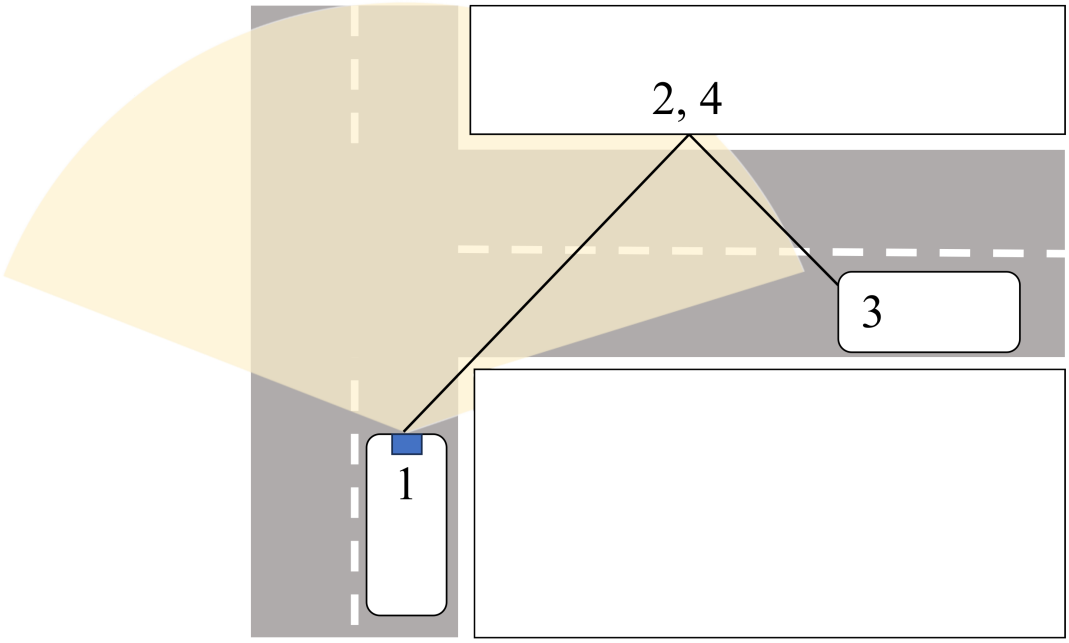


Non-line-of-sight object localization using multipath wave propagation - a machine learning approach

Master's thesis in Complex Adaptive Systems

ELLINOR LUNDBLAD, ISAK WALDENER

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

MASTER'S THESIS 2023

Non-line-of-sight object localization using multipath wave propagation - a machine learning approach

ELLINOR LUNDBLAD, ISAK WALDENER

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023

Non-line-of-sight object localization using multipath wave propagation - a machine learning approach

ELLINOR LUNDBLAD, ISAK WALDENER

Supervisor: Niclas Carlström, Aptiv
Examiner: Tomas McKelvey, Electrical Engineering

Master's Thesis 2023
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Figure of a NLOS scenario including a host car with a mounted radar and a NLOS target.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Non-line-of-sight object localization using multipath wave propagation - a machine learning approach
ELLINOR LUNDBLAD, ISAK WALDENER
Department of Electrical Engineering
Chalmers University of Technology

Abstract

This thesis report examines the application of machine learning models for detecting and positioning of non-line-of-sight (NLOS) and line-of-sight (LOS) targets using multi-path wave propagation. The report commences with exploring the data simulation process, emphasizing the distribution of scenarios and the incorporation of NLOS timeframes. The simulation aims to include variations in the dataset, enabling the models to understand the problem comprehensively. Next, the report focuses on machine-learning positioning, comparing the performance of two models: the U-Net and a Convolutional Neural Network (CNN). Both models achieve similar accuracy, but U-Net outperforms CNN in terms of mean squared error (MSE) and average Euclidean distance (AED). The report evaluates the models' performance on different scenario labels, scenario types, and numbers of targets. Finally, the report examines object tracking using machine learning measurements. The tracking algorithm demonstrates high performance, achieving low average Euclidean distance (AED) and absolute errors in position, velocity, heading, and turn-rate state estimation. The report concludes that machine learning models show promise in identifying and positioning NLOS and LOS targets, and object tracking using machine learning measurements yields satisfactory results.

Keywords: radar, multi-path, NLOS, U-Net, CNN, machine-learning, object tracking, multi-object

Acknowledgements

We want to thank Aptiv for their support and resources throughout this project. A special thanks to our supervisor, **Niclas Carlström**, whose expertise and guidance have been invaluable during the project.

Ellinor Lundblad Isak Waldener, Gothenburg, June 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AED	Average Euclidean Distance
CNN	Convolutional Neural Network
LOS	Line-of-sight
ML	Machine Learning
MSE	Mean Square Error
NLOS	Non-line-of-sight
ReLU	Rectified Linear Unit

Contents

List of Figures

List of Tables

1

Introduction

1.1 Introduction

1.1.1 Background

The Swedish "Nollvisionen" - the Zero Plan- aims to reduce fatal traffic incidents to zero. Last year, 2022, marked the 25th year since "Nollvisionen" was introduced. By October 2022, 178 people had died in traffic accidents in Sweden that year [1]. Sweden has among the safest roads in the world [2], but if we want to achieve our goal, we still have work to do.

Many automotive companies and researchers are working to increase the safety of road users and motorists. They develop driver-assisting technologies such as lane departure warning systems, adaptive cruise control, and emergency braking to ensure safer roads [3]. Aptiv is one of the companies in the industry of enabling safer driving. They utilize many technologies, including radar.

Radar is one of the most common technologies to detect objects and potential hazards in the driving environment. It has several upsides; unlike many other vision systems, such as cameras, its physical characteristics allow it to work in different weather conditions, including snow, rain, and fog [3].

Another potential upside of radar lies in the very core of its characteristics, the fact that it reflects on surfaces. We want to use this attribute in our project. While cameras or humans cannot detect objects around street corners or behind other cars, the reflecting nature of radar allows it to "see" around corners or cars, potentially discovering hidden objects and counteracting collisions. These types of radar detections are called multi-path detections since they have more than one reflection. The points of reflection are generally unknown, creating a setting where infinitely many solutions are possible. Therefore, finding the position of an object detected by multi-path radar is difficult. As a result, multi-path radar detections are generally unused, considering only targets in the Line-Of-Sight (LOS) of the radar [4].

Recent research attempts Non-Line-Of-Sight (NLOS) object detection and positioning in simulated and real-world scenarios with varying results. The most common approach is to use classical optimization algorithms, primarily nonlinear least squares and maxi-

mum likelihood [5]. In [6], an improvement of the maximum likelihood estimator using Taylor-series expansion-based linear quadratic programming is introduced and tested in simulations. The maximum likelihood approach is applied in real-world scenarios in [4]. Attempts to apply machine learning algorithms in NLOS object-tracking scenarios have been made. Several studies, using either real-world or simulated data, have applied machine learning models to position and describe objects in the NLOS scenario with promising results [7, 5, 8].

The attempts to detect NLOS objects have thus far been limited to single object detection. This report explores the detection of NLOS targets in a multi-object setting. This project aims to build on the studies made in the field by testing different machine learning methods for NLOS object detection in a multi-target setting. The project is done with Aptiv and use their radar technology and expertise.

1.1.2 Preliminary aim

The project aims to build a machine learning model to detect and track non-line-of-sight (NLOS) and line-of-sight (LOS) objects in risk scenarios using radar multi-path wave propagation.

1.1.3 Demarcations

The project is limited to automotive settings, with a single front radar sensor mounted on a host car.

The project will mainly focus on simulating data and the results of the machine learning model on simulated data. The simulated dataset will consist of scenarios in which multi-path radar detections indicating the NLOS object can occur.

The data will consist of scenarios with 19 timesteps of 0.2 seconds, resulting in a total of 4 seconds per sequence. The thesis will only handle sequences of the same lengths.

The machine learning models and the dataset will be limited in size due to limited computational power.

1.1.4 Problem formulation

The project will explore the application of machine learning methods in a non-line-of-sight (NLOS) radar tracking scenario. The central part of the project will consist of a data simulation and a machine learning application on the simulated data, concluding with a multi-target Kalman-based tracker of object position, velocity, heading, and turn rate using the machine learning outputs. The main points to examine are the following:

- How can we simulate a large and balanced data set covering a range of scenarios with NLOS objects where multipath-radar detection can occur?
- How well does a machine learning model, trained to detect scenarios where NLOS objects are present, perform?
- How can we develop and optimize a machine learning model to achieve high performance on the simulated dataset? With radar detection positions as input, the model is expected to generate accurate object positions as output.
- How well do the machine learning models perform in a multi-target setting? How does this compare to a single-target setting?
- Does the performance vary for the different simulated scenarios? Why?
- How can the machine learning outputs be used in an algorithm to track objects' position, velocity, heading, and turn rate over time?

2

Theory

2.1 Radar fundamentals

Initially an acronym for RAdio Detecting and Ranging, RADAR has become a universally recognized term due to its widespread use in modern technology[9].

Radar systems typically include a transmitter, an antenna, a receiver, and a signal processor. The transmitter generates electromagnetic waves, which are emitted by the antenna [10]. The waves "reflect" on surfaces and are captured by the receiver upon their return. After that, they are processed by the signal processor. In this thesis, the radar is mounted on a car called the host.

The reflected signals are used to identify objects of interest near our radar. By measuring the time it takes for the waves to return to the sensor, the system can determine the distance to the object. The frequency shift of the reflected waves can be analyzed based on the Doppler Effect to determine the targets' velocity [11]. This analysis is used to determine the range and range rate (radial velocity). The azimuth angle of the target is determined by the pointing angle of the antenna's main beam, or by use of a phased array antenna [10]. These values are typically presented relative to the mounted radar and need ego-motion compensation 2.1.1. The range r and azimuth angle θ can be seen in Figure 2.1.

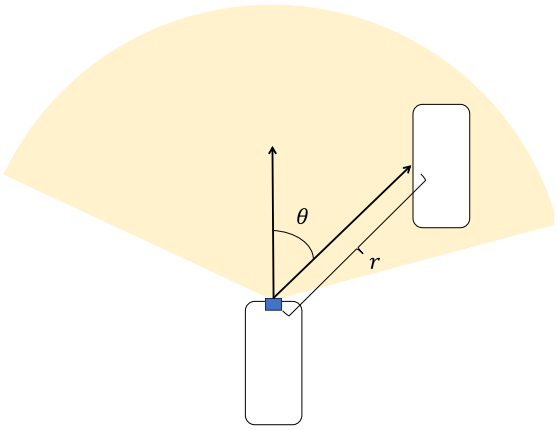


Figure 2.1: Figure of the host car with front-facing center mounted radar in blue, with range denoted r and azimuth angle θ relative host

2.1.1 Ego-motion compensation

The ego-motion compensation allows us to separate the movement of the host (the ego-motion) from the radar measurements. The range rate is corrected to account for the motion of the vehicle. The ego-motion compensation of the range-rate is described by the equations (1) and (2).

The ego-motion induced radial velocity, v_r, is calculated as in the equation (1),

$$v_r = -|\vec{v}|cos(\theta), \tag{1}$$

where θ is the azimuth angle and $\vec{v} = [v_x, v_y]^T$ is the radar sensor velocity in x- and y-directions relative to ground [12]. Compensation of the range rate $\dot{r}$, given the ego-motion induced radial velocity v_r, is described in Equation (2).

$$\dot{r} = \dot{r} - v_r \tag{2}$$

The radar data is very noisy by nature and requires a lot of signal processing. Noise sources in the radar data include;

- background returns; reflections on non-target objects
- electromagnetic interference,
- and electronic noise [10].

In this thesis, the idea is to utilize reflections on non-target objects, to get information about target objects. The electromagnetic waves reflect on a different object before and/or after reflecting on the target object. This is described as multi-path radar detecting and is described further in Section 2.1.2.

2.1.2 Multi-path scenarios

In Figure 2.2, a host car with a mounted radar in blue, labeled (a) detects a target object labeled (b) through direct or multi-path radar. The possible paths include:

1. Direct path radar: a-b-a (host-target-host).
2. Multi-path radar, a-c-b-a (host-object-target-host)
3. Multi-path radar, a-b-c-a (host-target-object-host)
4. Multi-path radar, a-c-b-c-a (host-object-target-object-host)

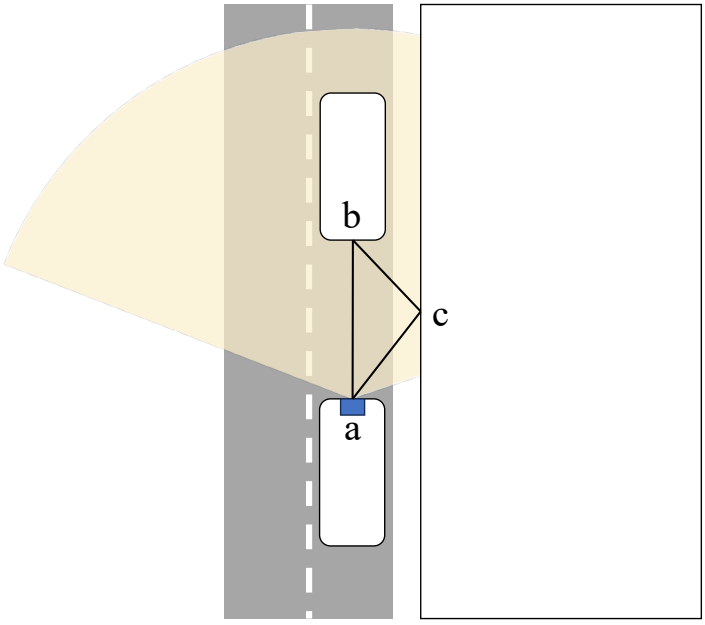


Figure 2.2: Figure over different radar paths from host (marked with blue mounted radar) and the target car

In cases two and three, the object commonly has detections at the object's surface and can be visible through single-path detections. This is the most common source of radar multi-path [10]. The last case, four, is more complicated and relevant to the project. It includes more advanced cases, for example, the one described in Figure 2.3.

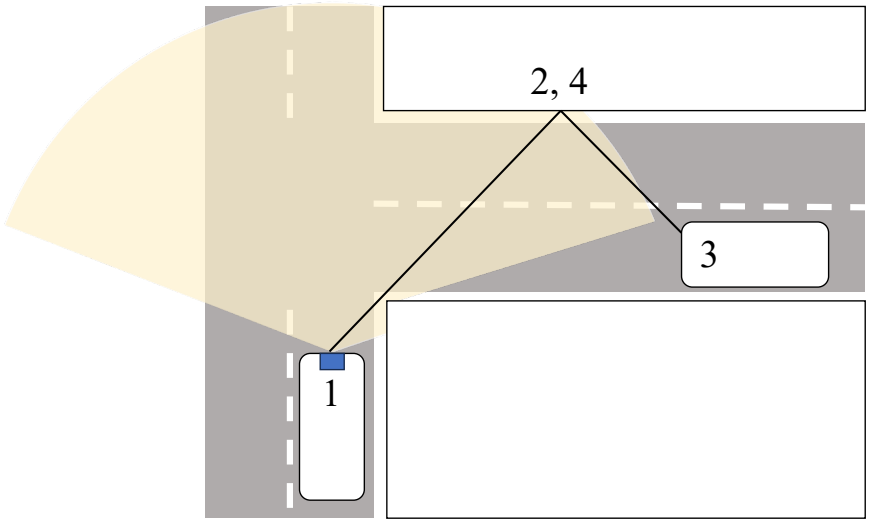


Figure 2.3: Advanced case of host-object-target-object-host multi-path detection

In this report, Non-line-of-sight (NLOS) refers to objects without direct path detections. If an object is considered in the line-of-sight (LOS), this does not mean it has no multi-path detections; it simply means that it has direct detections.

2.1.3 MATLAB radar simulation

A simulation environment can be controlled to create scenarios where multi-path propagation can occur. We need specific circumstances for these reflections to happen, and they are simpler to achieve in a controlled simulated environment.

This thesis uses the MATLAB Automated Driving Toolbox [13] to simulate scenarios combined with the MATLAB Radar Toolbox to simulate radar [14]. The MATLAB Radar Toolbox outputs detection-level radar data. The detections from Radar Toolbox are reported in the host vehicles coordinate system. The coordinate system used is the ISO 8855, which has the x-axis pointing forward from the car and the y-axis to the left [14].

The simulation takes object positions and velocities as input. This input works as ground truth for our machine learning models.

2.2 Machine learning

A machine learning algorithm is an iterative process wherein a computational model, such as an Artificial Neural Network or a linear model, is trained to effectively represent a particular dataset, targeting a specific objective, without requiring explicit programming [15]. Through training, the model progressively learns to capture the dataset's underlying patterns by adjusting its parameters in response to new data instances.

The trainable parameters in artificial neural networks generally include weights and biases. In the figure 2.4 is a simple artificial neural network . This model is a fully connected feed-forward neural network, as every neuron in a given layer connects to each neuron in the following layer. These types of layers are also referred to as Dense layers.

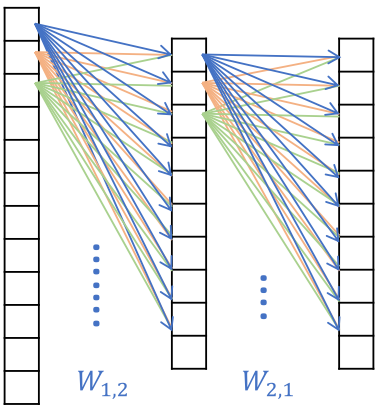


Figure 2.4: Figure of a simple feed-forward network with 1 hidden layer.

The output from each neuron is multiplied by a weight w, and a bias b is added as described in Equation (4). This value is then propagated through an activation function [16].

Given the notation in (3),

$$h^{Layer}_{number}, w^{Layer}_{n\,in,n\,out}, b^{Layer}_{number} \tag{3}$$

The hidden layer neurons for layer 2, $\vec{h}^2 = [h^2_1 \quad h^2_2 \quad h^2_3]$ are defined by the matrix-vector equation in Equation (4), where w are the weights, and b the biases [16].

$$\begin{bmatrix} h^2_1 & h^2_2 & h^2_3 \end{bmatrix} = \begin{bmatrix} h^1_1 & h^1_2 \end{bmatrix} \begin{bmatrix} w^1_{1,1} & w^1_{1,2} & w^1_{1,3} \\ w^1_{2,1} & w^1_{2,2} & w^1_{2,3} \end{bmatrix} + \begin{bmatrix} b^1_1 & b^1_2 & b^1_3 \end{bmatrix} \tag{4}$$

And in general, for layer l, the matrix equation becomes (Equation (5)).

$$\vec{h}^l W^l + \vec{b}^l = \vec{h}^{l+1}, \tag{5}$$

If the layer includes an activation function g the equation becomes (Equation (6)),

$$\vec{h}^{l+1} = g(\vec{h}^l W^l + \vec{b}^l) \tag{6}$$

Without the activation function g, these simple types of neural networks are essentially matrix multiplications and can only capture linear relationships. With the activation function, neural networks become expressive and can capture non-linear relationships [16]. Activation functions are further covered in Section 2.2.7.

2.2.1 Convolutional neural networks

One specific Neural Network is the Convolutional Neural Network (CNN). CNNs are most successfully used on data with spatial dependencies as, for example, image data. CNNs use alternating convolutional and pooling layers as a key component.[16]. The layers are described in Figure 2.5, as well as in more detail in Figure 2.7 and Figure 2.8.

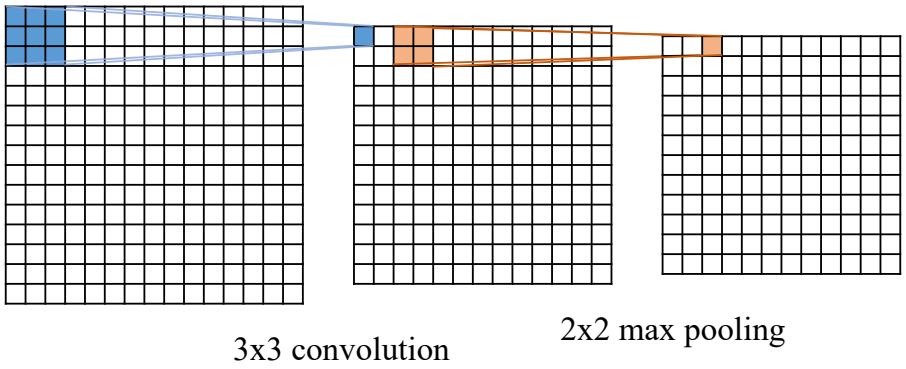


Figure 2.5: Figure of a 3x3 convolution and 2x2 max-pooling of an input

The convolution and max-pooling filters pan over the image as seen in Figure 2.6. The image shows filters with size 3x3 and step size 1. The filter moves over all columns for each row in turn. The same operation is applied for each 3x3 input. Larger step sizes and padding (added zeros around the input shape) affect the output and its shape.

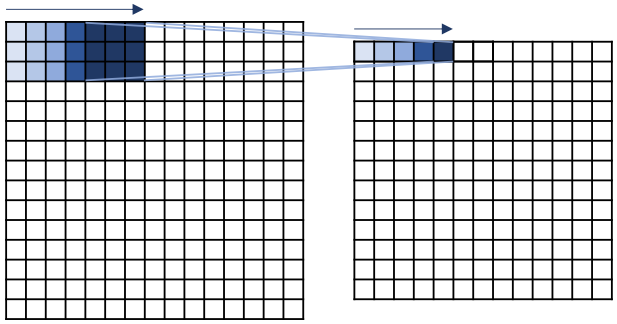


Figure 2.6: Figure showing how a 3x3 filter pans over the input image with step size 1 and padding 0.

Convolutional layers consist of one or more filters that pan over the image input, as shown in Figure 2.6. The convolutional step is described in Figure 2.7, where the 3x3 input in blue is passed through the 3x3 filter in grey, resulting in the 1x1 output in blue.

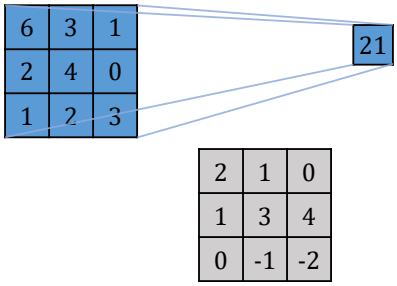


Figure 2.7: Figure of one step in convolution with a 3x3 input in blue, a 3x3 filter in grey, and a 1x1 output in blue. This step is iterated over the entire input image.

The filter in Figure 2.7 contains 9 individual weights and a bias term, b, which is added as described in Equation (7) Define inputs, filter weights, and biases as x_{ij}, f_{ij}, and b, respectively, where i determines the row number and j is the column number. Given an activation function g, one convolution step can be described as in Equation (7) [16].

$$Output = g\left(\sum_{i}^{N}\sum_{j}^{N} f_{ij}x_{ij} - b\right) \tag{7}$$

This step is iterated over the entire image, as described in figure 2.6.

Convolutional layers are commonly followed by max-pooling layers, as described in Figure 2.8. The maximum value in the 2x2 input cell is used as output.

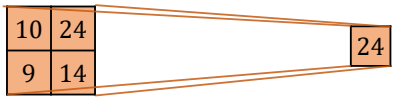


Figure 2.8: Figure of a 2x2 max-pooling with the input in the 2x2 grid to the left, and the output in a 1x1 square to the right.

A model with alternating convolutional and pooling layers performs well on pattern recognition tasks since they implement parameter-sharing [16]. Each filter pans over the input and applies the same weights on all inputs.

2.2.2 U-Net

The U-Net is a specific CNN commonly used in image segmentation and classification. The model is an encoder-decoder model and consists of two parts. The first part is the encoder, which converts the input to a smaller dimension and extracts important features from the data. The second part is the decoder, which converts the encoded input back to the input dimension [17].

The encoder and decoder parts of the model are built up by blocks for downsampling or upsampling, respectively. Each downsampling block has an increasing number of filters in the convolutional layers [17]. The upsampling blocks work in the opposite way, with a decreasing number of filters for each block. An overview of the U-Net model is seen in Figure 2.9. In Appendix A.1, a more detailed version of the U-Net architecture is available.

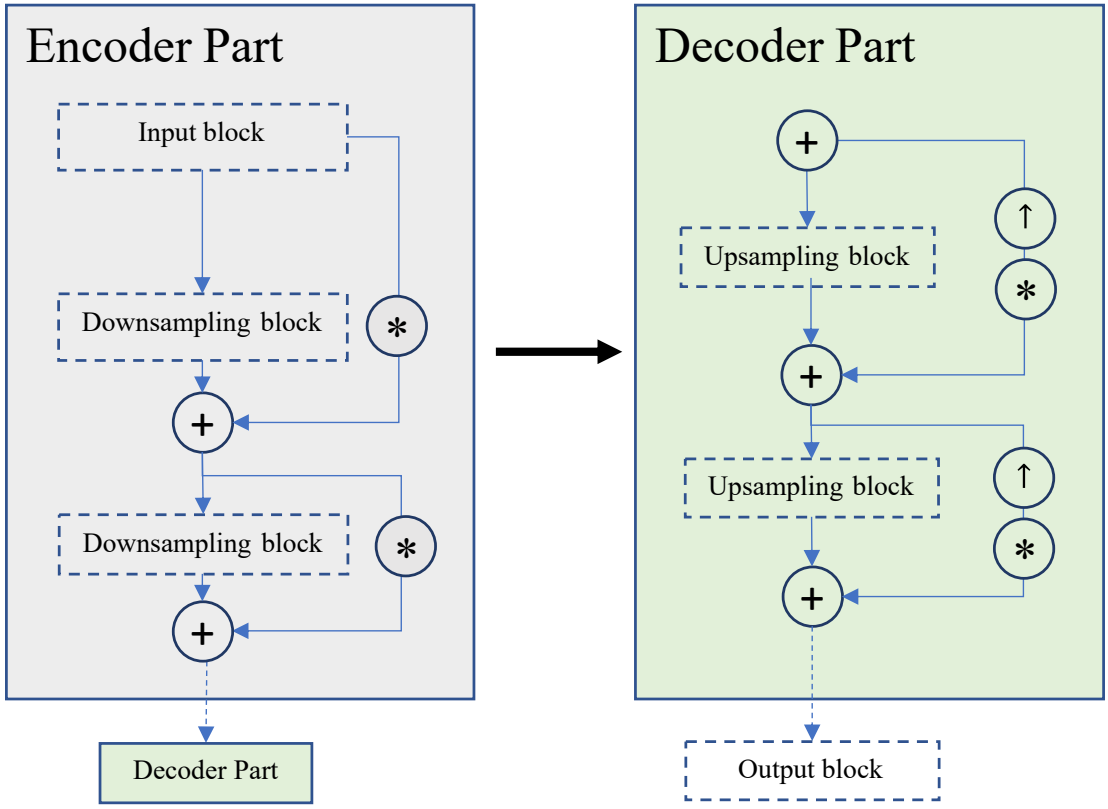


Figure 2.9: An overview of the U-Net model architecture. The $*$ indicates a convolution step, the $\uparrow$ indicates an upsampling, and the $+$ indicates a matrix addition. Multiple layers of downsampling and upsampling blocks can be used, indicated by the dotted arrow at the bottom of encoders and decorders.

In the model in this project, the downsampling blocks are built using separable convo-

lutional layers (Section 2.2.3) and max-pooling layers (Section 2.2.1). The max-pooling layers are used to downsample and extract important features. The upsampling blocks use transposed convolutional layers (Section 2.2.4). The transposed convolutional layers are used for upsampling and generating a feature map greater than the input.

The model was initially successfully used in biomedical image segmentation [17]. The U-Net model has been used to process radar images in multiple reports [18, 19, 20] with good results.

2.2.3 Seperable Convolutional layer

The separable convolution layer consists of two steps: a depthwise spatial convolution, then a pointwise convolution [21]. The depthwise convolution performs convolution with one filter for each depth layer. For a 3-depth input, three filters are used. The output from the convolution is then used for the pointwise convolution. For pointwise convolution, a 1x1 filter is convoluted over all input channels and each point in the input [21].

2.2.4 Transpose Convolutional layer

A transpose convolutional layer is used for upsampling data by doing convolution on a modified input. The layer works by adding extra zeros to the output to increase the dimension of the input [22]. After the input is increased, convolution is applied to the new input. This results in an output feature map of greater dimension greater than the input. The steps of a transposed convolution are displayed in Figure 2.10.

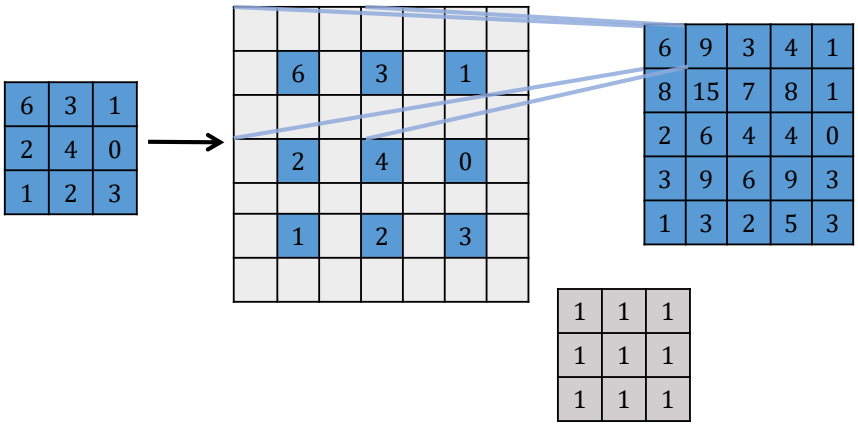


Figure 2.10: Figure over transposed convolution, with 3x3 input in blue, which is padded into a 7x7 grid, applied a 3x3 filter (in grey) convolution with step-size 1, resulting in the blue 5x5 output.

2.2.5 Dropout layer

Dropout layers are used in models to avoid overfitting, which is described in Section 2.3.2. During training, neurons in the layer are set to zero with a given probability P. This is done to prevent interdependence between neurons in a layer and is a way to prevent overfitting [16].

2.2.6 Batch normalization

Batch normalization is used in models to accelerate and make the training more stable. This is accomplished by normalizing the layer inputs. The normalization fixes the input mean values and variances, making it possible to use higher learning rates [23].

2.2.7 Activation functions

Activation functions are used in neural network layers to map the output from the layer to a given function. They are used in the networks to capture non-linear relationships [16].

2.2.7.1 ReLU

The Rectified Linear Unit (ReLU) function is commonly used as an activation function in neural networks [16]. The function returns the passed value, x, if positive, and 0 if it is negative. In Equation (8) below, the ReLU function can be seen.

$$f(x) = max(0, x) \tag{8}$$

2.2.7.2 Sigmoid

The Sigmoid function is commonly used as an activation function for the output layer in binary classification networks. The reason is that the functions convert the layer output to a value between 0 and 1, which means the output can be interpreted as probabilities [16]. The sigmoid functions can be seen in Equation (9) below.

$$f(x) = \frac{1}{1 + exp(-x)} \tag{9}$$

2.3 Model training

Before training the model, the dataset is randomly split into training, test, and validation datasets [24]. The training dataset is used in the model training, the validation dataset is used in hyper-parameter tuning and early stopping (Section 2.3.2), and the test set is used in the final evaluation (Section 2.5). The test set typically includes the larger part of the data.

In machine learning, models are optimized to fit a training dataset. The training typically occurs over a large number of epochs. In each epoch, the training algorithm processes the complete training dataset, wherein the model parameters undergo updates aimed at minimizing a specific loss function relative to the training data. This is done using an optimizer, typically a version of stochastic gradient descent [16].

Gradient descent uses the gradient of the loss function with respect to each parameter to decide the parameter's new value. The parameter, θ is updated according to Equation (10), where η is the learning rate and $\frac{\delta\theta}{\delta L}$ is the gradient of θ with respect to the loss function L [15].

$$\theta' = \theta - \eta\frac{\delta L}{\delta\theta} \tag{10}$$

The learning rate, η, decides how much a training sample affects the model parameters. Stochastic gradient descent is a version of gradient descent where random samples of the training data are used to compute the loss instead of the entire dataset. The Adam optimizer is an adaptable version of the stochastic gradient descent [25].

In supervised learning, the loss function compares model outputs and expected targets, increasing as differences between them grow. [15]. The training is an optimization with respect to this loss function.

This project's positioning machine learning models are applied to a version of a classification problem. While the output, in the end, is interpreted as positions in an x-y grid, it is actually represented as probabilities (values between 0 and 1) of positions on the grid. The loss function is, therefore, one that is commonly applied to classification problems - Binary Cross-Entropy (also known as log-loss), presented in Equation (11) [26].

The classification models have only a single output that is correct, making the Categorical Cross-Entropy more suitable. The Categorical Cross-Entropy loss is defined in (12).

2.3.1 Loss functions

The loss functions used to optimize the machine learning models are Binary Cross-Entropy and Categorical Cross-Entropy, as presented in this section.

2.3.1.1 Binary Cross-Entropy

Binary Cross-Entropy (log-loss) is a loss function used for binary classification problems. The loss gets affected for targets 0 and 1, which makes it work well with Sigmoid activation that outputs values between 0 and 1. The function can be used for multi-label classification since it calculates the loss for each neuron individually. The function is presented below in Equation (11) [26].

$$L(\vec{x}, \vec{y}) = -\frac{1}{M} \sum_{m=1}^{M} \left[y_m \, log(h_\theta(x_m)) + (1 - y_m) \, log(1 - h_\theta(x_m)) \right] \tag{11}$$

Where M is the number of samples, y_m is the target and x_m is the input value for each sample, and h_θ is the model with parameter set θ.

2.3.1.2 Categorical Cross-Entropy

Categorical Cross-Entropy is a loss function used for multi-label classification problems with only one correct class. The loss function is a version of Cross-Entropy loss for when the output is one hot encoded. The loss function is shown in Equation (12).

$$L(\vec{x}, \vec{y}) = -\frac{1}{M} \sum_{k=1}^{K} \sum_{m=1}^{M} y_m^k \, log(h_\theta(x_m, k)) \tag{12}$$

In the equation, K is the number of different classes in the output. M is the number of samples, y_m^k is the target for class k and x_m is the input value, and h_θ is the model with parameter set θ [26].

2.3.2 Overfitting

As described in Section 2.3, the model weights and parameters are adjusted to the training data and the selected loss function. As the model is introduced to the data multiple times, its weights are tuned to the specific problem. Overfitting occurs when the weights are adapted to the training set so that the model loses generalizability and corresponds to a performance loss on the test set [24].

The validation dataset can be used to prevent overfitting by introducing early stopping [24]. When the performance on the validation stops increasing or decreases for a given number of epochs, we consider the optimal weights found and stop model training [24].

Moreover, the validation dataset can be used to tune hyperparameters, such as the learning rate and the number of epochs. These adjust how much the model parameters (weights and biases) are affected by new samples and training time, respectively [16]. Another dataset, the test set, is used to evaluate final performance. This is done to increase the generalizability of the model.

2.4 Object tracking

Multi-target tracking complicates the multi-path problem and requires an algorithm to keep track of the objects. The chart in Figure 2.11 describes a common architecture used in object tracking in automotive settings [27]. The pipeline handles objects and their states over time, creating new and killing old objects as new observations are made. The gating, data association, track management, and state estimation steps are described below. These are necessary steps to track multiple objects.

Figure 2.11: Object tracking pipeline including gating, data association, track management, and state estimation.

2.4.0.1 Gating

The radar sensor samples the environment with a given frequency. At each new time step, new measurements are used to improve the representation of the environment. The

measurements are noisy, and a gating step is used to discard irrelevant or erroneous measurements. For example, very unlikely measurements given prior knowledge or known environmental constraints can be removed.

2.4.0.2 Data Association

At each new time step, the measurements are associated with existing objects. There are various ways to associate measurements. In this project, the Manhattan distance between the measurement and the track position is used. If (x_1, y_1) and (x_2, y_2) are two points, the Manhattan distance between them is described as in Equation (13).

$$\text{Manhattan Distance} = |x_1 - x_2| + |y_1 - y_2|. \tag{13}$$

If detections are within a specific distance of the track, they are associated with that target.

2.4.0.3 Track Management

Track management includes killing and creating objects and sometimes splitting and merging. Objects with no associated measurements are deemed irrelevant and removed/killed, and measurements with no associated objects can be used to create new objects. In complex environments, splitting and merging objects are necessary steps, but are deemed unnecessary for the simulated scenarios in this thesis.

2.4.0.4 State Estimation

The measurements associated with each individual track are used to update their estimated state. The tracks are commonly updated using a motion model and Kalman filtering, as described in Section 2.4.1.

2.4.1 Kalman filtering and motion model

Kalman filtering is an optimal estimation algorithm often used in object tracking. A Kalman filter algorithm is based on two main steps; prediction and update. The prediction step utilizes a model that, given prior knowledge, can predict the next step. In our case, the underlying model is a motion model that can predict an object's next position, velocity, heading, and turn rate, given the previous values of these states. The automatic weighting of the measurement and model prediction makes the Kalman filter powerful. It uses the probability distributions of the measurements and model predictions to estimate the actual state.

2.4.1.1 Update step

The Kalman state update consists of three steps formulated below as equations; the state update equation (15), the covariance update equation (17), and the Kalman gain calculation (20). The state update equation is described in simple terms as in Equation (14)

$$\begin{aligned}\textit{Current state estimate} =& \textit{Prediction of the current state} \\ &+ \textit{Factor}\,(\textit{Measurement} - \textit{Predicted value of the current state}),\end{aligned} \tag{14}$$

where the factor is commonly referred to as the Kalman Gain, K, and the ($\textit{Measurement} - \textit{Predicted value of the current state}$) is called the innovation. In matrix notation, the state update equation becomes,

$$\hat{\vec{x}}_{k|k} = \hat{\vec{x}}_{k|k-1} + K_k \vec{v}_k, \tag{15}$$

where $\vec{x}_{k|k}$ is the updated state,

$$\vec{v}_k = \vec{z}_k - H\hat{\vec{x}}_{k|k-1} \tag{16}$$

is the innovation, K_k is the Kalman gain, and H is the observation matrix given in Equation (26). z_k is the measurement [28].

The covariance update equation is given by Equation (17),

$$P_{k|k} = P_{k|k-1} - K_k S_k K_k^T, \tag{17}$$

where $P_{k|k}$ is the covariance matrix of the current state estimation, and S_k is defined by Equation (18),

$$S_k = H P_{k|k-1} H^T + R, \tag{18}$$

where H is the observation matrix, and R is the measurement noise covariance matrix [28].

The Kalman gain equation is described by Equation (19),

$$\textit{Kalman gain} = \frac{\textit{Variance in estimate}}{\textit{Variance in estimate } + \textit{ Variance in measurement}}, \tag{19}$$

and in matrix notation by Equation (20).

$$K_k = P_{k|k-1} H^T S_k^{-1}. \tag{20}$$

2.4.1.2 Prediction step

The state prediction step consists of two parts presented as equations; the state prediction (21) and covariance extrapolation (22). In our case, the underlying model is non-linear (24). This means the equations will need linearization. Below, the prediction step equations of an Extended Kalman Filter are presented, which utilizes first-degree linearizations in the prediction step [29].

The state prediction is described by Equation (21).

$$\hat{x}_{k|k-1} = f(\hat{x}_{k|k-1}) \tag{21}$$

The covariance extrapolation is described by Equation (22).

$$P_{k|k-1} = f'(\hat{x}_{k_1|k-1}) P_{k_1|k-1} f'(\hat{x}_{k|k-1})^T + Q \tag{22}$$

Where Q_k is the process noise matrix, and the function f is defined by the underlying model, in this project, a discretized constant turn rate and velocity (CTRV) model, described by Equation (25). The state, x_k for time step k is described by the vector in Equation (23),

$$\vec{x}_k = \begin{pmatrix} x_k & y_k & v_k & \theta_k & \omega_k \end{pmatrix}^T, \tag{23}$$

where x_k, y_k are the absolute x- and y-positions in meters, v_k is the velocity norm in m/s relative to ground, θ_k is the heading in radians, and ω_k is the turn rate in radians/s.

The prediction and update steps are iterated over time as new measurements are introduced, as pictured in Figure 2.12. At each new timestep, a filter prediction is made [28]. A measurement update is made if there is a measurement at the new time step. If there is no measurement at that time step, only the prediction step is done.

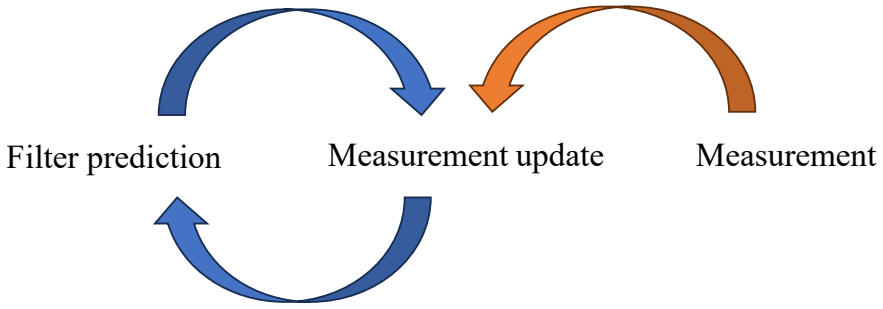


Figure 2.12: Figure of the prediction and update steps for the Kalman filter

2.4.1.3 Motion model

Equation (24) describes a simplified version of the CTRV model in [30] for timesteps of size T.

$$\vec{x}_k = \vec{x}_{k-1} + T \begin{bmatrix} v_{k-1} cos(\theta_{k-1}) \\ v_{k-1} sin(\theta_{k-1}) \\ 0 \\ \omega_{k-1} \\ 0 \end{bmatrix} \tag{24}$$

Equation (24) results in the expression for $f(\hat{x}_{k|k-1})$ in Equation (25).

$$f(\hat{x}_{k|k-1}) = \begin{bmatrix} x_{k-1} + T v_{k-1} cos(\theta_{k-1}) \\ y_{k-1} + T v_{k-1} sin(\theta_{k-1}) \\ v_{k-1} \\ \theta_{k-1} + T\omega_{k-1} \\ \omega_{k-1} \end{bmatrix} \tag{25}$$

2.4.1.4 Measurements and observation matrix

In this thesis, the measurements $z_k = [x_z,\ y_z]^T$ that the Kalman filter receives positions in x- and y-coordinates. This gives an observation matrix, H described in Equation (26).

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \tag{26}$$

The innovation equation (16) becomes

$$\vec{v}_k = \vec{z}_k - H_k \hat{\vec{x}}_{k|k-1} = \begin{bmatrix} x_z \\ y_z \end{bmatrix} - \begin{bmatrix} \hat{x}_{k|k-1} \\ \hat{y}_{k|k-1} \end{bmatrix}. \tag{27}$$

2.4.1.5 Filter tuning

The process noise matrix, Q, in Equation (22) should be selected to represent the process noise as a covariance matrix. A large value for Q results makes the filter more sensitive to the measurements, and a small value makes the measurements have a smaller impact [28].

Moreover, the measurement covariance R matrix and the initial estimate covariance P_0 matrix must be selected.

2.5 Evaluation and evaluation metrics

In the evaluation of the machine learning models and the tracking algorithm, the following evaluation metrics are used.

2.5.1 Machine Learning evaluation

2.5.1.1 Accuracy

The accuracy for a set of predictions, given that the number of correct predictions is known, is defined by Equation (28).

$$Accuracy = \frac{\textit{Number of correct predictions}}{\textit{Number of prections}} \tag{28}$$

2.5.1.2 Mean Squared Error

Given N pairs of points $\hat{\vec{x}}_i$ and $\vec{x}_i$, their Mean Squared Error (MSE) is defined by Equation (29).

$$MSE = \frac{1}{N} \sum_{i=1}^{N} |\hat{\vec{x}}_i - \vec{x}_i| \tag{29}$$

2.5.2 Object tracking evaluation metrics

2.5.2.1 Average Euclidean Distance

Given N pairs of points, $(\hat{x}_i, \hat{y}_i)$, (x_i, y_i) in 2D, their Average Euclidean Distance (AED) is defined by Equation (30).

$$AED = \frac{1}{N} \sum_{i=1}^{N} \sqrt{(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2} \tag{30}$$

3

Methods

3.1 Data generation through simulation

The project commenced by simulating a large amount of data tailored to create specific multi-path scenarios. The dataset was simulated using the MATLAB environments Automated Driving Toolbox and Radar Toolbox.

The Automated Driving Toolbox made designing and simulating driving scenarios possible. The toolbox came with a driving scenario designer, which could be used to create scenarios that replicate the real world. In the scenario designer, a scenario could be created by placing roads and targets. Trajectories and velocities could be set for the targets, which they would follow during the simulation. This made it possible to simulate scenarios with moving targets. To simulate houses next to the road, walls were added. Each house was placed at the corners of the intersection and was built using two walls.

To simulate the radar on the host, the Radar Toolbox was used. The radar toolbox made it possible to simulate how a radar would behave in the scenarios. The data from the radar was returned in the form of radar detections (Section 2.1.3) with the parameters range, azimuth, and range rate as described in Chapter 2.1. Each detection also returned the target hit, and, in the multi-path case, it returned both the target and the reflection target. This made it possible to automatically annotate the data. The detection data also contained the number of reflections, making the separation between multi-path and direct hits simple.

In the simulation, a front radar was used, mounted on the center of the host with a 0° angle. The detections from the simulation were reported in the host coordinate system. In all the scenarios, the host vehicle was moving. To get the range rate relative ground, it had to be ego-motion compensated as described in Section 2.1.1.

The positions, velocities, and labels (see Table 3.1) were extracted for the target objects in the simulation. The host position and heading in world coordinates were extracted for conversion from relative to absolute positions, as described in Section 3.3.

3.1.1 Scenarios

The data set included variations of some scenarios where multi-path propagation was possible. The scenarios include variations of

- Four-way intersection
- Three-way intersection
- 90° angle turn
- Curve with guardrail

The scenarios used were chosen due to their high likelihood of providing NLOS multi-path detections (Section 2.1.2). They are also scenarios that are common in reality. The scenarios were also easy to modify outside the designer. The scenarios were run for four seconds with 0.2 seconds between each timeframe extracted from the scenario, resulting in 19 timeframes per scenario.

One scenario used was a four-way intersection. The scenario was simulated with a host car and one to three target targets. All targets had different initial positions, and only one car was allowed at each entrance to the intersection. Figure 3.1 shows an example with the maximum number of cars. In the figure, the houses are brown next to the road. The host car is visualized in blue color and the other cars are the targets. The lines show the trajectory of the host and targets. The scenario simulated different types of four-way intersections with between one to four houses.

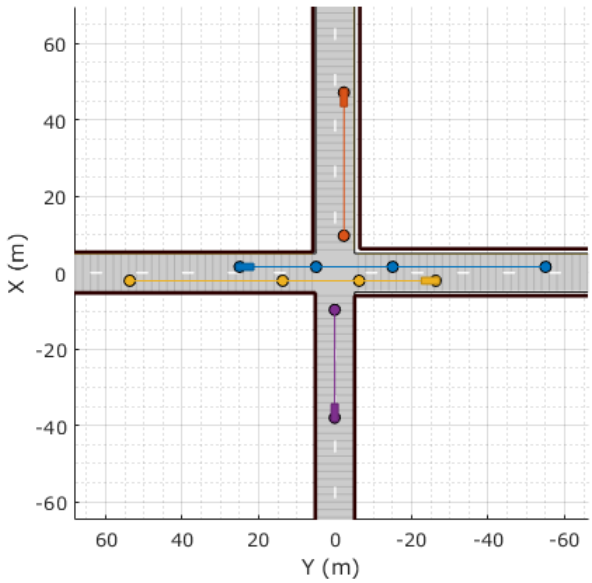


Figure 3.1: Example of a four-way intersection scenario where the host is shown in blue color, targets in orange, yellow, and purple, and walls in dark brown.

Another scenario used was a three-way intersection. The scenario was simulated using a host car and one or two target targets. Since the scenario isn't symmetrical, the initial position of the host was varied. This was done to ensure that all different scenario variations were included in the dataset. As in the four-way scenario, only one car could have an initial position at each entrance to the intersection. The scenario was simulated with one, two, or three houses. The position of the houses can be seen in Figure 3.2. In the figure, the host car is visualized in blue color and the other cars are the targets. The lines show the trajectory of the host and targets.

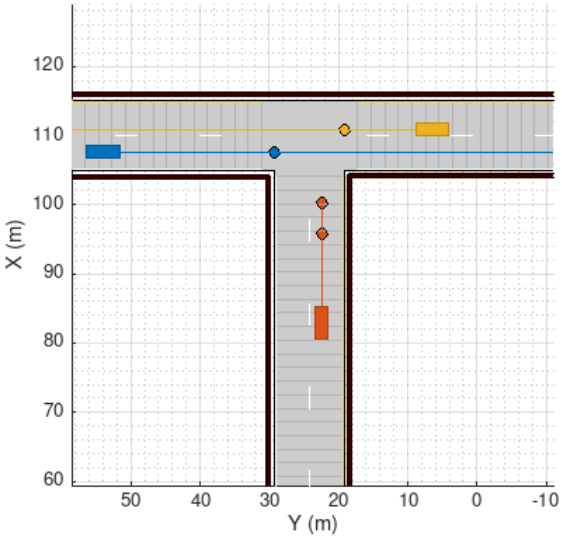


Figure 3.2: Example of a three-way intersection scenario where the host is visualized in blue, targets in orange and yellow, and walls in dark brown.

Another scenario used was the $90°$ angle turn scenario. This scenario simulated a $90°$ angle turn with houses next to the road. The scenario was used since it generated multi-path detection and is an NLOS scenario. In the scenario, the host had a random initial position to generate data for both right and left turns. The scenario and the trajectories of the host and target can be seen in Figure 3.3.

The final scenario used was a curve with a guardrail. Again, the scenario was used since it is common in traffic and generates NLOS multi-path. When generating the scenario, three road centers were used. The MATLAB function then used these road centers to create the road. The road center in the middle was randomized to generate a different variation of the same scenario. This resulted in the curve changing in each new scenario. In the scenario, two cars were used, the host and one target. The cars' initial position and trajectory can be seen in Figure 3.4.

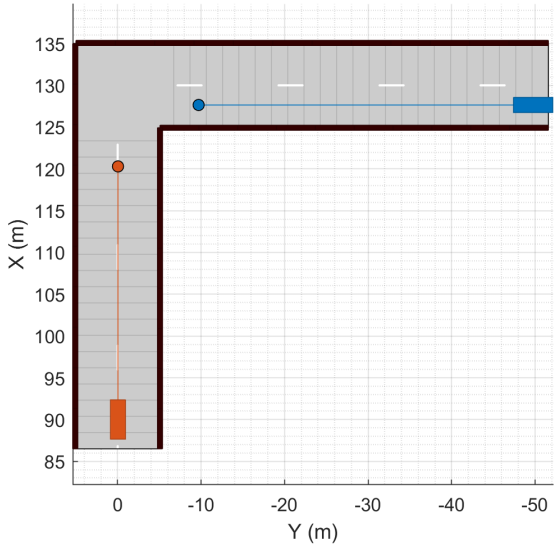


Figure 3.3: Example of a 90° angle turn scenario where host is shown in blue, target in orange, and walls in dark brown.

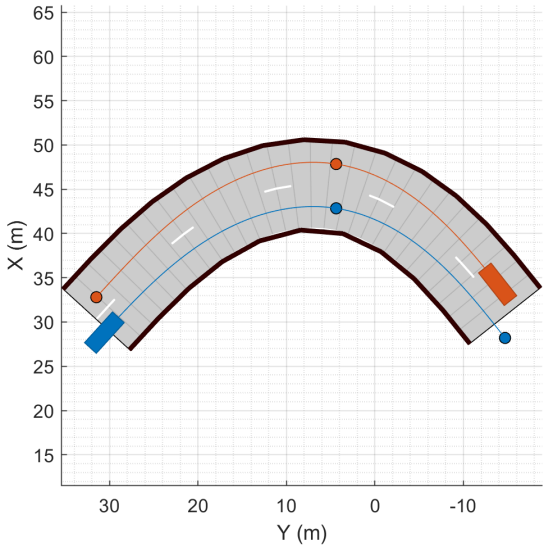


Figure 3.4: Example of a curve with guardrail scenario with host visualized in blue, target in orange, and walls in dark brown.

To increase the size of the dataset, the scenarios described above were used as a base to create new variations of the scenarios. The variation in the scenarios was achieved by randomizing different parameters for both the targets and the scenario.

3.1.2 Scenario variation

For the host and target objects, three parameters were randomized; initial position, velocity, and end position. The initial positions were randomized in both x- and y-direction. The values were randomized by different amounts depending on the scenario type and target start position. For non-symmetric scenarios (for example three-way), the start position needed to be randomized to get all different versions of the scenarios. The variation in velocities was achieved by randomizing a value in a range between 7 and 12 m/s. For each scenario, the object had two velocities; one until it reached the intersection and another after. MATLAB handled the amount of acceleration or breaking. For the intersection scenarios with multiple targets, a traffic light function was implemented to randomize which cars could drive through the intersection. This was done to prevent cars from driving into each other and to mirror reality.

For the scenario parameters, three changes were made to create variation. For each scenario, a random number of walls were used. In addition, the walls were offset from the road with a random value between 0 and 2 m. This replicates reality where houses are placed at different offsets from the roads. The last modification to the scenario was to have a varying road width between 5 and 10 m.

3.1.3 Scenario labeling

As described in Section 3.1, the radar detections' reflection points were accessed and allowed automatic labeling of our dataset. All moving objects in the scenario were labeled, using the points of reflection, at each time step. The objects were labeled as presented in Table 3.1.

Table 3.1: Table over object labeling in simulation data

Type of reflection	Label
Direct reflection on a target	0
Direct reflection and multi-path reflection on target	0
Multi-path reflection only	1
No reflection on target	-1

After that, the scenarios were labeled per time step, using the labels in Table 3.1.

Table 3.2: Table describing scenario labels for the dataset

Object labels	Scenario Label	#*	Scenario description
All 0s or -1s	LOS	0	The objects in the scenario all have direct reflections.
Contains object labeled both 0 & 1	LOS + NLOS	1	There are objects in the scenario that have direct reflections and objects that only have indirect reflection.
All 1s or -1s	NLOS	2	The objects in the scenario only have indirect reflections.
Only -1 labels	No detected car	-1	The objects in the scenario are not detected (have no reflections on them).

**: numerical label*

The data also includes categories referring to the different types of scenarios for use in the result analysis.

3.1.4 Data preprocessing

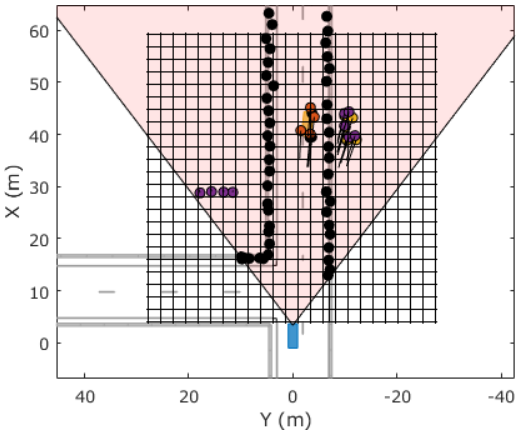
The data from MATLAB comes in the form of detection data including range, range rate, and azimuth. The detection data was first ego-motion compensated as described in Section 2.1.1. Thereafter, the detection data was converted from polar coordinates to x- and y-positions for use as input for the machine learning models. Finally, the data was converted into a grid format to be used as input in the machine learning models, as displayed in Figure 3.5.

The grid format used consisted of two layers of size 128x128. The first layer was a binary layer that had a one if there were a detection in the grid position, otherwise a zero. The second layer had the detections' range rate as the value instead of a binary value. This grid format is suitable since the number of detections and objects varies between the different scenarios and the timeframes of a single scenario.

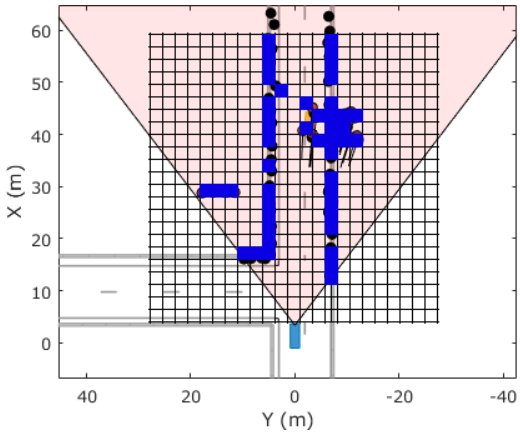
Converting the data from the simulation to a grid format was done by rescaling the x- and y-position to the grid size. The rescaling was done using Equation (31). The equation rescales from the old x- and y-range in MATLAB to the new grid range.

$$\vec{x}_{grid} = \frac{range_{grid}\ (\vec{x}_{MATLAB} - min_{MATLAB})}{range_{MATLAB}} + min_{grid} \tag{31}$$

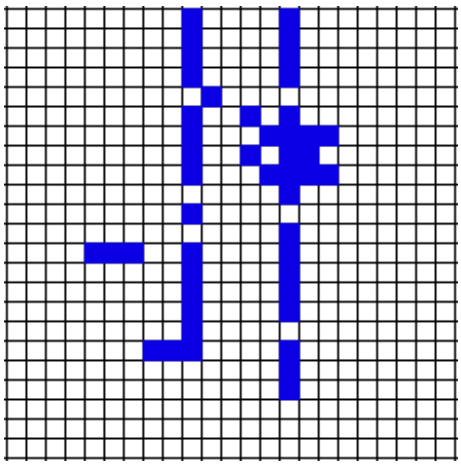
In the equation above, x_{MATLAB} is the x-value from the MATLAB simulation. $range_{MATLAB}$ is the difference between the maximum and minimum x-values in the simulation data. $range_{grid}$ is the difference between the max value of the grid and the min value of the grid, which is 128. min_{MATLAB} is the minimal value in the simulation and min_{grid} is the minimal value in grid. The detection data from the simulation was between 0 and 150 m in x and between -45 and 45 m in terms of y. The cars' ranges ranged between -50 to 80 m in x and between -50 and 50 m in y.



(a) Figure showing the grid placement in the scenario



(b) Figure showing the detections to grid convertion



(c) Grid representation used for the machine learning models

Figure 3.5: Representation of the grid conversion from detection to the grid, not shown in correct grid scale

In order to enhance the resolution of the grids without increasing their size, two modifications were implemented. The first modification involved eliminating cars located behind the ego vehicle, as they are undetectable by radar. Subsequently, the grid was repositioned with the host origin situated at the midpoint of the grid edge, as illustrated in Figure 3.5a.

The next improvement was to filter out detection and cars outside a given boundary. This removed all detections that had a range greater than 80 m. The boundary of 80 m was chosen since we only simulate objects at a maximum range of 80 m. Detections that had a normed y-value greater than 40 m were removed. This boundary for y was chosen to get the same grid resolution in both x and y. Filtering out detections far away improved the resolution since the scaling is done from a smaller range. The detections to grid representation can be viewed in Figure 3.5. The figures show where the grid is placed in the scenario and how the detections are converted to a grid representation. In the figure, direct path detections are shown in black, and multi-path detections are shown in yellow, red, and purple. The different multi-path cases are presented in Section 2.1.2. In Figure

3.5c the final grid used as input is shown. The grids shown in the figures are not true to scale and are solely used to visualize the conversion from detections to grids.

3.2 Application of machine learning model

The data was split into separate training and test sets, where the training set was used in hyper-parameter tuning and network fitting. The hyper-parameter tuning was done for the learning rate used in the model. The validation set was used for hyper-parameter tuning and early stopping (Section 2.3.2), and the test set was used for result analysis. The training set consisted of 64 % of the data, the validation set 16 %, and the test set 20 %. Note that since the data from the simulation consists of 19 timesteps long sequences (a total of four seconds), the split was done such that data points from the same sequence are in the same set split. This to prevent the models from being evaluated on scenarios they have already encountered.

The machine learning models were constructed in Python. The Keras package was used to create the models. Keras is an API for deep learning that runs on top of TensorFlow, a machine learning platform. Keras was chosen due to previous experience in the package.

3.2.1 Implementation

Three machine learning tasks were implemented, as listed below, where the main goal was the last task.

1. **Scenario Labeling task:** Use a Convolutional Neural Network (CNN) to classify the scenarios, with the target labels as in 3.2.

2. **Number of object estimation:** Use a CNN to identify the number of targets in the scenario.

3. **Positioning task:** Use the position of the target objects in the simulation data as a target and train a Deep CNN to position the unknown object.

3.2.1.1 Classification

The same network structure was used for the scenario labeling and the number of object estimation tasks described above. The network was a CNN with three convolutional layers (Section 2.2). This network structure was selected for its relatively small size and simple structure, and its ability to process image-like data. Between each convolutional layer was a max-pooling layer. The final max-pooling layer was followed by a Dropout-layer and a flattened layer. A flattened layer is used to flatten the output to a one-dimensional array. The last two layers in the model were Dense ReLU layers with 64 and 4 neurons. The output layer had 4 neurons because the classification tasks had 4 classes each.

The input data was on a grid containing information about radar detections' positions and range rates as described in section (Section 3.1.4), and the target was a number between -1 and 2 for the labels and between 0 and 3 for the number of cars. The model used a

sparse categorical Cross-Entropy as the loss function. The loss function works well for the output data since only one category is correct at each time step. The optimizer used was the Adam optimizer with a learning rate of 0.0028 for label classification and 0.00025 for the classification number of objects.

3.2.1.2 Positioning CNN

The first machine learning model used for positioning was a CNN model. The model consisted of three convolutional layers with max-pooling layers in between (Section 2.2.1). Following the convolutional layers were four dense layers (Section 2.2). The first three dense layers used a ReLU activation function (Section 2.2.7.1). The final dense layer in the model was the output and had a Sigmoid activation (Section 2.2.7.2). A Sigmoid activation was used for the output to return the probability that an object is in a specific position in the grid. This allows for the interpretation of the outputs as probabilities. One extra output unit was added to represent when no target object exists in the scenario. This was needed to ensure convergence of the loss function; the output target always had one position with a value greater than zero.

The loss function used for the model was Binary Cross-Entropy as described in Section 2.3.1.1. The loss function was used since the target output is binary and the possibility to interpret the outputs as probabilities. As multiple outputs could be correct, binary is a better choice compared to Categorical Cross-Entropy. Binary Cross-Entropy works well for the output used since it computes the loss by comparing each output to the true value. The model was trained using the Adam optimizer with a 0.005 learning rate (Section 2.3).

The input to the model was a three-dimensional grid, with radar detection positions and range rates as described in Section 3.1.4. The grid input works well with convolutional layers that need the input to be a grid. The targets described in Section 3.1.4 were flattened to match the output layer of the model, and an extra position was added last in the target. The last position in the flattened output was set to one if no object existed in the scenario and zero otherwise.

3.2.1.3 Positioning U-Net

The second model used was a U-Net model, which is described in Section 2.2.2. The U-Net model was selected because of its promising results in radar applications and its ability to process image-like input data. The first layer in the model was a convolutional layer. The model consisted of three downsampling blocks, which were followed by four upsampling blocks. The final layer of the model was a convolutional layer with a Sigmoid activation function that returned a grid with the probabilities of object positions. The output was in the form of the targets described in Section 3.1.4.

The U-Net model utilizes convolutional layers, max-pooling layers, separable convolutional layers, transposed convolutional layers, and batch normalization. These layers are described in Section 2.2. An overview of the U-Net structure is seen in Figure 2.9, and a more detailed version can be found in Figure A.1.

One downsampling block consists of two separable convolutional layers followed by a

max-pooling layer. Before the separable convolutional layers, a ReLU layer was used, and after the convolutional layer, a batch-normalization. The last layer in the downsampling block was a convolutional layer. The downsampling blocks used an increasing number of filters to sample down the data from the previous block. This means that the number of filters needs to be a power of two for the model to work. Therefore, the model used 64, 128, and 256 for the downsampling blocks.

The upsampling blocks were built with the same structure as the downsampling blocks. Instead of the separable convolutional layer, a convolutional transpose layer was used. The max-pooling layer was changed to an upsampling layer. The upsampling blocks used a decreasing number of filters to sample up the data. The number of filters used was 256, 128, 64, and 32.

The loss function Binary Cross-Entropy and the Adam optimizer with a 0.0017 learning rate were used.

The three-dimensional input and two-dimensional outputs described in Section 3.1.4 were used for the model. Since U-Net returns a grid, an extra position for when no objects exist could not be added. This was solved by setting the top left corner to one if no objects existed in the target. Since the model outputs are the same dimensions as the input data, there is no need to flatten the target data, which could lead to information loss.

One variation of the U-Net model was to use two timesteps as input for the model. The input for the model was the input data at the previous time step t_{n-1} and the current time step t_n. The model then used this input to predict the position for the target at t_n. This variant was examined to see if the model could use previous data to improve the prediction.

3.2.2 Result analysis

After training a machine learning model, its results were verified on a test set. The outcome of the analysis on the test set was visualized in plots using Python, using appropriate evaluation metrics.

3.3 Object Tracking

After the result analysis, the outputs of the best-performing machine learning model were used to track the objects over time. The tracking algorithm utilizes the machine learning outputs as measurements for x- and y-positions in the model.

The machine learning model that predicts the number of targets was used to decide how many measurements to use so that the number corresponds to the number of targets, n. The output from the positioning model was preprocessed so that the n most likely positions are provided as measurements to the tracking algorithm.

As the positional output from the machine learning was in a grid format relative to the host, the measurements were converted from the grid format to Euclidean coordinates

and to absolute instead of relative coordinates. First, the positions x and y were rotated according to Equation (32), where θ is the heading angle of the host.

$$\begin{aligned} x_{rotated} &= x\ cos(\theta) - y\ sin(\theta) \\ y_{rotated} &= x\ sin(\theta) + y\ cos(\theta) \end{aligned} \tag{32}$$

Then, the positions were translated according to Equation (33).

$$\begin{aligned} x_{world} &= x_{rotated} + x_{host} \\ y_{world} &= y_{rotated} + y_{host} \end{aligned} \tag{33}$$

The positional measurements, $x_{world},\ y_{world}$ were passed through the tracking algorithm at every time step, as presented in Figure 3.6.

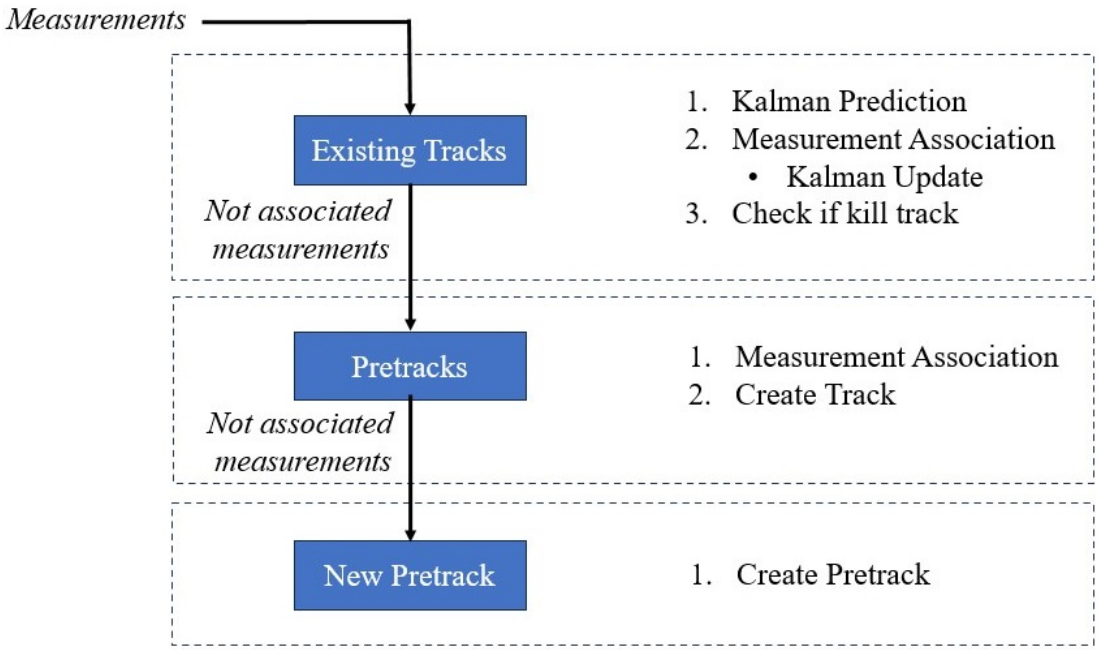


Figure 3.6: Image description of the tracking algorithm.

First, the algorithm loops over already existing tracks. All existing tracks use their previous state in a prediction step in the Kalman filter as described in Section 2.4.1.2. Then, the measurements are associated with the existing tracks. If a measurement is within 5 meters of Manhattan distance (Equation (13)) to the track's latest state, the measurement is associated with the track. The Manhattan distance is used since the targets move in orthogonal directions in the majority of the scenarios, meanring that the scenarios are typical for those seen in a Manhattan-like grid, with $90°$ angles, as described in Section 3.1.1. A Kalman measurement update is done if the track receives an associated measurement, as described in Section 2.4.1.1. If the track has not received an associated measurement in two iterations, it is killed and will no longer be used.

The measurements not associated after passing through all existing tracks are checked against the pre-tracks. The same association step is done for all the pre-tracks. If the pre-track has not received measurements in three iterations, it is killed and removed. Then, if the pre-track has more than two associated measurements, it is used to create a track. The initial state position, velocity, and heading of the track are calculated from the pre-track measurements as described in Equations (34) - (36), where v is the velocity, $\vec{x} = [x, y]^T$ is the position, and θ is the heading. N is the number of associated measurements to the pretrack. The initial turn rate was set to 0.

$$v_{track} = \frac{\vec{x}_{pretrack\ end} - \vec{x}_{pretrack\ start}}{N\Delta t} \tag{34}$$

$$\vec{x}_{track} = \frac{1}{N}\sum^{N} \vec{x}_{pretrack} \tag{35}$$

$$\theta_{track} = atan2\left(\frac{y_N - y_0}{x_N - x_0}\right) \tag{36}$$

where $atan2$ is the two-argument arctangent, which returns an angle θ between $-\pi$ and π.

If there are still measurements that are not associated after passing through all tracks and pre-tracks, they are used to create new pre-tracks.

This algorithm was iterated over each scenario of 19 timesteps separately so that at the start of a new scenario no tracks exist.

3.3.1 Filter tuning and result analysis

Since the actual target positions and velocities were known from the simulation, the tracking error was calculated by comparing the estimated positions and velocities with the corresponding known targets in the simulation. The target that had the smallest difference in position was selected as the matching target. Its velocity and position were then used to calculate errors. These errors were used to tune the Kalman filter and present the tracking algorithm's resulting errors.

A tuning algorithm was constructed to select the values for the Q and P_0 (initial value for P) matrices (Section 2.4.1) in the Kalman filter. The algorithm chosen was random search, sampled from a normal distribution. The sampled values for P_0 and Q were evaluated on all the data, combining velocity Means Squared Error (MSE) (Equation (29)) and Average Euclidean Distance (AED) for the x- and y-positions (Equation (30)).

The random search means and standard deviation were updated such that the mean values of their distributions were the values with the lowest resulting error. The standard deviations used in the distributions were decreased as the number of iterations increased.

The value for the measurement error matrix, R, was calculated by measuring the errors in x- and y-positions from the machine learning model output. Their covariance matrix was used as R.

3.4 Algorithm summary

The steps above are summarized in Figure 3.7, which describes the machine learning and tracking pipeline employed in this project.

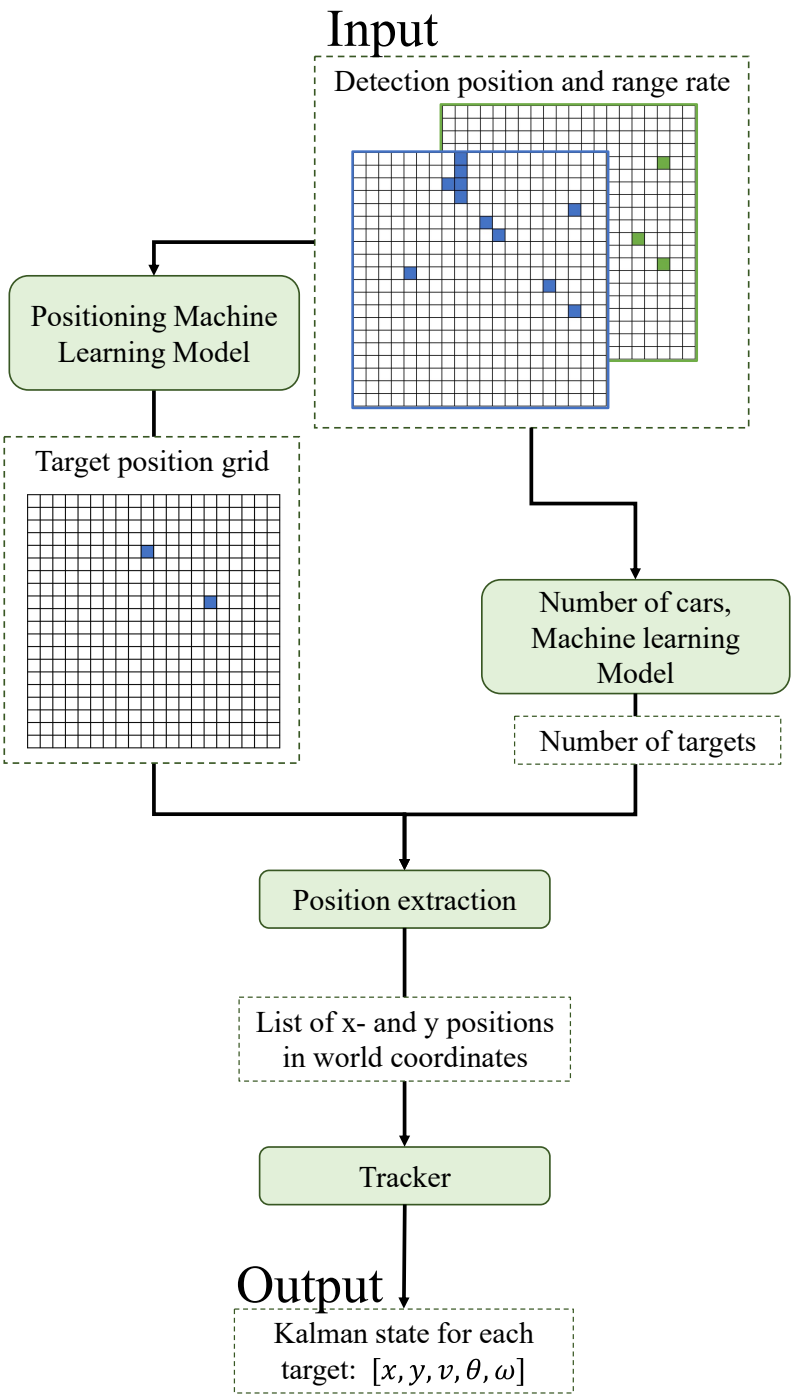


Figure 3.7: A figure over the tracking pipeline implemented in this project. The three main components are the positioning machine learning model, the number of cars predictive model, and the tracker. The pipeline displays how the input from one timestep is used to create a state vector containing position, velocity, heading, and turn rate.

The input matrices, containing information about radar detection positions and their range rate, are used as input into two separate machine learning models. One, the positioning model, outputs a target position grid with positional probabilities. Using the output of the other machine learning model, which predicts the number of targets, the n largest

grid probabilities are used as positional measurements for our tracking algorithm. This pipeline is iterated for each time step and outputs a Kalman state vector with the estimated state for each target.

4

Results

4.1 Simulation

This section presents the simulated datasets and their distributions of scenario labels, types, and number of targets.

In Table 4.1, the distribution of scenario timeframes can be seen for the training dataset, where each timeframe represents the data from a single timestep. The majority of the data set is LOS timeframes, as seen in Table 4.1a. Table 4.1b shows the distribution between the different scenario types. There are more four-way and three-way scenarios compared to the other types. The majority of the timeframes include one target, thereafter two, zero, and three targets. Few timeframes with three targets exist In Figure 4.1, the distribution of timeframes can be seen for scenarios and labels. The three-way scenarios contain the most NLOS targets and the four-way scenarios contain the most LOS + NLOS. The figure shows that in the $90°$ angle turn and curve scenario, the target is mostly either NLOS or not visible. In the figure, no timeframes for LOS+NLOS exist in curve and $90°$ angle turn scenario due to the scenario only containing one target.

Table 4.1: Distribution of scenario labels, types, and number of targets for the training dataset

(a) Scenario label distribution

No target	2383
LOS	5158
LOS + NLOS	1968
NLOS	2651

(b) Scenario type distribution

Four-way	4864
Three-way	4731
Curve	1216
$90°$ angle turn	1349

(c) Distribution of the number of targets

0	2383
1	5802
2	3508
3	467

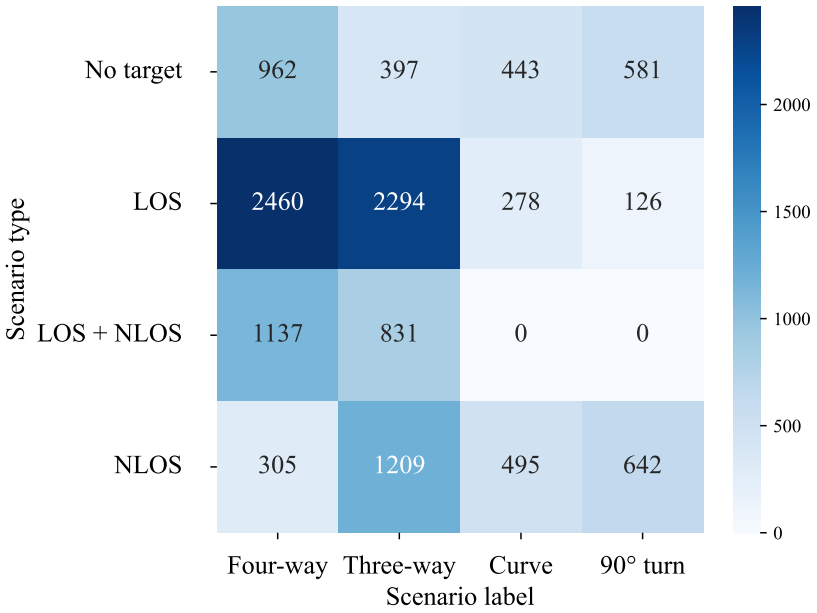


Figure 4.1: Heatmap over the distribution of scenario types and labels in the training dataset

The corresponding distributions for the test dataset can be seen in Table 4.2. The distribution follows the distribution of the training dataset. Figure 4.2 shows the distribution of timeframes over the scenarios and labels.

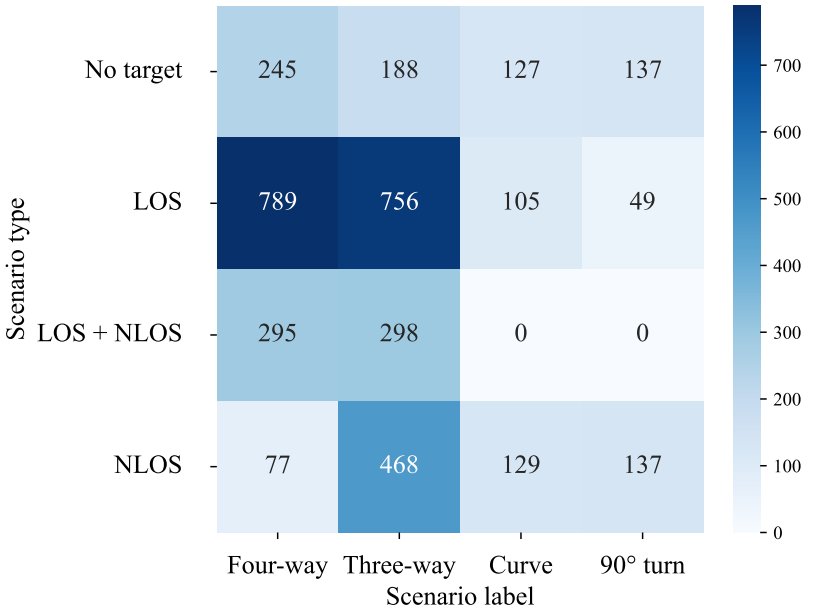


Figure 4.2: Heatmap over the distribution of scenario types and labels in the test dataset

Table 4.2: Distribution of scenario labels, types, and number of targets for the test dataset

(a) Scenario label distribution

No target	697
LOS	1699
LOS + NLOS	593
NLOS	811

(b) Scenario type distribution

Four-way	1406
Three-way	1710
Curve	361
90° angle turn	323

(c) Distribution of the number of targets

0	697
1	1862
2	1102
3	139

4.2 Machine Learning

All evaluation of the machine learning models is done on the test set described in Section 4.1. The training of the models is done on a separate training set. Machine learning models were constructed for three different tasks, scenario labeling (Section 4.2.1), number of targets prediction (Section 4.2.2), and positioning (Section 4.2.3).

4.2.1 Scenario labeling model

The results of the classification model trained to predict the scenario label; NLOS, LOS + NLOS, LOS, or no target, are presented below.

In Figure 4.3, the confusion matrix for predicting labels is presented. The diagonal indicates correct prediction, and the other grid positions indicate incorrect predictions. The heatmap shows that the model predicts correctly for the majority of timeframes. This can be further seen in Table 4.3 where the accuracy for each scenario label is shown. The model learns to classify the different labels well and achieves the best accuracy when no target exists. The table shows that the model performs similarly for LOS and NLOS predictions but has slightly poorer performance on LOS and NLOS combined.

Table 4.3: Label classifiction accuracy for each scenario label

Scenario Label	Accuracy
No target	99.70 %
LOS	98.29 %
LOS + NLOS	97.30 %
NLOS	98.34 %

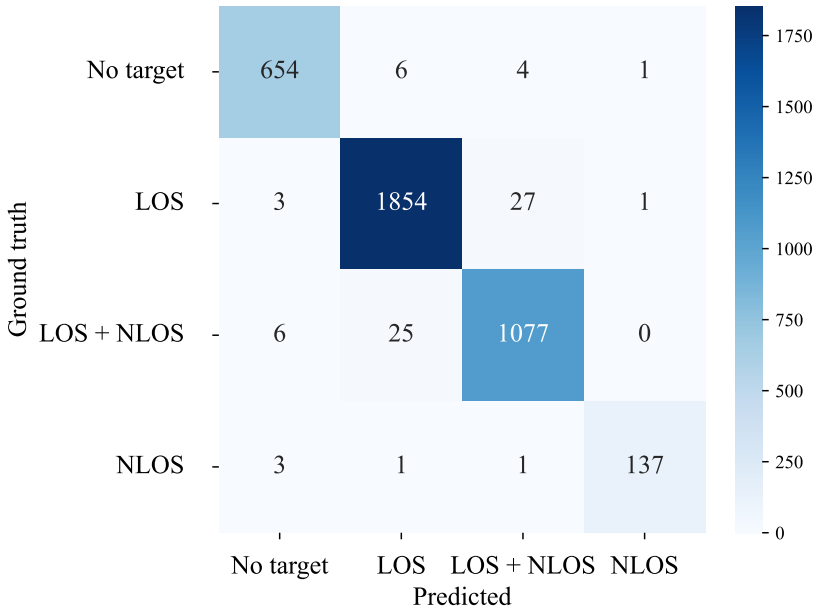


Figure 4.3: Confusion matrix for scenario label prediction.

The label classification accuracy for each scenario type is presented in Table 4.4. The model has the highest accuracy in three-way and curve scenarios and the poorest accuracy in the 90° turn scenario.

Table 4.4: Label classification accuracy for each scenario type

Scenario Type	Accuracy
Four-way	97.01 %
Three-way	99.64 %
Curve	99.45 %
90° angle turn	96.59 %

4.2.2 Number of targets model

Presented in this section are the results from the classification model used to predict the number of targets in a scenario. The predicted number of targets is used in object tracking to extract the relevant measurements from the positioning model.

The confusion matrix for the classification of the number of targets is shown in Figure 4.4 below. The correct predictions are displayed in the diagonal, where most of the predictions are counted, indicating that the model classified the scenario timeframe correctly.

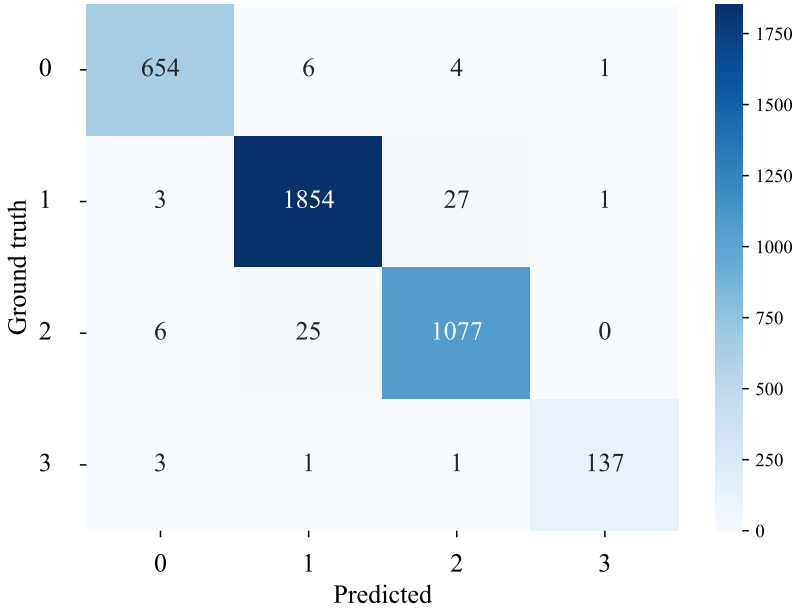


Figure 4.4: Confusion matrix predicting the number of targets

Table 4.5 presents the accuracy for each class. The model has similar accuracy for zero to three targets. The model achieves the poorest accuracy for two targets and the highest for three targets.

Table 4.5: Classification accuracy for each number of targets

N.o. targets	Accuracy
0	98.20 %
1	98.30 %
2	97.11 %
3	98.56 %

The prediction accuracy with respect to scenario type is presented in Table 4.6. The model achieves the best accuracy for three-way scenarios and the worst for four-way scenarios.

Table 4.6: Accuracy for each scenario label for the classification model

Scenario Type	Accuracy
Four-way	95.52 %
Three-way	99.59 %
Curve	99.72 %
90° angle turn	97.83 %

4.2.3 Positioning model

The final machine learning model results are presented in this section. The models evaluated are U-Net and CNN. The U-net was also modified to take the previous and the current timeframes as input data instead of only the current timeframe. This model is presented below as U-Net 2t.

The table below presents the accuracy of the three models described in Section 3.2.1. The accuracy is described by Equation (28). For a single prediction to be considered correct, all values of the prediction array are the same as the expected outputs. Table 4.7 reveals that the U-Net and CNN have similar accuracy and the U-Net 2t has a slightly lower accuracy.

Table 4.7: Accuracy on the entire test dataset for the three different models

Model	Accuracy
U-Net	93.16 %
U-Net 2t	90.06 %
CNN	92.97 %

The table 4.8 shows the mean squared error for the different models over all scenarios.

Table 4.8: MSE for the three different models on the test dataset

Model	MSE
U-Net	6.72 m^2
U-Net 2t	33.72 m^2
CNN	26.92 m^2

Table 4.9 shows the Average Euclidean distance for the different models.

Table 4.9: AED for the three different models on the test dataset

Model	AED
U-Net	0.21 m
U-Net 2t	0.87 m
CNN	0.84 m

Figures 4.5 - 4.7 are histograms of the errors from the models. In the histograms, the probability distribution functions are fitted on all the errors. The results with zero errors have been excluded from the histogram to improve the visual clarity of the errors.

The error for the U-Net model is presented in Figure 4.5. The errors from the model are close to 0 the majority of the time. The figure shows the x and y errors separately, and it can be seen that the model has a lower maximum error for x compared to y. It can also be

seen that the model has more errors in x than y. The mean and standard deviation further shows this, where x has a mean closer to 0 and a smaller standard deviation. The error in x has a mean value of -0.005 m and a standard deviation of 1.702 m; the error in y has a mean of -0.029 m and a standard deviation of 1.954 m.

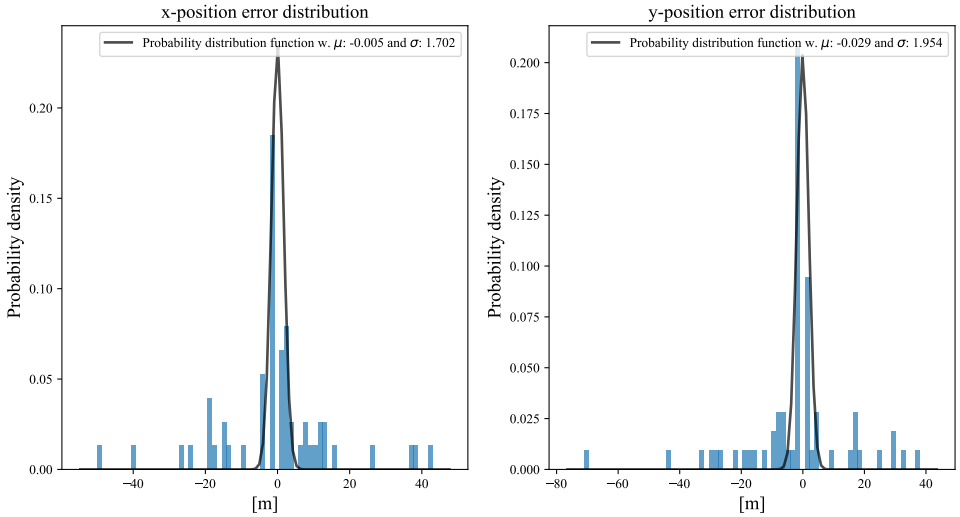


Figure 4.5: Histogram showing the distance in x- and y-positions between target and prediction for U-Net

The histogram in Figure 4.6 shows the errors for the U-Net 2t. It can be seen that the errors in x are smaller compared to y. The error in x has a mean value of -0.0118 m and a standard deviation of 3.299 m; the error in y has a mean of -0.249 m and a standard deviation of 4.771 m.

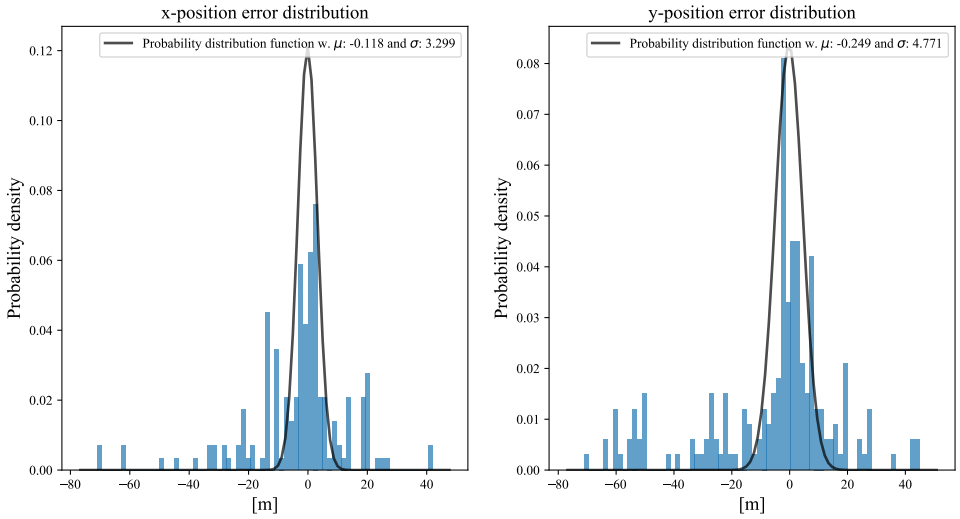


Figure 4.6: Histogram showing the distance in x- and y-positions between target and prediction for U-net 2t

The errors are shown as a histogram for the CNN model in Figure 4.7 below. The model has more errors in x compared to y. The error in x has a mean value of -0.056 m and a standard deviation of 3.223 m; the error in y has a mean of -0.059 m and a standard deviation of 4.065 m.

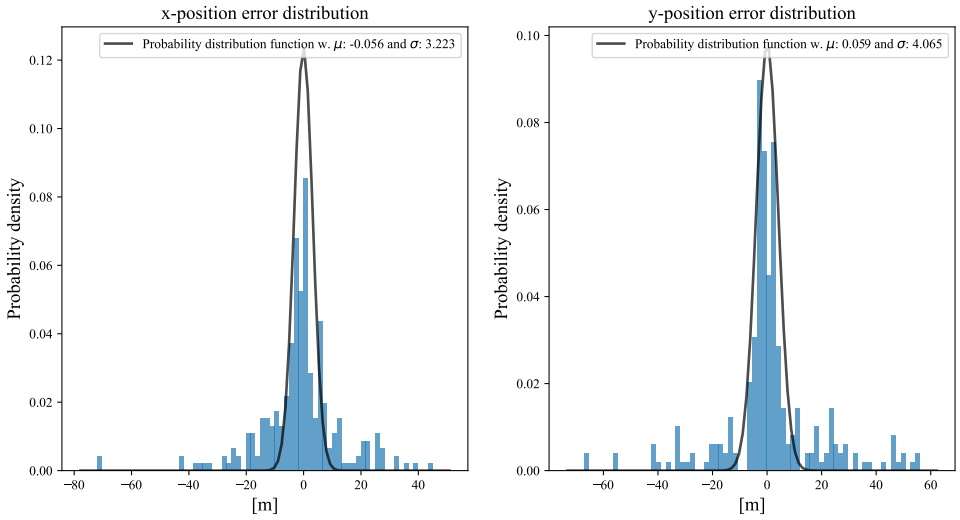


Figure 4.7: Histogram showing the distance in x- and y-positions between target and prediction for CNN

4.2.3.1 Comparison for different scenario labels

A result comparison for the different scenario labels: No target, LOS, LOS + NLOS, and NLOS was done.

Figure 4.8 displays the accuracy of the three models for each scenario label. It can be seen that the U-Net and CNN model achieve similar accuracy for the different labels. The U-Net 2t performs slightly worse for all four labels.

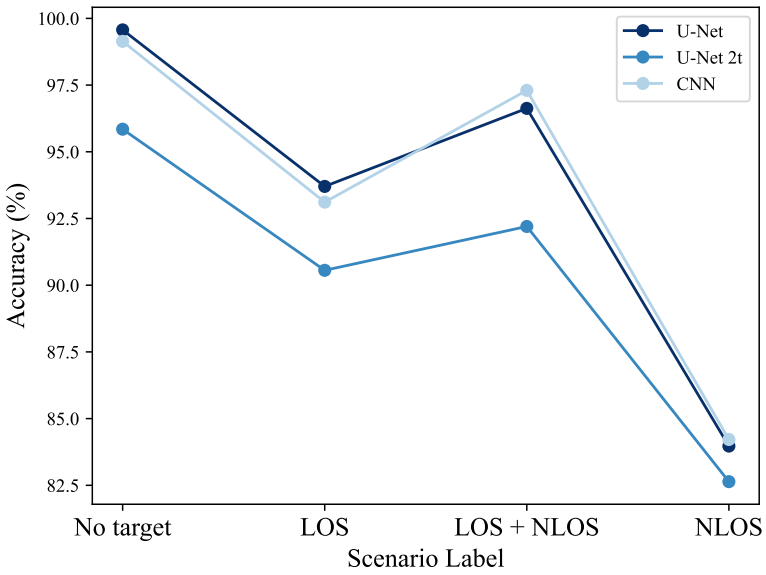


Figure 4.8: Positioning accuracy for the different models with respect to the scenario labels

Table 4.10 presents the accuracy values for the different labels plotted in Figure 4.8. The table shows that U-Net achieves the best accuracy for the label no target, followed by CNN. The columns referring to timeframes containing targets (LOS, LOS + NLOS, and LOS) display a similar performance of U-Net and the CNN model. The U-Net 2t has a poorer performance for all scenario labels.

Table 4.10: Position accuracy for the different labels

Model	No target	LOS	LOS + NLOS	NLOS
U-Net	99.57 %	93.70 %	96.62 %	84.56 %
U-Net 2t	95.85 %	90.55 %	92.20 %	82.64 %
CNN	99.14 %	93.11 %	97.30 %	84.09 %

4.2.3.2 Comparison for different scenario types

A result comparison for the different scenario types: no target, LOS, LOS + NLOS, and NLOS was done.

Figure 4.9 shows the positioning accuracy for the different scenario types. All models achieve the highest accuracy for the three-way scenario and the lowest for the 90° turn scenario. In the four-way scenario, the CNN and U-Net models achieve better accuracy than U-Net 2t. In the curve scenario, the U-Net model performs best, followed by CNN.

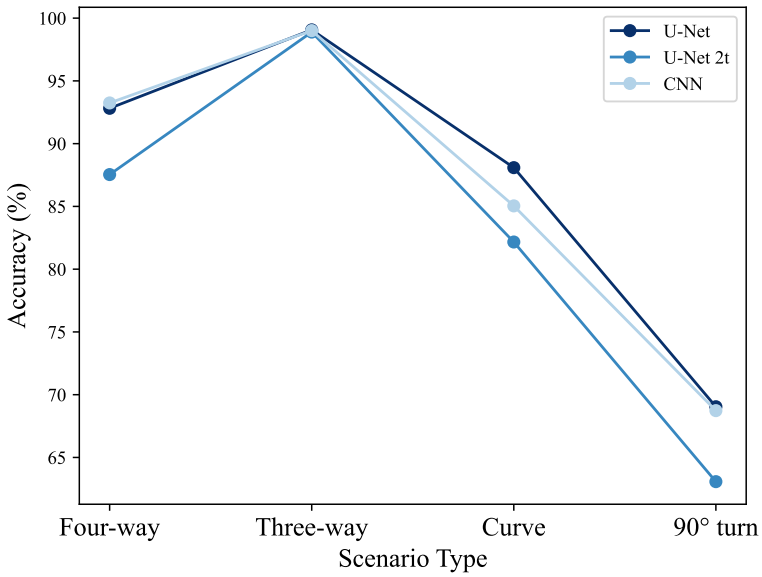


Figure 4.9: Positioning accuracy shown for each of the scenario types

Tables 4.11 present the accuracy values seen in Figure 4.9. The table shows that U-Net and CNN achieve similar accuracy and that U-Net 2t achieves a lower accuracy in the four-way and three-way scenarios. In the curve scenario, the U-Net model achieves the best accuracy, followed by the CNN and U-Net 2t model. The models have a poorer performance in the 90° turn scenario. All three models perform similarly for the scenario, with U-Net performing best.

Table 4.11: Position accuracy for the different scenarios

Model	Four-way	Three-way	Curve	90° turn
U-Net	92.81 %	99.06 %	88.09 %	69.04 %
U-Net 2t	87.54 %	98.89 %	82.16 %	63.07 %
CNN	93.24 %	99.01 %	85.04 %	68.73 %

In Figure 4.10, the accuracy values for the U-Net model are displayed as a heatmap over the labels and scenarios. The model achieves the best accuracy for LOS in the curve scenario and the worst accuracy for no target in the curve scenario. LOS + NLOS have -1 accuracy in curve and 90° turn, which indicates that no timeframes for the scenario exist in the data set.

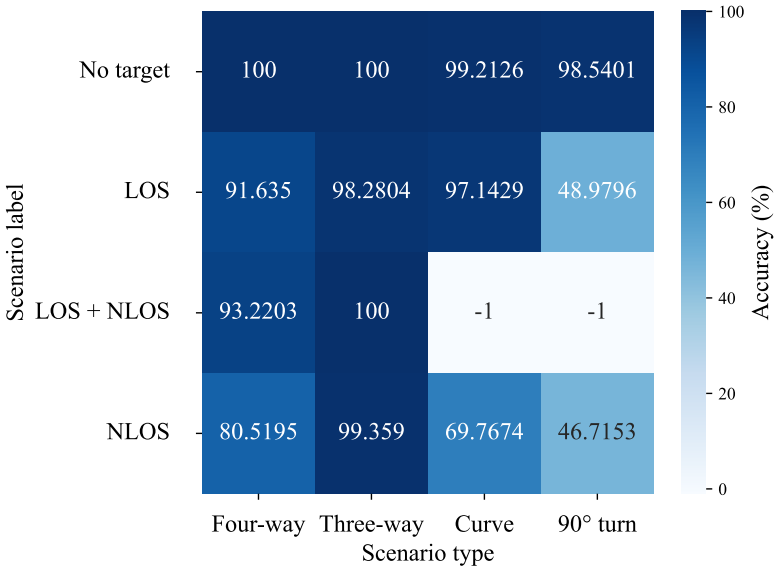
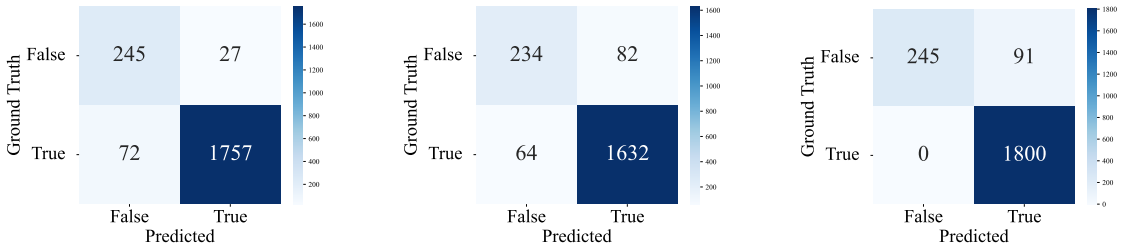


Figure 4.10: Heatmap showing accuracy for the labels and scenarios for the U-Net model

Figures 4.11-4.14 show the confusion matrix for the different scenario types. In the top left of the matrices is the number of times the model predicts no target correctly. The bottom left is when the model predicts no target when a target exists. The top right is when the model predicts the position wrong, and the bottom right is when the model predicts the position correctly. The model predictions were considered correct if the predicted position was less than 2 m from the target.

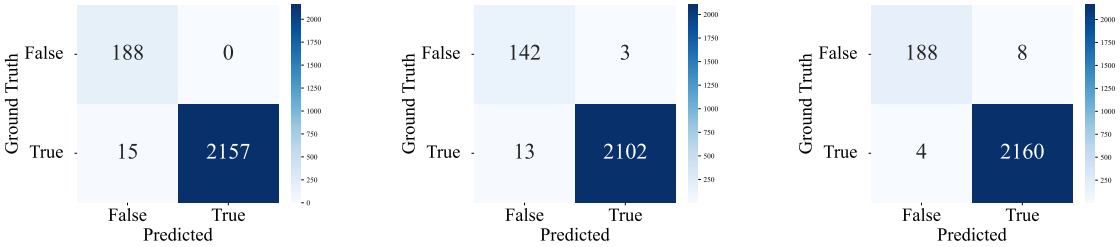
Figure 4.11 is the confusion matrix for the four-way scenario for the models. The models predict the correct position most of the time. One difference can be seen between the U-Net models and the CNN model when the model predicts incorrectly. The U-Net model predicts that no target exists more often than CNN, which instead predicts incorrect positions. This trend can be seen in all the different scenario types.



(a) U-Net **(b)** U-Net 2t **(c)** CNN

Figure 4.11: Confusion matrices for the four-way scenario

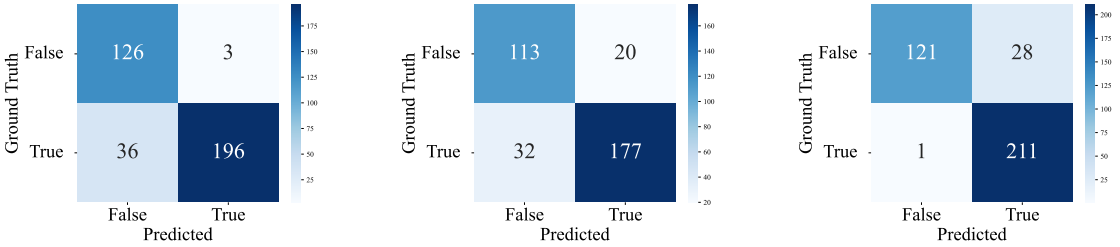
Figure 4.12 presents the confusion matrices for the three-way scenario. The models perform well for the scenario which can be seen from the low number of incorrect predictions.



(a) U-Net **(b)** U-Net 2t **(c)** CNN

Figure 4.12: Confusion matrices for the three-way scenario

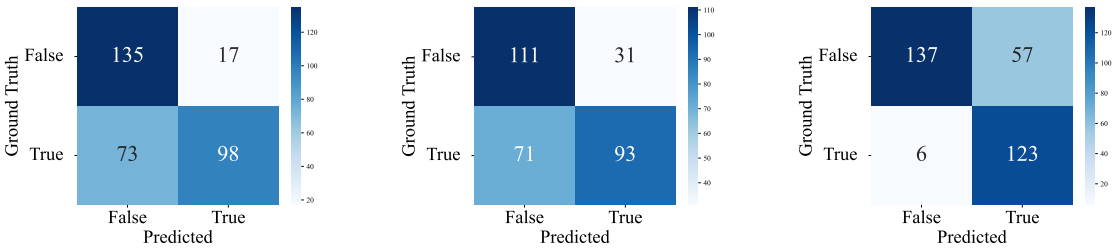
Figure 4.13 shows the confusion matrices for the curve scenario. The matrices show that the models perform well and achieve low number of incorrect predictions.



(a) U-Net **(b)** U-Net 2t **(c)** CNN

Figure 4.13: Confusion matrices for the curve scenario

In Figure 4.14, the confusion matrix for the 90° angle turn scenario can be seen. It can be seen that the U-Net model predicts no target when a target exists half the time. The CNN model predicts correctly that a target exists but predicts an incorrect position which can be seen in the top right in Figure 4.14c.



(a) U-Net **(b)** U-Net 2t **(c)** CNN

Figure 4.14: Confusion matrices for the 90° angle turn scenario

4.2.3.3 Comparison for different number of targets

The plot below shows the three models' accuracy for different numbers of targets. The accuracy values are from the test dataset described in Section 4.1. In Table 4.2c, the

number of timeframes of each scenario type can be seen. Figure 4.15 shows that U-Net and CNN achieve similar accuracy for zero to two targets, and CNN achieves a higher accuracy for three targets. The U-Net 2t has lower accuracy for all numbers of targets.

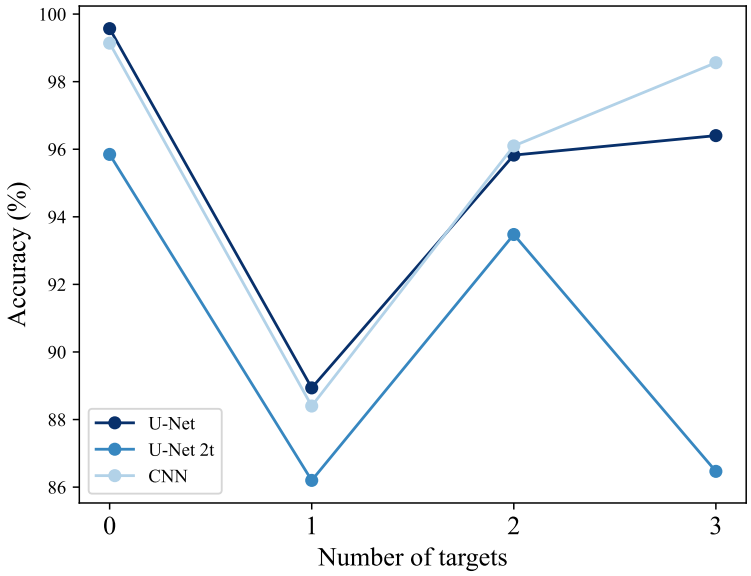


Figure 4.15: Accuracy for different number of targets

The Table 4.12 below presents the accuracy values from Figure 4.15. The table shows that U-Net and CNN achieve the best accuracy for no target. In scenario timeframes with one target CNN and U-Net have the best accuracy, and U-Net 2t is slightly worse. The same trend is visible in scenarios with two and three cars.

Table 4.12: Position accuracy for the different number of objects

Model	No target	One target	Two targets	Three targets
U-Net	99.57 %	88.93 %	95.82 %	96.40 %
U-Net 2t	95.85 %	86.19 %	93.48 %	86.46 %
CNN	99.14 %	88.40 %	96.10 %	98.56 %

4.3 Object tracking

The positional outputs from the best-performing machine learning model, U-Net, were input to an algorithm to track the objects over time. They were used as measurements in a Kalman filter as described in Section 2.4.1.1.

R is calculated to represent the measurement error covariance, calculated from the distributions of errors in x- and y-positions from the machine learning model, as described in Figure 4.5.

$$R = \begin{bmatrix} 2.89 & 0.744 \\ 0.744 & 3.82 \end{bmatrix} \tag{37}$$

The tuning of the Kalman filter resulted in the following values for Q (38) and P_0 (39).

$$Q = \begin{bmatrix} 13.820 & 0 & 0 & 0 & 0 \\ 0 & 0.421 & 0 & 0 & 0 \\ 0 & 0 & 0.179 & 0 & 0 \\ 0 & 0 & 0 & 9.781 & 0 \\ 0 & 0 & 0 & 0 & 12.744 \end{bmatrix} \tag{38}$$

$$P_0 = \begin{bmatrix} 30.210 & 0 & 0 & 0 & 0 \\ 0 & 83.106 & 0 & 0 & 0 \\ 0 & 0 & 6.989 & 0 & 0 \\ 0 & 0 & 0 & 162.982 & 0 \\ 0 & 0 & 0 & 0 & 16.852 \end{bmatrix} \tag{39}$$

An example of the result of the algorithm is displayed in Figures 4.16, 4.17, and 4.18. The plots show the measurements in x- and y positions as blue dots, the target values in orange, and the predicted values in blue. The plots capture the behavior of the tracking algorithm. Figures 4.16 and 4.17 show how the Kalman filter adapts the positional state to the input measurements. Despite not receiving any velocity measurements, the tracker also succeeds in estimating the objects' velocities (Figure 4.18). The mean absolute error in velocity over all scenarios and timeframes is 1.34 m/s, and the AED in position is 0.80 m.

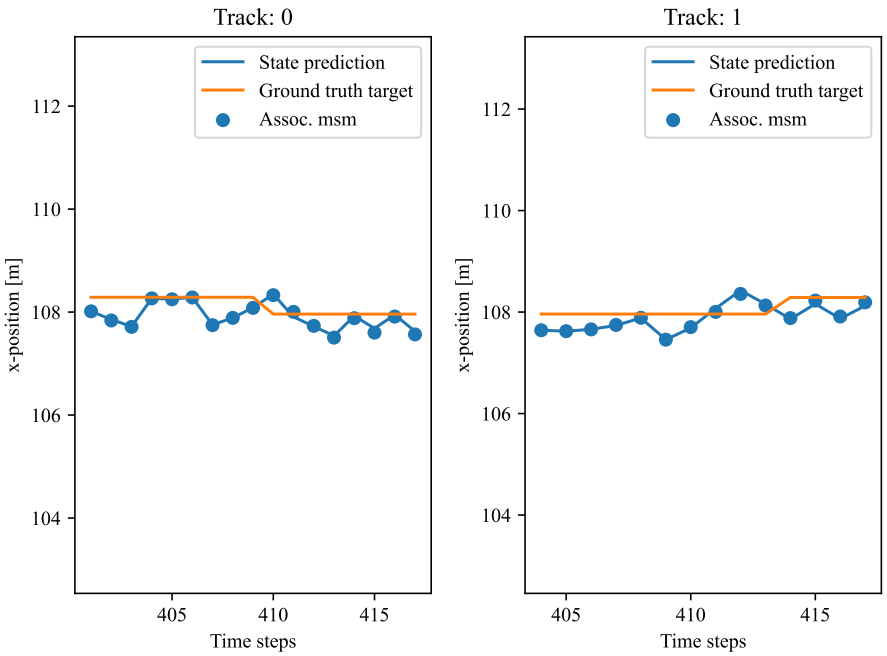


Figure 4.16: An example of the result of our object tracking in x-position, with the target x-position in orange, the estimated x-position in blue, and the associated measurements as blue dots.

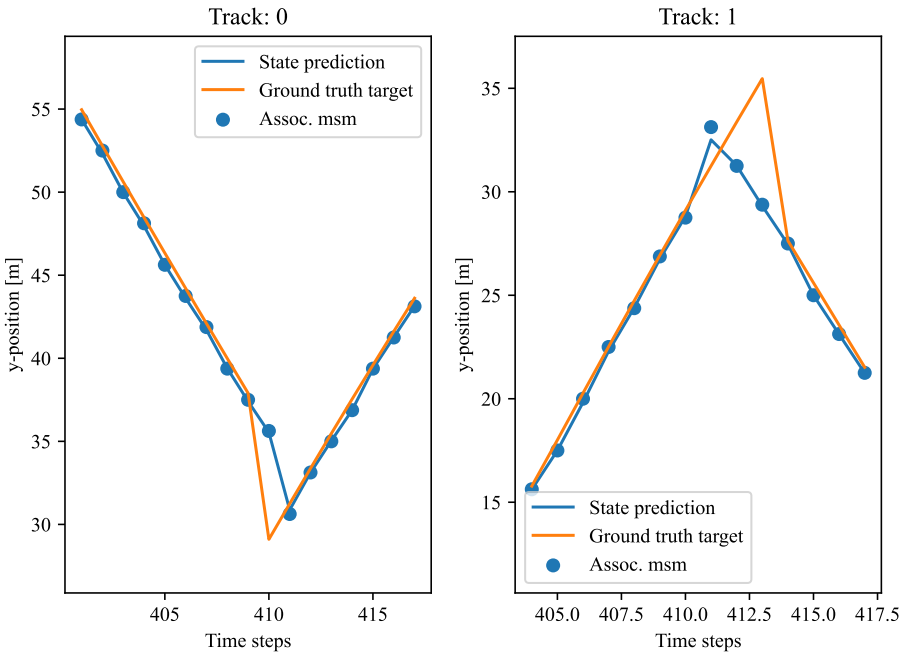


Figure 4.17: An example of the result of our object tracking in y-position, with the target y-position in orange, the estimated y-position in blue, and the associated measurements as blue dots.

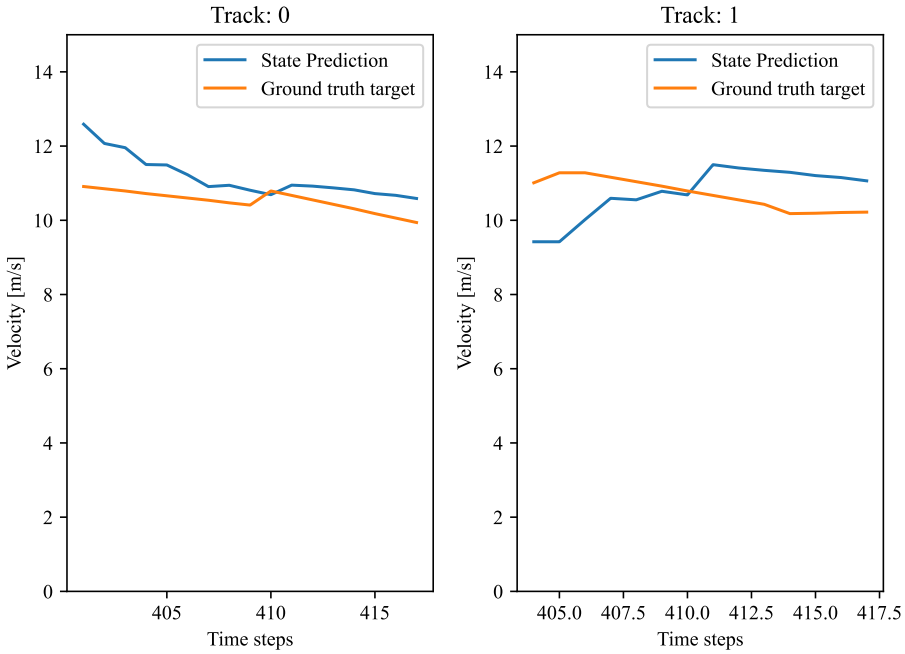


Figure 4.18: An example of the result of our object tracking algorithm in velocity, with the target velocity in orange, the estimated velocity in blue.

The distributions of errors are displayed in the plot below. Figure 4.19 shows that the distribution of x-position errors has a mean of -0.124 and a standard deviation of 0.878.

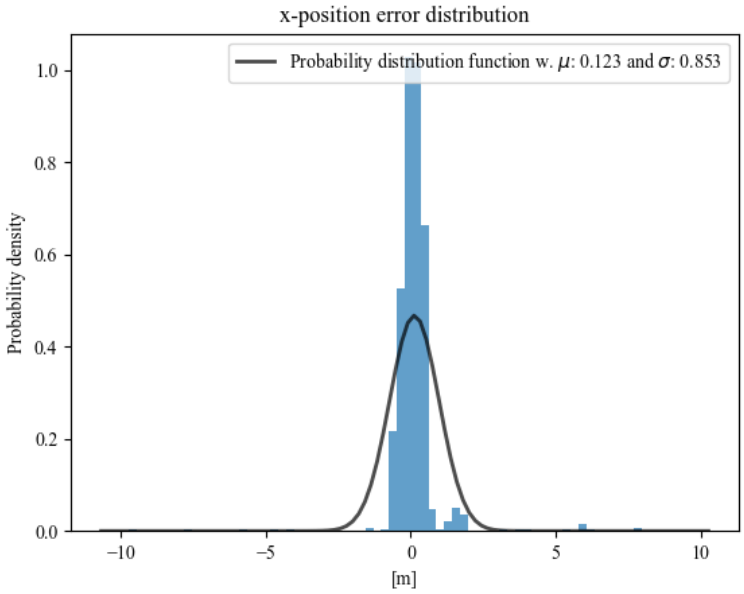


Figure 4.19: Distribution of errors in x-position, with mean -0.124 m and standard deviation 0.878 m.

Figure 4.20 shows that the distribution of y-position errors has a mean of -0.119 m and a standard deviation of 1.624 m.

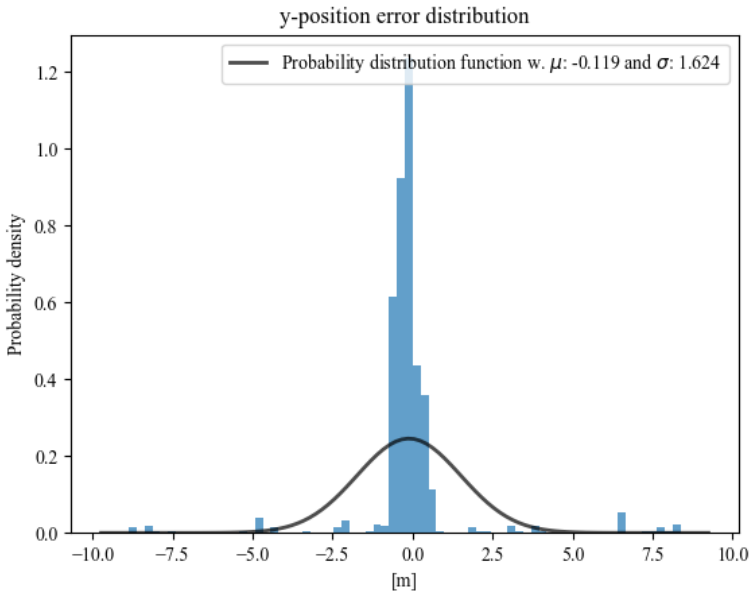


Figure 4.20: Distribution of errors in y-position, with mean -0.119 m and standard deviation 1.624 m.

Figure 4.21 shows that the velocity error distribution has a mean of -0.47 m/s with a standard deviation of 2.13 m/s.

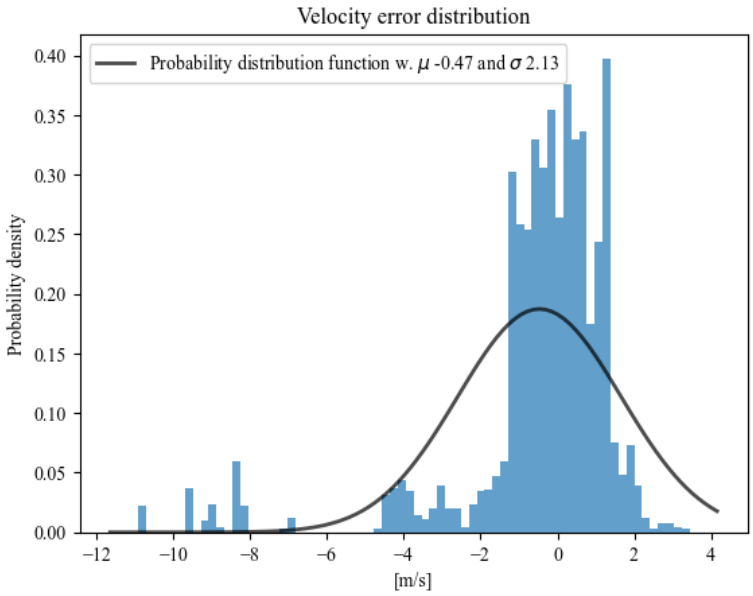


Figure 4.21: Distribution of errors in velocity with mean -0.47 m/s and standard deviation 2.13 m/s.

Figure 4.21 shows that the heading error distribution has a mean of 0.008 radians and a standard deviation of 0.789 radians. This corresponds to 0.458 and 45 degrees.

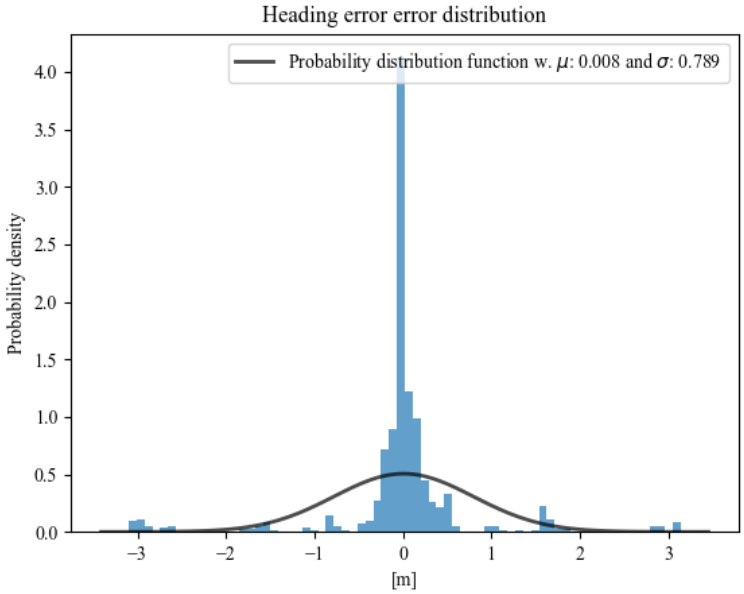


Figure 4.22: Distribution of errors in heading with mean 0.008 radians and standard deviation 0.789.

The plot 4.21 shows that the heading error distribution has a mean of -0.013 radians/s and a standard deviation of 0.401 radians/s. This corresponds to 0.7452 and 22.10 degrees.

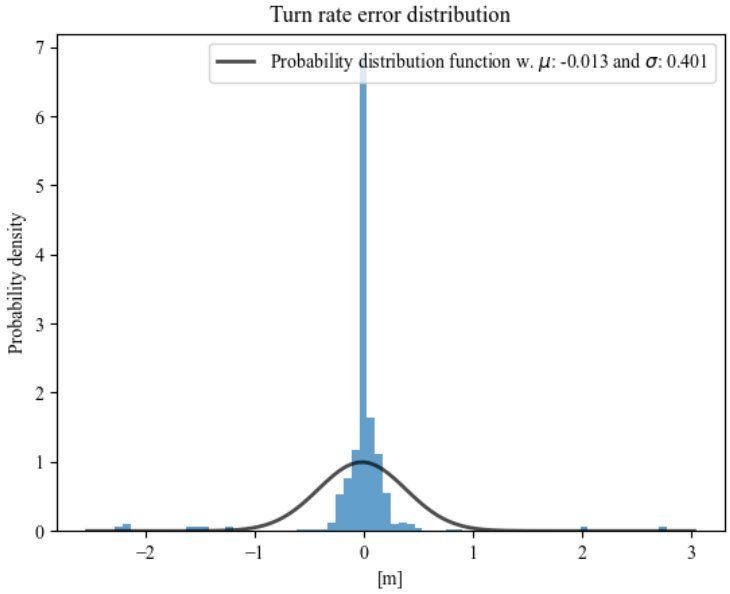


Figure 4.23: Distribution of errors in turn rate with mean -0.013 radians/s and standard deviation 0.401 radians/s.

From Figures 4.19, 4.20, and 4.21, we can see that there is a larger spread in y than in x. The velocity histogram 4.21 has a surprising spike between -8 and -10. This means that the tracking algorithm overestimates the objects' velocities. The plots below show the absolute errors and their standard deviations over time. Note that the plots should not be confused with a plot of the Kalman filter's convergence over time. The number of time steps refers to the time of the scenario, not the specific track. So a track with a corresponding Kalman filter can be initialized at timestep 14, and its initial error will be counted at timestep 14, not 1. Figure 4.24 shows the distribution of AED in position over time. The error is continuously low, with an increase at around 12 timesteps.

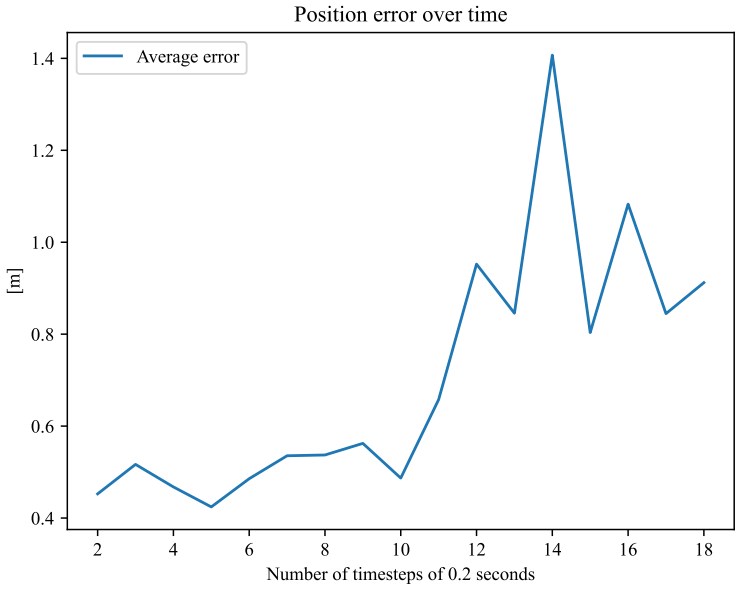


Figure 4.24: Distribution of Euclidean distance between target and estimated position

Figure 4.25 shows the distribution of absolute error in velocity over time. The error is of similar size, around 1.2 m/s, over all timesteps, but increases towards the end.

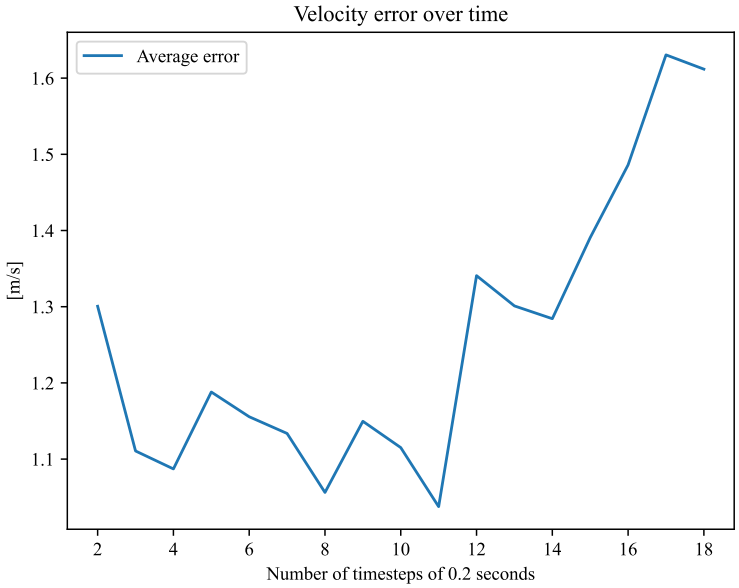


Figure 4.25: Distribution of absolute velocity error over time

Figure 4.26 shows the distribution of absolute error in heading over time. The error increases at around 10-13 timesteps and decreases again after around 17 timesteps.

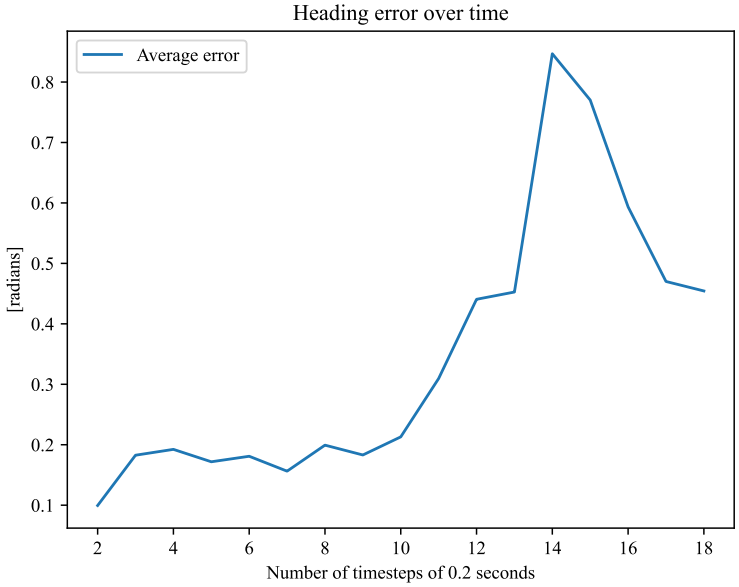


Figure 4.26: Distribution of absolute heading error over time

Figure 4.26 shows the distribution of absolute error in turn-rate over time. The error increases over time, with some jumps at the end of the plot.

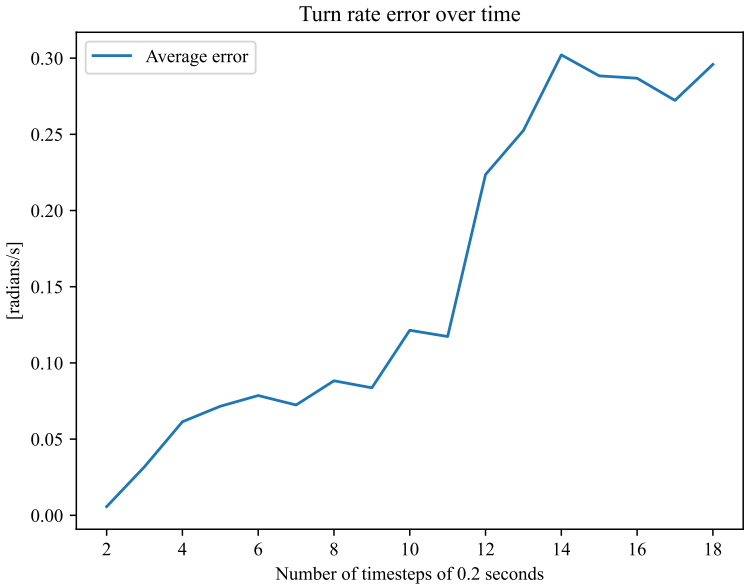


Figure 4.27: Distribution of turn rate error over time

The following error analysis is focused on the position and velocity states since they are the main focus of this thesis.

In Table 4.13, errors are presented for the different types of scenarios according to Table 3.2. The LOS label refers to scenarios with only targets in line-of-sight for the radar. The NLOS label refers to scenarios with only targets not in line-of-sight for the radar. LOS + NLOS refers to a combination of the previously mentioned. The position and velocity errors are similar for NLOS and LOS and the highest for NLOS+LOS.

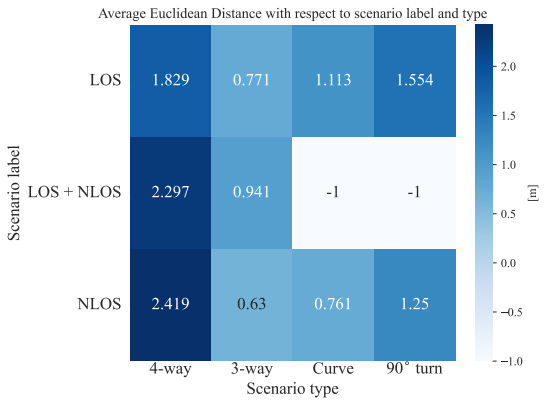
Table 4.13: Table of errors for different scenario labels

Scenario Label	Mean Absolute velocity error	Average Euclidean Distance
LOS	1.15 m/s	0.73 m
LOS + NLOS	1.37 m/s	1.01 m
NLOS	0.94 m/s	0.71 m

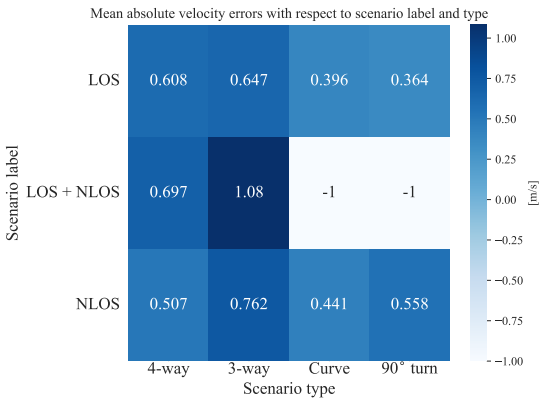
Table 4.14 shows the velocity and position errors for the simulated scenario types. The velocity error is the largest for four-way scenarios, and the positional error is the largest for three-way scenarios.

Table 4.14: Table of errors for different types of scenarios

Scenario type	Mean Absolute velocity error	Average Euclidean Distance
four-way	1.72 m/s	0.65 m
three-way	0.79 m/s	0.94 m
Curve	0.94 m/s	0.47 m
90° Turn	1.58 m/s	0.50 m



(a) Heatmap of Average Euclidean distance with respect to scenario label and type. The largest error is 2.13 m for LOS + NLOS and 4-way scenario. The smallest error is 0.58 m for NLOS and Curve scenario.



(b) Heatmap of Mean absolute velocity error with respect to scenario label and type. The largest error is 1.31 m/s for LOS + NLOS and 3-way scenario. The smallest error is 0.347 m/s for LOS and 90° scenario.

Figure 4.28: Heatmaps over errors in distance and velocity for the tracking algorithm. For scenarios Curve and 90° turn, there are no scenarios with LOS and NLOS. These are instead marked with a -1 in the plots.

The positional and velocity errors with respect to distance are presented in Figures 4.29 and 4.30, respectively.

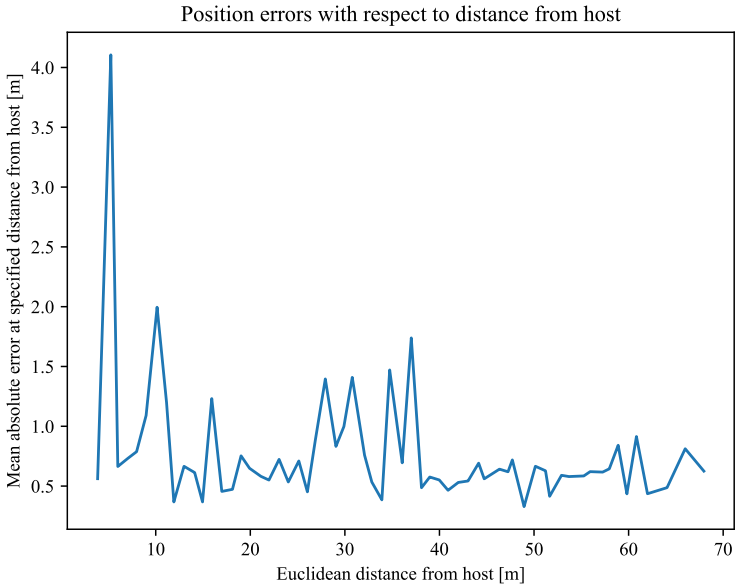


Figure 4.29: Mean positional error with respect to distance from the host, sampled with 1m resolution

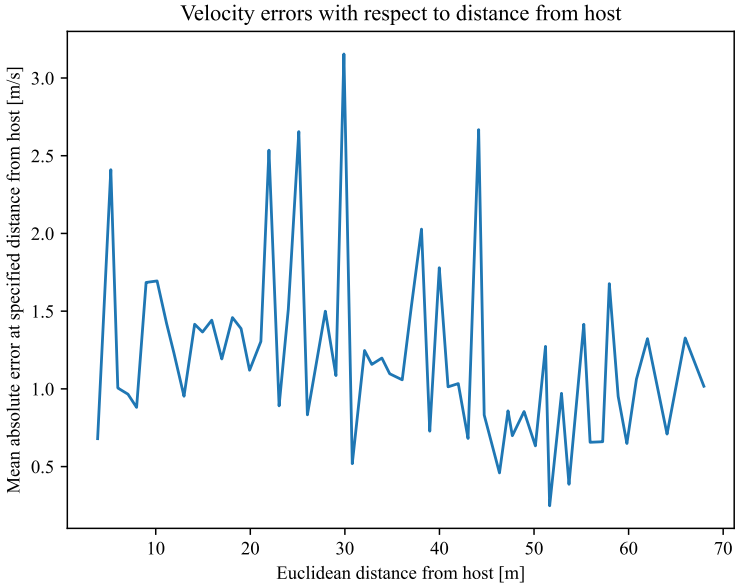


Figure 4.30: Mean velocity error with respect to distance from the host, sampled with 1m resolution

5

Discussion

The thesis has examined the application of machine learning models to identify and determine the position of NLOS and LOS targets. The results were presented in Chapter 4, and in this section follows a discussion of them.

5.1 Data simulation

When simulating a dataset for a machine learning model, the data-simulation needs to be planned to represent the problem the model should solve. In the dataset used, described in Section 4.1, the distribution of the scenarios is done to get more timeframes from the scenarios with more variation. In Table 4.1, the distribution shows that both four-way and three-way have four times more timeframes compared to $90°$ turn and curve. The reason for the large discrepancy in timeframes is the difference in the amount of variation between the scenarios. The most impactful difference is the number of targets and the number of houses.

An even split between the different scenarios could lead to better learning of the scenarios with smaller variation but may lead to worse performance on the scenarios with large variation. Too many of the same type could lead to increased accuracy for the models without the model learning to solve the general problem.

The scenarios used for the dataset were chosen to contain at least one timeframe with NLOS or LOS + NLOS. The reason was to get as many NLOS timeframes as possible. In figures 4.1 and 4.2, which shows a heatmap with the number of timeframes for each scenario label and type, it can be seen that all possible combinations of scenario labels and types exist in both the training and test dataset. Now, the LOS + NLOS scenario labels do not exist for Curve, and $90°$ turn, since these are not possible combinations. These scenarios only contain up to one target. Since both the train and test datasets represent all the combinations, the model has the possibility of learning to solve the problem.

5.2 Machine learning classification

The model structure used for the classification problems was a convolutional neural network (CNN). A simpler model was used since the problems were simpler classification

problems. The model showed good results in label classification, indicated in Figure 4.3, and in the number of targets classification, indicated in Figure 4.4. The model structure was not developed further since the model's performance was deemed good enough for both classification tasks.

5.3 Machine Learning positioning

The results of the CNN, U-Net, and U-Net 2t for the positing task are discussed in this coming section.

5.3.1 Results

Predicting target positions using radar detections as input works well, as seen in Section 4.2.3. All model types showed good results and achieved good accuracy, as seen in Table 4.7. U-Net and CNN achieve similar accuracy, and U-Net 2t achieves lower accuracy. The models may produce similar outcomes due to their shared layer composition. Both U-Net and CNN use convolution as the primary layer for extracting features, and given the results, the models seem to extract similar features. However, there are differences between the model structure and the results. The differences in model structure is discussed further in Section 5.3.2.

While the models achieve similar accuracy, a difference can be seen in the distance error, Mean Squared Error (MSE), and Average Euclidean Distance (AED). The MSE values in Table 4.8 and AED values in Table 4.9 show that U-Net achieves a significantly better result than the other models. The reason for the large difference can be seen in Figures 4.5 - 4.7, where histograms of the errors in positional prediction are shown. While the mean errors are similar, the standard deviation in errors is almost doubled for U-Net 2t and CNN compared to U-Net.

The underlying reason for this difference may be seen in the confusion matrices in Figures 4.11 - 4.14. The confusion matrices use a measure of 2 m Euclidean distance to determine if a model has made a correct prediction. The confusion matrices reveal that U-Net often predicts the absence of a target despite its existence, while CNN tends to predict the presence of a target when one exists. This is particularly clear in Figure 4.14, but there is a similar tendency across all scenario types. So, when the U-Net model is very uncertain of the position, it outputs that no target is found. When the CNN model is uncertain, it takes a guess at the position of the target. When the model outputs no target, it does not affect the MSE, while the uncertain guess of the CNN model increases the MSE. However, the U-Net also has higher accuracy, which would be negatively affected by the guess of no target when there is one. The false positives indicated in the MSE of CNN and Figures 4.11 - 4.14 is an undesired behavior since it can lead to, for example, unwanted emergency breaking in an end application.

Moreover, Figures 4.5 - 4.7 show that most errors are equal to or close to zero. This indicates that the model finds the object but doesn't manage to predict the exact position. To further evaluate the models, Figures 4.11 - 4.14 are reviewed again. The confusion

matrices show that the models perform better when close predictions (less than 2 m away) are considered correct compared to the accuracy.

The model's performance on a subset of the data will now be discussed. Figure 4.8 shows the position accuracy over different scenario labels. As seen in the figure, the models achieved high accuracy for all the labels in the test data. The results show that U-Net and CNN have better accuracy than U-Net 2t for all labels. The high accuracy for NLOS indicates that the models learn to position the car even without direct hits.

Considering the different scenario types, the models perform well for the four-way, three-way, and curve scenarios but have problems with the 90° turn scenario. This could be because the 90° turn scenario contains walls on all sides except one, potentially generating more direct and multi-path detections, resulting in a more difficult task for the models.

The high accuracy for the three-way scenarios in Figure 4.10 indicates that the introduced variation didn't work as expected. The scenario timeframes in the test set are either too similar to the scenario timeframes in training data, or the positioning in the scenario is too simple for the model.

The accuracy for different numbers of targets is shown in Figure 4.15. The models achieve high accuracy for zero, two, and three targets and lower accuracy for one target. The reason for the lower accuracy for one target could be that it exists in all scenarios, whereas two and three targets only exist in two and one scenarios, respectively. The different distributions of scenarios in the different number of targets affect the result, and the model is not necessarily better at positioning zero, two, and three targets than it is at positioning one target. It would rather be expected that positioning one target is simpler than two and three.

The accuracy for no car is high for all scenario types and is probably due to the range rate inputs. Because of ego-motion compensation, the input array should have only zeros in the range rate input when no moving target exists, which the model most likely learns to identify.

Note that the accuracy values measure how often the model predicts correctly for all targets in the scenario. This value can be misleading since a prediction close to the target still results in a wrong prediction. Moreover, if two out of three targets are correct, the prediction is considered incorrect in the accuracy measurement.

The number of timeframes in each category and the variation can influence the accuracy with respect to different labels and scenarios. As mentioned, This could be the case when positioning three targets, which only occurs in the four-way scenario, while the positioning of one target occurs in all scenarios. In Figure 4.15, it can be noticed that the model achieves higher accuracy for three targets compared to one target. However, this could be misleading since the distributions of scenario labels and types differ substantially between the different numbers of targets, as seen in Table 4.2c.

Given the accuracy and MSE results, the U-Net model output was chosen for the object tracking, which is further discussed in Section 5.4. The model was chosen since it

achieved the best overall accuracy (see Table 4.7) and the lowest MSE (see Table 4.8). The U-Net accuracy for each label and scenario is shown in Figure 4.10. The model successfully learns to locate NLOS targets in nearly all three-way timeframes and a majority of four-way timeframes. The model demonstrates decent proficiency in learning NLOS for curve scenarios but exhibits poor learning performance for 90° scenarios.

5.3.2 Model comparison

The three models' most significant differences are the outputs from the models and their size. The output from the CNN model is a flattened one-dimensional array compared to the grid output by the U-Net. The output of the CNN is flattened using ReLU layers. U-Net, instead, recreates the input shape using convolutional layers. This could be a reason for the difference in accuracy. The other significant difference is the size of the models. U-Net is a larger model and uses more layers compared to CNN. This could be a reason for the difference in accuracy.

The U-Net 2t was implemented to explore if more input data could be useful for the model and if it could utilize previous timeframes to predict the next. Instead, the model performed worse than the other models. A reason for the poorer performance could be that model did not have the host velocity, and therefore, the model had trouble learning how to utilize both timeframes. The inputs are, as mentioned in Section 3.1, relative to the host, which moves. Using multiple timesteps as input also reduces the amount of data that can be used to train the model because each input uses twice the size. More timesteps could be used, but due to the limiting GPU memory, this would lead to a smaller dataset, and since two timesteps didn't show promising results, this was not further explored.

5.3.3 General machine learning

The machine learning models exhibited good results for the dataset used, as discussed in Section 5.3.1. Keep in mind that machine learning models are trained based on the available data and can consequently be sensitive to perturbations in the input data. A machine learning driven car may be more sensitive to attacks by, for example, perturbation of the radar signal than an explicitly programmed algorithm.

The models are only trained on four different scenarios and their variations. If the model was tested on a completely new scenario, the results would likely not be as good. An increase in the number of targets might cause confusion for the model since it only trained on positioning three targets. The scenarios used in the dataset only have one target moving in the same direction. If multiple targets were to move in the same direction within close distance, the model could have difficulties detecting all cars.

Moreover, the input to the models contains the range rate. If the velocities of the targets were lower, the range rate would be affected, which could lead to difficulties in positioning the target.

5.4 Object tracking

Object tracking using machine learning measurements proved to work well, as seen in Section 4.3.

The object tracking was done using a simple motion model; a more advanced one may be suitable for more difficult scenarios. The tracking algorithm was simple since we only had up to three targets, which were fairly easy to discern from each other using the machine learning measurements. The association step was more straightforward since only one measurement was received per object. Moreover, the measurement quality from the machine learning was high; the AED was 0.21 m (Table 4.9), and the squared errors were 2.89 in x and 3.83 in y (Table 4.8). The high MSE (compared to AED) indicates that the model sometimes makes large errors, but on average performs very well.

In Figures 4.16 and 4.17, the associated measurements, ground truth target, and state prediction are shown. One can observe that the predicted path follows the measurements closely. This is likely due to the low measurement errors in general, 0.21m in AED (Table 4.9). When tuning the parameters, the average error was used, and the lowest was achieved by being sensitive to the measurements.

The resulting positional and velocity errors from the tracker, 0.80 m and 1.30 m/s on average, are considered a good result considering that our measurements were only of x- and y-positions, not velocity. However, more effort can and should be put into the tracking algorithm and Kalman tuning if it should work in more complicated tracking tasks. For the simulated scenarios in this thesis, this tracking algorithm was considered adequate.

Some results of the object tracking need further discussion. Figures 4.19, 4.20, and 4.21 show the distribution of errors for the x- and y-positions, as well as the velocity. The probability distribution for the y-position has a similar mean to the x-position but a wider spread. The wider spread reflects the slightly larger spread in errors shown in the machine learning results (Figure 4.5).

The velocity plot, Figure 4.21, shows a small probability of larger errors at around -10 m/s. This means that the tracker model overestimates the velocity and likely occurs when the target stops or decreases speed abruptly. The velocity estimation may have a slower change with respect to new measurements than the x- and y-positions, which could be why we see this spike in the velocity histogram but not the position histogram.

The error plots over time; Figures 4.24 - 4.27 display some differences in error over time. The smallest errors are in the beginning, which indicates that our way of estimating the first state using the Pretrack module was successful. The errors grow slightly over time in all plots. This does not necessarily mean that the tracking is poorer for later times. It might be that the tracks later in the scenario are more difficult to track. There is a larger probability of the target stopping at the end of the scenario than at the beginning since it will not risk colliding with the host or other targets until after running for a few timesteps. This is seen in the discussion of the velocity error distribution above. Moreover, detected objects may not exist in all scenarios for the late timesteps. The more complex scenarios

may have more objects at the end of the scenario, which would increase the error later in the scenarios.

The errors of the tracking algorithm were also analyzed for the different scenario labels and well as scenario types, as presented in Table 4.13 and Table 4.14. The heatmaps in Figure 4.28 show a comparison of the errors for the different scenarios types and labels.

The AED heatmap in Figure 4.28a shows larger errors for the four-way scenarios in general and the four-way scenarios with NLOS targets specifically. This does not reflect the accuracy of the machine learning measurements given in the heatmap in Figure 4.10. This may be because the accuracy measure does not necessarily translate well into a distance measure. Moreover, the measurement association step can filter out invalid measurements from the machine learning model. This means that outputs from the machine learning model can bring down the accuracy of the model without affecting the AED in the tracker. Incorrect associations may cause high errors for LOS + NLOS and NLOS. The four-way scenario allows up to three targets, which can be within a close distance of each other at their closest point. This means that the association of measurements or targets may be incorrect, causing a larger error.

Another surprising result when comparing to the heatmap over accuracy is that the curve NLOS performance is high in the tracker but has low accuracy in the machine learning model. Figures 4.11 - 4.14 indicate how the number of measurements that are within 2 meters from the host may give a better indication. For the curve scenario, this evaluation metric (Figure 4.13) can explain the relatively low error for the tracking algorithm in this scenario type. The number of predictions that are more than 2 meters away from the target is comparatively low. The number of erroneous predictions of 90° turn is, however, comparatively high, as seen in Figure 4.28a, which is reflected in the U-Net accuracy in that scenario (Table 4.10).

The heatmap of mean absolute velocity error 4.28b errors differ from the heatmap of positional errors position plots. Since the tracker receives the positions as measurements and the velocity is estimated using a motion model, there can be some differences in which scenarios perform best.

Thereafter, the positional and velocity errors with respect to distance from the host were analyzed. Figure 4.29 displays a lower positional error with increasing distance from the host. This may be a surprising result but can be explained by the nature of the simulated scenarios. NLOS can only occur in close proximity to the host, and LOS allows us to detect targets further away. This means that all targets more than 40 meters away are in the LOS of the host, where we, in general, have higher accuracy than in NLOS, as seen in Table 4.10. A similar behavior, but not as distinct, can be observed in Figure 4.30.

5.5 Future work

The application of machine learning on simulated scenarios indicates that CNN-based machine learning models have the ability to detect and position NLOS objects. However, machine learning is limited to the training data it has already seen. For a model to work in reality, a vast amount of real-world data with ground truth would be needed. Ground truth labeling can be done with, for example LIDAR data in LOS scenarios. In NLOS scenarios, labeling may need to be done manually, which is very arduous. This is a limitation when applying machine learning to multi-path radar data in the real world. In addition, the simulation scenarios are also more well-behaved than data in the real-world.

Recent research attempts in other areas use a large simulated dataset to train a machine learning model and a small real-world dataset to adapt it to real-world situations [31]. A similar approach to automotive radar machine learning generally, and in the multi-path setting specifically, could be interesting.

6

Conclusion

This report shows that radar target positioning in multi-path and multi-object scenarios using machine learning is successful. Furthermore, the suggested U-Net model outperforms a simple CNN model.

The U-Net performs well in NLOS, LOS, and LOS + NLOS scenarios. However, the NLOS scenarios generally lead to slightly lower model accuracy. The 90° turn is the most challenging scenario for all the models, likely because it can generate more radar detections, both multi-path and direct, than the other scenarios. On the other hand, the three-way scenario has very high accuracy, indicating that it was remarkably straightforward for the model to learn. The high accuracy for the three-way scenario suggests that the efforts to introduce variation in the three-way scenario were not adequate and that the model could have overfitted.

With this, the thesis emphasizes the importance of a well-planned data simulation procedure in machine learning applications. By considering the distribution of situations and incorporating essential elements such as NLOS timeframes, we can improve the accuracy and applicability of the models.

Moreover, using the outputs of the machine learning model as measurements in a Kalman filter-aided tracking algorithm was successful. A random-search tuning of the Kalman filter had good results in terms of distance and velocity errors. In addition, the tuning highlighted the accuracy of machine learning measurements since it resulted in the Kalman filter states following the measurements for x- and y-positions.

The overall positive result indicates that a similar approach could be successful in real-world scenarios. The challenge is that this type of machine learning training requires a large amount of labeled data. While more straightforward radar scenarios can use, for example, LIDAR data, to automatically label the data, this is challenging in multi-path scenarios. For the work of this thesis to generalize to real-world scenarios, some way of automatically labeling NLOS objects is needed.

Bibliography

[1] Transportstyrelsen, "Statistik över vägtrafikolyckor," November 15 2022 [Online]. Available: https://www.transportstyrelsen.se/sv/vagtrafik/statistik/olycksstatistik/statistik-over-vagtrafikolyckor/, accessed: 2022-12-15.

[2] W. H. Organization, "Global status report on road safety 2018." Available: https://www.who.int/publications-detail-redirect/9789241565684, accessed: 2022-12-15.

[3] V. K. Kukkala, J. Tunnell, S. Pasricha, and T. Bradley, "Advanced driver-assistance systems: A path toward autonomous vehicles," *IEEE Consumer Electronics Magazine*, vol. 7, no. 5, pp. 18–25, 2018.

[4] O. Rabaste, J. Bosse, D. Poullin, I. Hinostroza, T. Letertre, T. Chonavel, *et al.*, "Around-the-corner radar: Detection and localization of a target in non-line of sight," in *2017 IEEE Radar Conference (RadarConf)*, pp. 0842–0847, IEEE, 2017.

[5] M. N. de Sousa and R. S. Thomä, "Enhancement of localization systems in nlos urban scenario with multipath ray tracing fingerprints and machine learning," *Sensors*, vol. 18, no. 11, p. 4073, 2018.

[6] K. Yu and Y. J. Guo, "Improved positioning algorithms for nonline-of-sight environments," *IEEE Transactions on Vehicular Technology*, vol. 57, no. 4, pp. 2342–2353, 2008.

[7] F. Kraus, N. Scheiner, W. Ritter, and K. Dietmayer, "Using machine learning to detect ghost images in automotive radar," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1–7, IEEE, 2020.

[8] N. Scheiner, F. Kraus, F. Wei, B. Phan, F. Mannan, N. Appenrodt, W. Ritter, J. Dickmann, K. Dietmayer, B. Sick, *et al.*, "Seeing around street corners: Non-line-of-sight detection and tracking in-the-wild using doppler radar," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2068–2077, 2020.

[9] M. A. Richards, *Fundamentals of radar signal processing*. McGraw-Hill Education, 2014.

[10] M. A. Richards, J. Scheer, W. A. Holm, and W. L. Melvin, *Principles of modern*

radar, vol. 1. Citeseer, 2010.

[11] C. Wolff, "Radar basics: Radar basic principles," *Radartutorial. eu*, 2014.

[12] N. Karlström and A. Öqvist, "Target identification using low level radar measurements," 2018.

[13] "Automated driving toolbox," *MATLAB*.

[14] "Radar toolbox," *Radar Toolbox Documentation - MathWorks Nordic*.

[15] B. Mahesh, "Machine learning algorithms-a review," *International Journal of Science and Research (IJSR).[Internet]*, vol. 9, pp. 381–386, 2020.

[16] B. Mehlig, *Machine learning with neural networks: an introduction for scientists and engineers*. Cambridge University Press, 2021.

[17] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pp. 234–241, Springer, 2015.

[18] T. Nie, K. Deng, C. Shao, C. Zhao, K. Ren, and J. Song, "Self-attention unet model for radar based precipitation nowcasting," in *2021 IEEE Sixth International Conference on Data Science in Cyberspace (DSC)*, pp. 493–499, 2021.

[19] J. Tang, C. Chen, Z. Huang, X. Zhang, W. Li, M. Huang, and L. Deng, "Crack unet: Crack recognition algorithm based on three-dimensional ground penetrating radar images," *Sensors*, vol. 22, no. 23, 2022.

[20] M. Stephan and A. Santra, "Radar-based human target detection using deep residual u-net for smart home applications," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pp. 175–182, 2019.

[21] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[22] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," *arXiv preprint arXiv:1603.07285*, 2016.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, pmlr, 2015.

[24] I. N. Da Silva, D. Hernane Spatti, R. Andrade Flauzino, L. H. B. Liboni, S. F. dos Reis Alves, I. N. da Silva, D. Hernane Spatti, R. Andrade Flauzino, L. H. B. Liboni, and S. F. dos Reis Alves, *Artificial neural network architectures and training processes*. Springer, 2017.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[26] Y. Ho and S. Wookey, "The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling," *IEEE Access*, vol. 8, pp. 4806–4813, 2020.

[27] R. Pereira, G. Carvalho, L. Garrote, and U. J. Nunes, "Sort and deep-sort based multi-object tracking for mobile robotics: Evaluation with new data association metrics," *Applied Sciences*, vol. 12, no. 3, 2022.

[28] A. Becker, *Online kalman filter tutorial* (2017). Accessed: May 17th 2023. [Online]. Available: https://www.kalmanfilter.net/default.aspx.

[29] M. I. Ribeiro, "Kalman and extended kalman filters: Concept, derivation and properties," *Institute for Systems and Robotics*, vol. 43, no. 46, pp. 3736–3741, 2004.

[30] R. Schubert, E. Richter, and G. Wanielik, "Comparison and evaluation of advanced motion models for vehicle tracking," in *2008 11th International Conference on Information Fusion*, pp. 1–6, 2008.

[31] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 4243–4250, IEEE, 2018.

A

Appendix 1

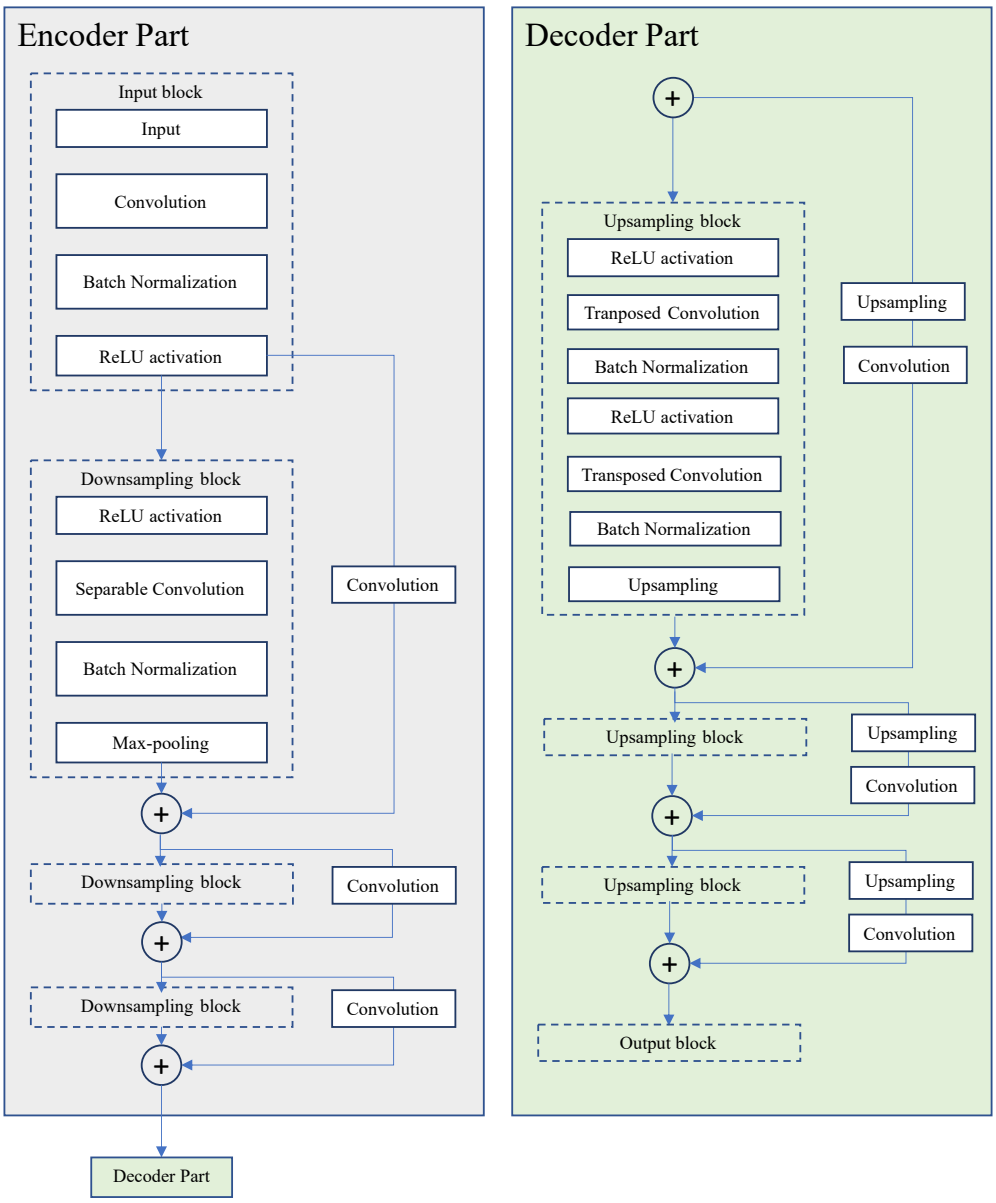


Figure A.1: Figure of U-net model used in the thesis

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY