight-shaped trajectory and the result is shown in Figure 6.11.

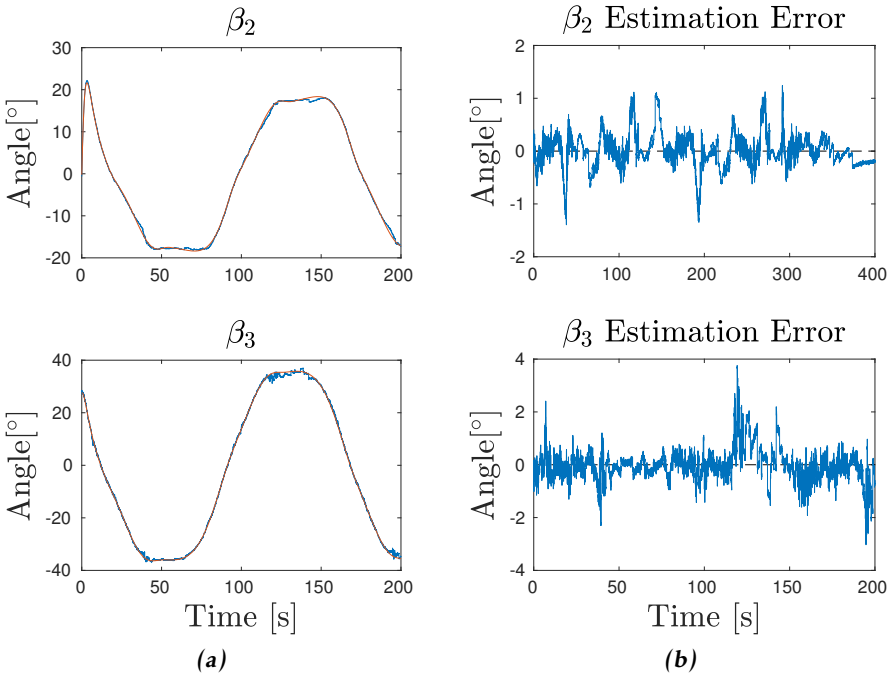


Figure 6.11: *End-to-end estimation of β_2 and β_3 for a non-rectangular trailer using a neural network compared with ground truth.*

The result shows that the neural network approach is not limited to rectangularly shaped trailers which makes a case for using neural networks to estimate the angles when no model of the trailer does exist but a large training set is available.

6.2 Camera

In this section, estimations based on camera images are presented and interpreted. First, the results from the model-based approach of detecting markers on the trailer are presented and after that results from the trained CNN is shown.

6.2.1 Point Detection

Three markers were placed on the short side of the trailer, two in each bottom corner and one in between those. The result of the point detection is shown in Figure 6.12. Note that marker two goes out of picture around the time 10 seconds.

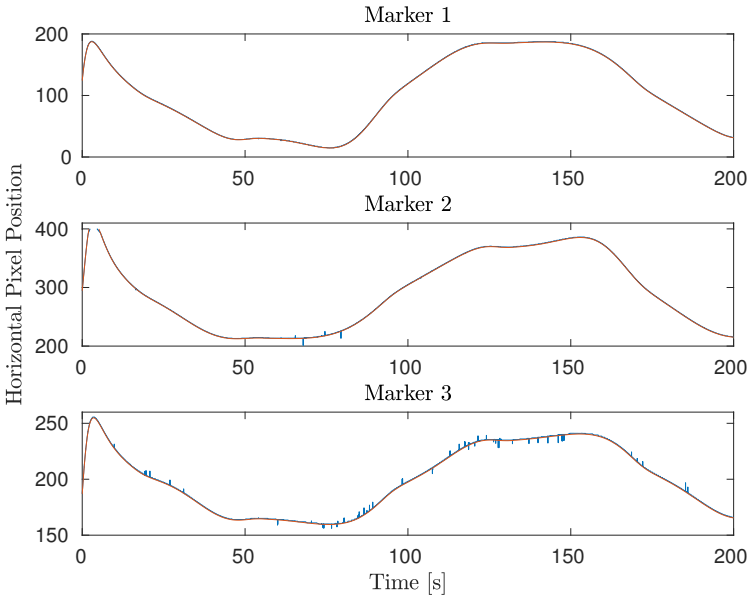


Figure 6.12: *Detected horizontal pixel position of the markers compared with expected position according to the measurement model* (2.9).

The detected pixel position of the markers corresponds well with the measurement model, which is expected since the camera used in simulation are a perfect pinhole camera. However, there are some spikes in the detected pixel position which can impair the performance of the state estimation if these are used directly as measurements. Therefore a median filter was employed to the detection before being used as measurements in the EKF.

6.2.2 Convolutional Neural Network

A CNN was trained with training data for the case when $L_2 = 2.8$ and $L_k = 0.4$. The network was evaluated on camera images generated from the truck and trailer reversing in figure eight with a constant velocity of 1 m/s in simulation. The resulting estimations are shown in Figure 6.13. Important to note is that when using the CNN no markers needs to be attached to the trailer.

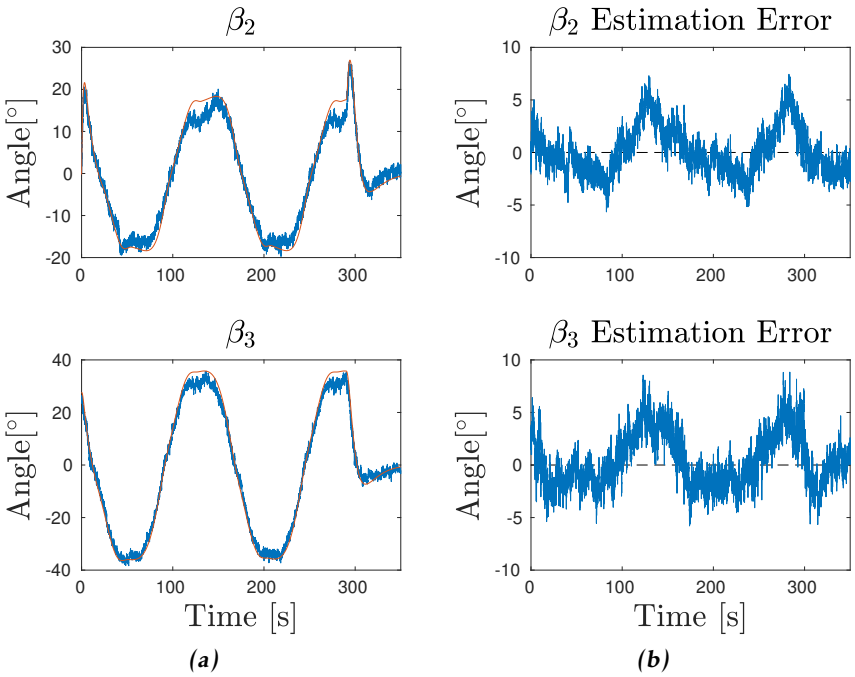


Figure 6.13: *End-to-end estimation of β_2 and β_3 using a CNN compared with ground truth.*

The CNN gives reasonable estimates compared with the ground truth. The accuracy seems to be lower when the absolute values of the angles are larger. A possible explanation for this is that parts of the trailer will be outside the field of view of the camera.

Images from the camera were generated for angles on a grid to investigate further whether the estimates were better or worse for certain angles. These results are shown in Figure 6.14 where the absolute error for β_2 and β_3 are shown for the different combinations of angles. The figure also shows the critical cases when parts of the trailer go out of the view of the camera.

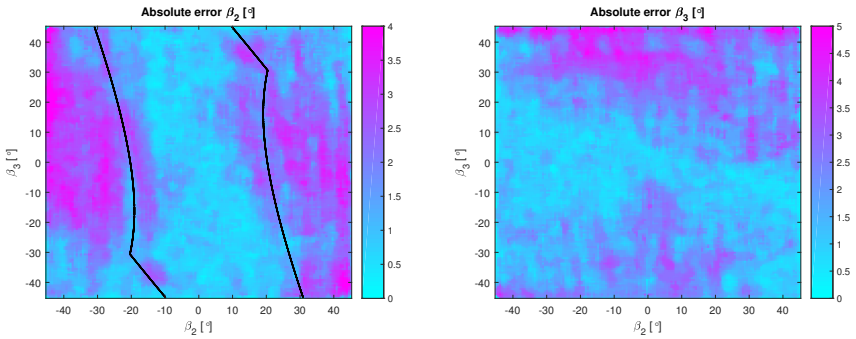


Figure 6.14: *Absolute Error. The solid black lines marks where an edge of the trailer moves out the field of view.*

The CNN seems to perform better at smaller angles compared with larger angles, as could be seen in Figure 6.13. It is clear that the accuracy of β_2 is reduced when parts of the trailer are out of view of the camera. This suggests that the CNN uses information such as the position of the trailer corners in the picture, or something similar that relies on the entire trailer being in view.

Another observation is that the plot for the error of β_3 is asymmetric which suggest, using the same reasoning as in Section 6.1.2, that the network architecture has the potential to perform even better with more training.

Activations

Figure 6.15b shows the activations of 16 of the feature maps from the first convolutional layer when the image in Figure 6.15a was fed to the CNN.

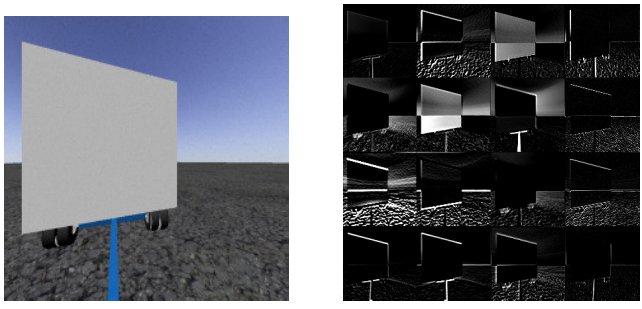


Figure 6.15: *Activations of the first layer of the CNN. A higher activation of a neuron corresponds to a whiter pixel.*

If a filter produces high activations for a particular feature in the picture the filter can be seen as extracting this feature. Figure 6.15 shows how the filters in the first layer extract high-level features such as corners and edges. It also shows that some filters specifically extract the edges of the trailer while some extract the dolly.

6.3 EKF

In this section, the estimates from previous sections are fused with the system dynamics to improve their accuracy and make them smoother. First only the angle dynamics of the system is used and only measurements from the LIDAR and camera are used. Then the full dynamics of the system is used to estimate the global position and orientation of the system, this means that also GNSS measurements are used. Finally results from when the states are extended by augmenting the lengths L_2,L_3 and L_k to account for biases in the geometry are shown.

6.3.1 Angle Dynamics

The angular dynamics for β_3 and β_2 from the vehicle model given by (2.3d) and (2.3e) were used together with the estimates derived in the previous sections. Measurement and process noise were tuned to produce adequate estimates. The process noise for both β_3 and β_2 was set to 0.05 and the final values for the measurement noises are presented in Table 6.4.

Table 6.4: *The tuned measurement noises for the EKF estimating β_2 and β_3.*

	ϕ	L_x	L_y	u_1	u_2	u_3	$\hat{\beta}_3^{NN_3}$	$\hat{\beta}_2^{NN_3}$	$\hat{\beta}_3^{CNN}$	$\hat{\beta}_2^{CNN}$
Noise	10	100	100	500	500	500	10	10	10	10

The filter was initialized using the method described in Section 4.1.6. The result for the state estimates when the vehicle was reversing along an eight-shaped path is shown in Figure 6.16.

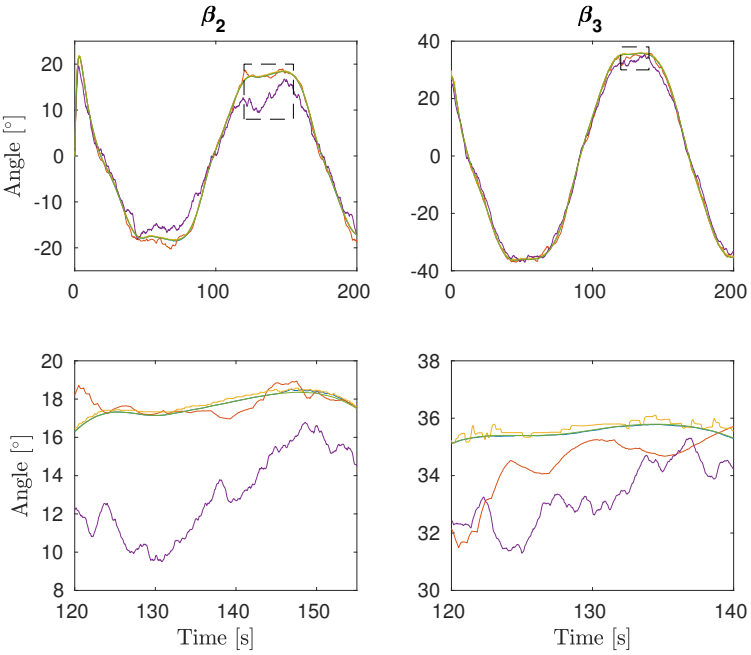


Figure 6.16: *Resulting estimates when using the angle dynamics together with measurements from RANSAC,NN3, Camera point detection and CNN compared with ground truth.*

The results are in accordance with the previous results, the model-based approaches give state estimates with higher precision.

6.3.2 Full Dynamics

Next, the entire dynamics of the system, given by (2.3), was used as well as GNSS measurements to estimate the global pose of the system in addition to the internal state. The process noise for the new dynamics and the GNSS measurements are given in Table 6.5. The same process and measurement noise were used for the angle dynamics and earlier used measurements.

Table 6.5: *The tuned measurement and process noise for the EKF estimating $x_3, y_3, \theta_3, \beta_3$ and β_2.*

	x_1	y_1	θ_1	$\dot{x}_3$	$\dot{y}_3$	$\dot{\theta}_3$
Noise	50	50	50	1	1	0.5

The filter was initialized using the method described in Section 4.1.6 by using an initial LIDAR-scan and GNSS measurement. The resulting position error when

reversing in a figure eight is shown in Figure 6.17.

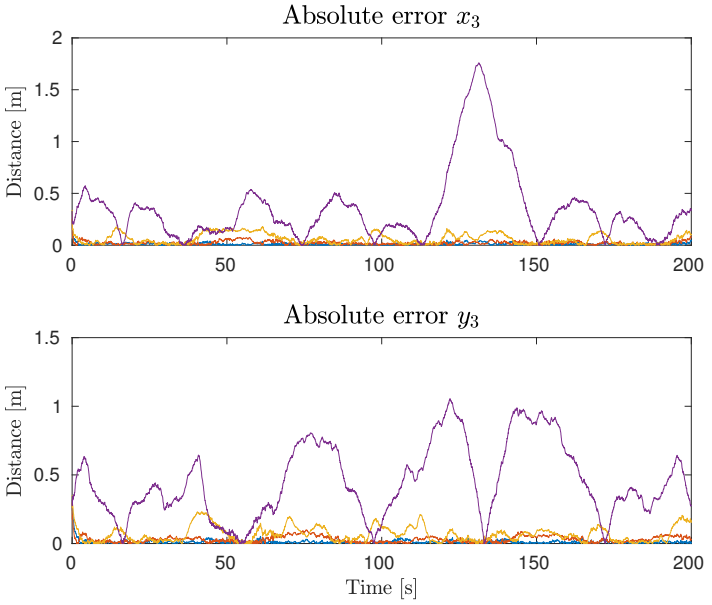


Figure 6.17: *Resulting absolute position estimate error when using the full vehicle dynamics together with measurements from a GNSS and RANSAC,NN3, Camera point detection and CNN.*

The result shows that the estimation errors of β_2 and β_3 from the CNN is translated to a comparably severe error in position estimates. The other approaches seem to give reasonable position estimates.

6.3.3 Parameter Estimation

L_2,L_3 and L_k were estimated by augmenting the lengths to the states, allowing them to be changed over time. As was argued in Section 2.6, GNSS measurements tend to make the L_3 drift since it is impossible to distinguish changes in x_3 and y_3 from a change in L_3 solely based on GNSS measurements of the truck's position. Instead, L_3 was estimated by using the interdependence between L_3 and the dynamics of β_3 in (2.3d). Hence only the angle dynamics (2.3d) and (2.3e) were used when estimating L_2,L_3 and L_k.

The end-to-end approaches for generating measurements to the filter are not applicable when L_2 and L_k are to be estimated since direct measurements of β_3 and β_2 contain no information about L_2 and L_k. Hence only the measurements from the RANSAC approach were used for the parameter estimation.

The process noises for L_2,L_3 and L_k were tuned by balancing the smoothness and speed of convergence of the parameter estimations. The final values decided upon were 10, 1 and 0.1 for L_2,L_3 and L_k respectively. The result of the parameter estimation, using the data set when the system reversed in an eight-shaped path, is shown in Figure 6.18. L_k was initialized with an error of 50%, 0.6 meters instead of the true length of 0.4 meters, L_2 was initialized with an error of 20%, 3.36 meters instead of the true length of 2.8 meters, and L_3 was initialized with an error of 10%, 7.26 meters instead of the true length of 6.6 meters.

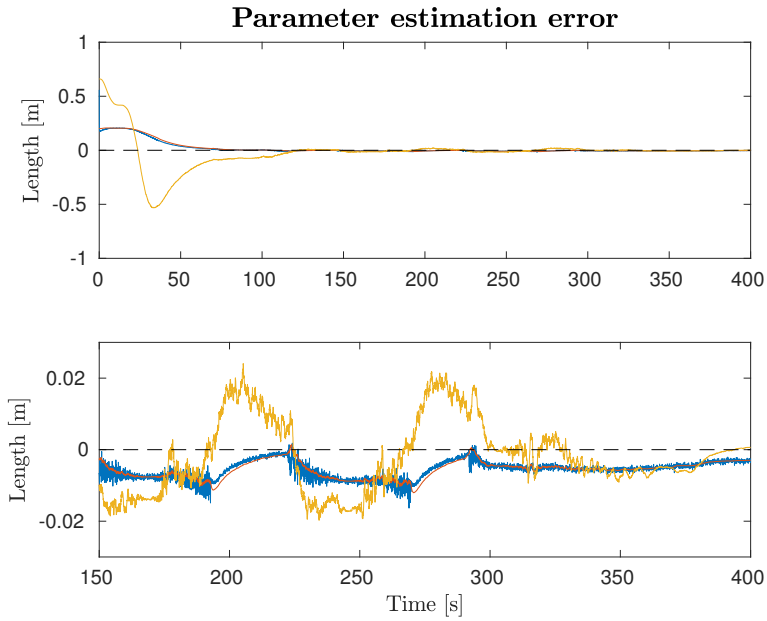


Figure 6.18: *Resulting estimation error of L_2, L_3 and L_k when running the augmented EKF using the angle dynamics and measurements from RANSAC. The bottom picture is a zoomed in version of the upper picture.*

In Figure 6.18 the parameters fluctuate between very accurate estimations and a small bias when they have converged to a reasonable length. The high accuracy is obtained when two sides of the trailer are hit by the LIDAR, leading to a very accurate estimation of the middle point of the trailer's short side. This suggests that the truck should be driven in a way which exposes two sides of the trailer when L_2,L_3 and L_k are to be calibrated.

A filter without the state augmentation ran alongside the augmented filter to compare the performances. Figure 6.19 shows the estimation error for the filter using the erroneous lengths and the filter estimating the parameters online.

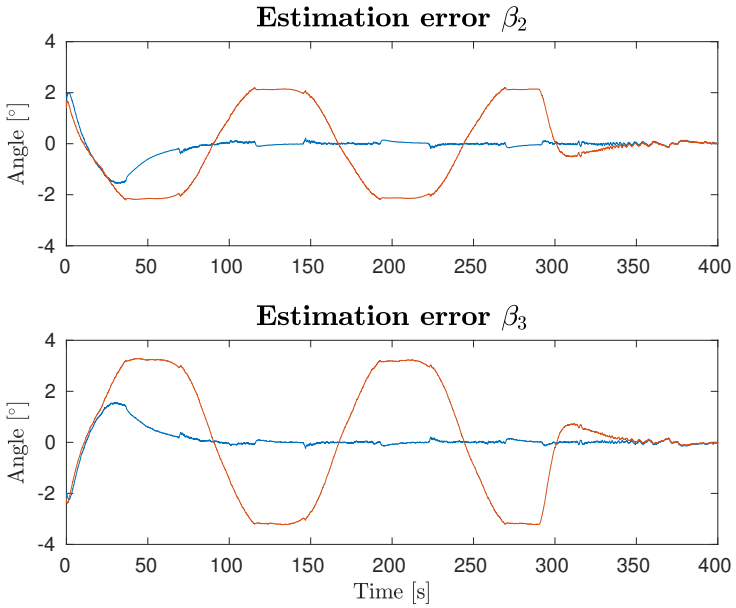


Figure 6.19: *Estimation error of β_2 and $\beta 3$ when using the augmented EKF and a plain EKF using the angle dynamics and measurements from RANSAC.*

The results show that the augmented filter is able to estimate the parameters with good accuracy which produces a significant increase in the accuracy of the state estimates.

7
Conclusions

7.1 Conclusion

State estimation for a truck and trailer system through sensors mounted on the truck has been investigated in this thesis work.

A simulation environment for the system has been developed which allows for collection of data to train deep neural network for end-to-end angle estimation from LIDAR scans and camera images. Multiple fully connected deep neural networks have been trained on the LIDAR data and a convolutional neural network has been trained on the camera images.

Model-based reference approaches which extract relevant information from LIDAR scans and camera images have also been implemented and evaluated in simulation and have been compared with the results from the neural networks. The model-based method for the LIDAR scan uses RANSAC to extract dominating lines in the point cloud and these are used to estimate the middle point of the trailer's short side and the angle between the truck and trailer, quantities which can be related to the system states through measurement equations. The model-based method for camera images has been based on detection of markers on the trailer in the images and these pixel positions have been related to the system states.

Measurements from the model-based approaches and from the deep neural networks have been used in EKFs together with a motion model for the system for state estimation. Also, an augmented EKF has been implemented to estimate the lengths of the dolly and the trailer to improve the performance of the observer and the controller.

Results in simulation have shown that the deep neural network approaches give decent angle estimation, with the network trained on LIDAR data being superior, but the model-based approaches were shown to be more accurate and more robust. In the case of a rectangularly shaped trailer, the model-based approach for the LIDAR-scans has shown to be the most accurate and has proven robust to protrusions on the trailer, making it the preferred method of choice in most practical cases. However, the flexibility of the deep neural network compared with the model-based approaches have been highlighted and the learning approach can be viable in situations where a model of the trailer is missing but lots of data is available.

The augmented EKF has shown to be able to estimate biases in lengths of the dolly and trailer in simulation and this has been shown to improve the state estimates.

7.2 Future work

The used neural network architectures give decent results but lots of improvements can potentially be made. Further exploration of suitable hyperparameters such as the number of hidden layers, number of neurons in each layer or number of convolutional layers and number of filters in each convolutional layer is needed. Also, as is always the case with deep learning, improvements can be achieved if the networks are allowed to be trained for longer duration and if more training data are used. The work done in this thesis should be seen as a proof of concept with lots of room for refinement.

A logical continuation, since all the evaluations and training have been done on simulated data, is to try out the approaches on real-world data and see if the same conclusions stand. Of special interest is to see what happens to the L_3 estimate since this is based on the motion model of the system, which might not be accurate enough in practice to give sufficient accuracy of L_3.

Another interesting situation for further investigation is to see how biases in the placement of the sensors, such as yaw-bias or a translation bias affect the estimates.

To train a neural network online during driving for angle estimation is also an area worth exploring. The neural networks that have been trained approximate static relationships between the sensor data and the angles, but one could also exploit the dynamics of the system by training recurrent neural networks.

Bibliography

[1] C. Altafini. The general n-trailer problem: conversion into chained form. In *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No.98CH36171)*, volume 3, pages 3129–3130 vol.3, 1998. doi: 10.1109/CDC.1998.757991. Cited on pages 2 and 7.

[2] C. Altafini, A. Speranzon, and K. H. Johansson. Hybrid control of a truck and trailer vehicle. In *International Workshop on Hybrid Systems: Computation and Control*, pages 21–34. Springer, 2002. Cited on page 2.

[3] L. Beyer, A. Hermans, and B. Leibe. DROW: real-time deep learning based wheelchair detection in 2d range data. *CoRR*, abs/1603.02636, 2016. URL `http://arxiv.org/abs/1603.02636`. Cited on page 2.

[4] L. Caup, J. Salmen, I. Muharemovic, and S. Houben. Video-based trailer detection and articulation estimation. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 1179–1184, June 2013. doi: 10.1109/IVS.2013.6629626. Cited on page 3.

[5] C. de Saxe and D. Cebon. A visual template-matching method for articulation angle measurement. In *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, pages 626–631, Sept 2015. doi: 10.1109/ITSC.2015.108. Cited on page 3.

[6] J. Deng, W. Dong, R. Socher, L. J. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009. doi: 10.1109/CVPR.2009.5206848. Cited on page 39.

[7] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981. ISSN 0001-0782. doi: 10.1145/358669.358692. URL `http://doi.acm.org/10.1145/358669.358692`. Cited on page 32.

[8] C. Fuchs, F. Neuhaus, and D. Paulus. Advanced 3-d trailer pose estimation for articulated vehicles. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 211–216, June 2015. doi: 10.1109/IVS.2015.7225688. Cited on page 3.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016. doi: 10.1109/CVPR.2016.90. Cited on page 21.

[10] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *International 1989 Joint Conference on Neural Networks*, pages 593–605 vol.1, 1989. doi: 10.1109/IJCNN.1989.118638. Cited on page 25.

[11] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. Cited on page 17.

[12] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960. Cited on page 27.

[13] A. Kendall, M. Grimes, and R. Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2938–2946, Dec 2015. doi: 10.1109/ICCV.2015.336. Cited on pages 2, 26, and 39.

[14] N. Koenigand and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149–2154. IEEE. Cited on page 35.

[15] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. Cited on page 21.

[16] B. Li, T. Zhang, and T. Xia. Vehicle detection from 3d lidar using fully convolutional network. *CoRR*, abs/1608.07916, 2016. URL `http://arxiv.org/abs/1608.07916`. Cited on page 2.

[17] O. Ljungqvist. Motion Planning and Stabilization for a Reversing Truck and Trailer System. Master's thesis, Linköping University, 2015. Cited on pages 2, 8, and 15.

[18] P. Nyberg. Stabilization, Sensor Fusion and Path Following for Autonomous Reversing of a Full-scale Truck and Trailer System. Master's thesis, Linköping University, 2016. Cited on pages 2 and 15.

[19] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1717–1724. IEEE, 2014. Cited on page 26.

[20] A. Petrovskayaand and S. Thrun. Model based vehicle detection and tracking for autonomous urban driving. *Autonomous Robots*, 26(2-3):123–139, 2009. Cited on page 2.

[21] Scania. Rear-view-cameras. URL `https://accessories.scania.com/en/catalog/Safety---Security/Camera-and-monitoring-systems/Rear-view-cameras`. Accessed 2018-05-22. Cited on page 6.

[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. Cited on page 25.

[23] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. URL `http://arxiv.org/abs/1409.4842`. Cited on pages 21 and 39.