Andreas Lebherz, BSc

# Modelling Perception Errors of Automated Vehicles using Machine Learning Methods

**Master's Thesis**
to achieve the university degree of Dipl-Ing.

Master's degree programme: Information and Computer Engineering

submitted to

**Graz University of Technology**

Supervisor
Univ.-Prof. Dipl-Ing. Dr.techn. Daniel Watzenig

Institute of Automation and Control
Head: Univ.-Prof. Dipl-Ing. Dr.techn. Martin Horn

Graz, April 2021

## Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Date

Signature

*This thesis was developed in cooperation with the BMW Group, Munich.*

# Abstract

Autonomous driving has the potential to establish an entirely new form of mobility, while utilizing existing road infrastructure. Self-driving vehicles use a variety of sensors to perceive their environment and plan their future actions. Established automotive assessment processes, such as the V-Model, have a limited scope and will not be sufficient to test every aspect of this technology. This thesis introduces a state of the art assessment process for automated driving systems, that can help to push a future market release.

Automated vehicles must be able to react properly on every occurring traffic situation, when being introduced to open streets with other road users. Therefore, extensive testing is necessary to ensure the safety of all traffic participants. Realistic and precise simulations can improve this assessment process, by replacing real-world tests with virtualized scenarios.

This work introduces a potential way to generate realistic sensor data from test cases, that can be used to simulate the real-world perception of an automated vehicle. The main focus lies in the development of a perception model that is implemented using machine learning methods. A perception model is the combination of all sensor models and the sensor fusion algorithm, that comprise the perception system of an autonomous vehicle, in a single software interface. The presented model is based on neural networks and is necessary to predict the vehicle's environment view from a test case. Further, an evaluation method to examine the performance of the implemented model in a virtualized environment is developed. As a final step, the results of this application are presented and possible extensions are discussed.

# Kurzfassung

Autonomes Fahren hat das Potenzial, eine völlig neue Form der Mobilität zu etablieren und dabei vorhandenen Straßeninfrastruktur zu nutzen. Selbstfahrende Fahrzeuge verwenden eine Vielzahl von Sensoren, um ihre Umgebung wahrzunehmen und ihre zukünftigen Bewegungen zu planen. Etablierte Prozesse wie das V-Modell werden nicht ausreichen, um jeden Aspekt dieser Technologie zu testen. Diese Arbeit stellt einen Prozess für die Bewertung von automatisierten Fahrsysteme vor, der dazu beitragen kann eine zukünftige Marktfreigabe voranzutreiben.

Autonome Fahrzeuge müssen in der Lage sein, auf jede auftretende Verkehrssituation richtig zu reagieren, wenn sie mit anderen Verkehrsteilnehmern in den Straßenverkehr eingeführt werden. Daher sind umfangreiche Tests erforderlich, um die Sicherheit aller Verkehrsteilnehmer zu gewährleisten. Realistische und präzise Simulationen können diesen Bewertungsprozess verbessern, indem sie reale Tests durch virtuelle Szenarien ersetzen.

Diese Arbeit stellt einen möglichen Weg vor, um realistische Sensordaten aus Testfällen zu generieren. Diese können verwendet werden, um die reale Wahrnehmung eines autonomen Fahrzeugs zu simulieren. Das Hauptaugenmerk liegt hierbei auf der Entwicklung eines Modells für die Wahrnehmung von Fahrzeugen, das mithilfe von Methoden des maschinellen Lernens implementiert wird. Dieses Wahrnehmungsmodell umfasst die Kombination aller Sensordaten und des Sensorfusionsalgorithmus in einer einzigen Software-Schnittstelle. Das vorgestellte Modell basiert auf einem neuronalen Netzwerk und ist in der Lage, Sensordaten aus Testfällen zu generieren. Des Weiteren wird eine Bewertungsmethode zur Untersuchung des implementierten Modells, in einer virtualisierten Umgebung, entwickelt. Als letzter Schritt werden die Ergebnisse dieser Anwendung vorgestellt und mögliche Erweiterungen diskutiert.

# Contents

# List of Figures

# List of Tables

# 1 Introduction

The European Road Safety Observatory (ERSO) states that the by far largest group of fatalities in road accidents in the European Union, are cause by car users [1]. The reason for the majority of these accidents is human misbehavior [2], [3]. Fully autonomous vehicles can help to drastically reduce this number by step-wise removing the human factor from the road [4], [5].
The introduction of Autonomous Driving (AD) brings several other advantages. A reduction of $CO_2$ emission is expected, since autonomous vehicles ease the access to car-sharing options and improve the fuel economy [6]. In addition, the recovery of personal freedom for elderly or handicapped persons, by enabling traffic participation, is another advantage of this technology [7].
The Society of Automotive Engineers (SAE) defines the 6 levels of automation to

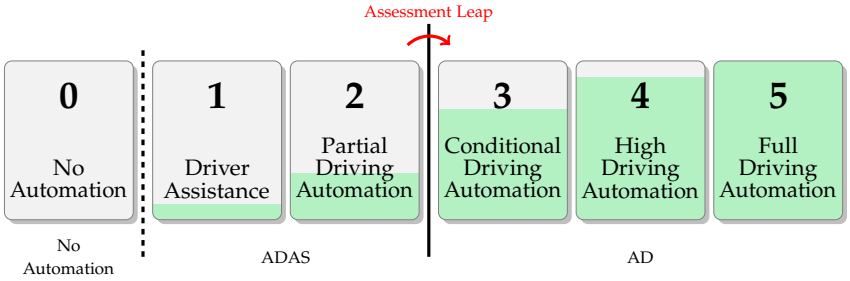



Figure 1.1: The 6 levels of automation as defined by the SAE [8]. The green parts symbolize the ratio of automation at the current level. Based on [9].

categorize the autonomous capabilities of a vehicle [8]. Vehicles in the automation level 0 (*"No automation"*) are described as fully manual vehicles, where every driving aspect is controlled by humans. Level 1 (*"Driver Assistance"*) describes vehicles with a single automated aspect, such as active Lane Keeping Assistant System (LKAS) to assist in steering or Active Cruise Control (ACC) to assist in distance control. Level 2 (*"partial driving automation"*) categorizes a combination of Level 1 functionalities. The vehicle is capable to assist in more than one domain at once such as self parking features serve as assistant in steering while also controlling the throttle and brake system. A key distinction has to be made between level 2 and level 3 (*"conditional driving automation"*). From level 2 downwards, the driver is at all time responsible for supervision of the driving task and has to intervene immediately when necessary. Level 3 relaxes the time constraint of this supervision task for specific scenarios. That means the driver is given a predefined time period in which he has to react. This distinction is critical in the assessment of associated driving functions, since it shifts the responsibility form the driver to the vehicle.
Vehicles capable of level 4 (*"high driving automation"*) take over the responsibility for the driving task. They can perform all driving functions in specified locations and/or conditions. In contrast to level 3, level 4 driving functions don't require a fall-back user in the specific domain e.g. highway. Level 5 (*"full driving automation"*) vehicles do not require any human interaction with the driving task at all. They can handle every possible traffic situation by themselves, to an extent a human driver would be able to

do. Figure 1.1 pictures the 6 levels of automation and shows the underlying driving technologies. Advanced Driver Assistant Systems (ADAS) assist the driver but don't assume responsibility of the supervision task, whereas AD functions are capable of doing so [2].
Altough the predictions for the public availability of level 3 and level 4 capable vehicles where fairly optimistic in the last decade [10], [11], the final assessment process before market release has yet to be defined. The standard norm ISO-26262 [12] specifies this procedure for ADAS with a final evaluation through real-world testing. For a specific ADAS functionality, such as LKAS, a vector space for possible test cases can be covered by a test-set due to limited scope. However, this does not account for partial or fully automated driving functions because of the complexity of the systems [9]. The amount of test cases that would need coverage tends towards infinity, which makes real-world tests over the whole test space economically infeasible [13].

## 1.1 Virtual Assessment of Automated Driving

This section serves as an introduction on the virtual assessment process as possible extension to conventional automotive assessment methods. Therefore, the V-Model [14], as an example for a conventional assessment method, is discussed and the possible benefits that come with the addition of a virtual assessment are pointed out.
Established automotive procedures, that are used for the development and integration of ADAS, are not sufficient to establish and assess fully automated driving functions [15]. The classic V-model, shown in Figure 1.2, represents a standard system development process that is used in the automotive domain. It describes the design process of functions and requirements in multiple levels of detail. The development starts at the top left corner with the definition of the requirements at vehicle level. In a step-wise approach, these requirements are refined and split into smaller components, as progressing down the path on the left side. In addition, the possible hazards and risks are analyzed and assessed at each level [16]. The technical implementation is placed at the bottom of the model, which is usually a software domain in the context of automated driving. Once the implementation is finished, the right side of the model is approached in an upward direction. Every level validates the components of the associated level and verifies the compliance with the safety requirements. The refinement of the requirements on the left side of the model would need to extend towards every possible, unforeseeable use-case to guarantee safety. Due to the structure of this process, this would lead to an inapplicable amount of required test cases, which makes it economically infeasible [17]. Using agile development methods, modern software development became extremely efficient in adapting to changes in the requirements at every stage of the development process [18]. On the contrary, the V-Model is typically not designed for that use-case. The extension of the test domain through virtual and hybrid testing, such as Hardware in the Loop (HiL) and Software in the Loop (SiL) tests, can help to reduce this drawback and increase the overall flexibility of the development process [19].

The following section summarizes the method of virtual assessment as a possible improvement for the assessment of autonomous driving and points out its key advantages over approaches like the V-Model. A distinct definition of the terms, used to describe this process, is required to avoid misconception. Ulbrich et.al. [21] formulate a *scenario* as a time-series of scenes that are characterized by a temporal correlation. A scenario can also be referred to as test case and is an unambiguous and consistent description of what happened during a test [22]. A *scene* consists of all static and dynamic information

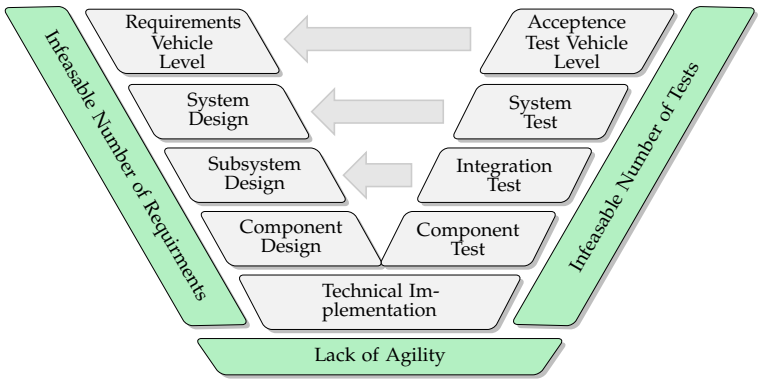



Figure 1.2: V-Model development process. The green boxes point out the problems with this approach when applied to development of ADS. (Based on [9], [20])

that is available at a certain point in time. In the following, *simulation* defines the virtual replication of a scenario that is used for the evaluation of the vehicles AD function. Further, the term *Ego* labels the vehicle that is used for the recording of test-data and the evaluation in the simulation environment. The Ego vehicle is able to perceive its environment and is equipped with an AD function.
The virtual assessment, as referred to in [9], uses the simulation of real-world scenarios and their evaluation in compliance with the ISO 26262. It uses a scenario-based approach that extends real-world tests with virtualized, slightly modified versions of that scenarios to increase test coverage. Virtual tests are parallelizable and can be executed at a high speed [23]. Ground Truth (GT) and Sensor Data (SD) can both be used as basis for the scenario-generation. GT is a fault-free representation the vehicle's environment, whereas SD represents the sensor-view of the vehicle. Because of sensor effects SD is a shifted version of the GT and contains a higher number of errors. The key assumption of this assessment process is that a scenario in the virtual domain can be used to replace a real-world test, if the experiments result can be replicated accurately enough by virtualization [9].

## 1.2 Motivation

In the previous section, a virtual assessment is outlined and its components are described briefly. The following part explains possible improvements to this procedure, by introducing a Sensor Model (SM) to the simulation. Groh et.al [22] state, that having a consistent and unambiguous scenario-description does not guarantee an exact simulation of a traffic-scenario. The simulation based on SD does not comply with the scenario-description, since at time of the recording, the AD function can only approximate the exact position of the vehicle and makes its decision base on that. On the other hand, relying only on GT data when simulating a scenario does not guarantee the same behavior of the AD function in the real- and virtual world because of the perception mismatch caused by sensor and Sensor Fusion (SF) errors [22].
In the following, the combination of all sensor models and SF into one model is labeled as Perception Model (PM) [24]. Applying a PM can recreate the sensor-view of the vehicle during the Field Operating Test (FOT), solely based on the GT information [25]. Figure 1.3 shows the closed-loop perception of an automated vehicle and the flow of information. The vehicle perceives its environment through different sensors and extracts information about its surroundings via SF. This information is further processed in the AD functions plan and act. In the simulation, the GT data is used as

input for the PM. The decision making and AD function in the simulation consist of the same software components that run on the vehicle.

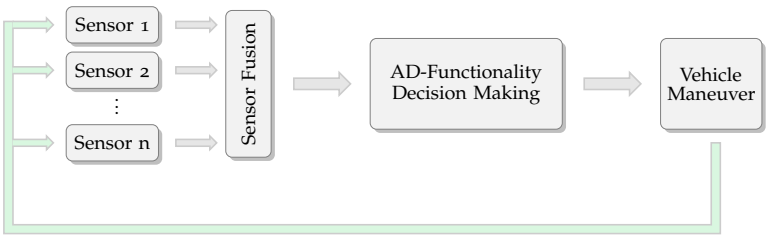



Figure 1.3: Perception path of an autonomous vehicle. The modeling of the environment is highlighted in green. (Based on [24])

## 1.3 Goals

The goal of this thesis is the development and evaluation of a PM in the context of the virtual assessment of autonomous driving. The introduced virtual assessment process uses the simulation of scenarios to extend the spectrum of real-world tests with virtualized versions. The implemented PM should be able to reproduce the sensor-error that is caused by the onboard sensors of the vehicle, while recording the test case. The sensor-error $\Delta S$ represents the difference between the GT and SD and can be expressed as

$$\Delta S = GT - SD. \tag{1.1}$$

The task can be interpreted as identification of a nonlinear system that uses GT data as input to approximate SD. This leads to the formulation of the first research question of this thesis:

**(1)** Is a perception model capable of offline reproducing the sensor-error $\Delta S$?

Using the PM in the simulation environment for the virtual assessment should increase the accuracy of the process. The model will be provided with GT data and has to output the predicted SD accordingly. After this, the performance of the simulation is evaluated and compared to the real-world test case. The second research question formulates therefore to:

**(2)** Does the inclusion of a perception model improve the quality of the virtual assessment process?

To answer those questions, this thesis shows the implementation of a PM and the integration of this model into the provided simulation environment.

## 1.4 Structure of this Thesis

The rest of this work is structured in the following way. Chapter 2 shows the methodology that is used to accomplish the stated results and explains the experimental setup. An explanation of the necessary background information, needed to understand the full scope of this thesis is provided in Chapter 3. In chapter 4, parallels to relevant scientific publications are drawn and the practical part of this work is explained in depth. Finally, chapter 5 highlights the results of the experiments and chapter 6 concludes the thesis and points out possible extensions to this work.

# 2 Methodology

This chapter describes the research methodology that is applied in this thesis. Figure 2.1 shows all methodological parts that are applied and highlights implementation tasks. Blocks that describe literature research and the general framework are labeled as gray. Blocks that involve implementation or integration tasks are colored in green and make up the main tasks of this work. Arrows symbolize dependencies, where gray describes a knowledge dependency and green an implementation dependency.

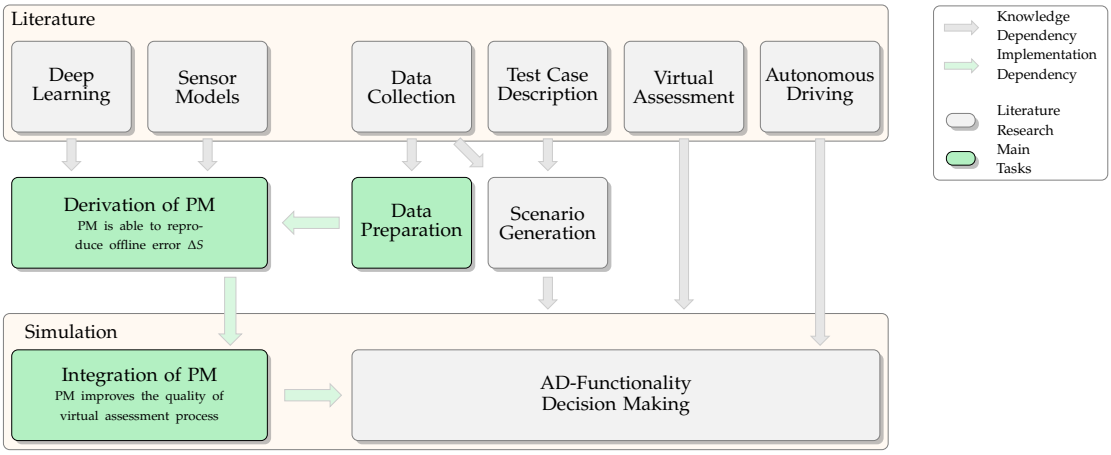



Figure 2.1: Block diagram of the applied research methodology.

The top row consist of blocks describing the literature research that is necessary to understand the concepts of this project. Focus thereby lies on the predefined assessment process, as well as the fundamentals for the derivation of the PM. The bottom row highlights blocks involved in the simulation chain and their dependencies.
A short introduction to deep learning and SMs is done in chapter 3. Hereby the main focus is set on processing sequences using Neural Networks (NNs). In addition, nonlinear-system identification is explained briefly. The introduction to virtual assessment for the evaluation of Autonomous Driving Systems (ADS) started in chapter 1 and continues in chapter 3. Main focus is the assessment process itself, as well as the scenario-description. Further, chapter 3 describes how data is collected and how the generation of scenarios is solved by reprocessing the collected data.
The derivation of a PM, that is based on machine learning methods, addresses the first research question of this thesis. The task is to derive a network structure that is able to learn the sensor-error $\Delta S$ from collected data and predict accurate sensor values when fed with GT information. The integration of this PM into the provided simulation framework attempts to answer the second research question of this thesis. Therefore an open-loop evaluation is performed to showcase if the accuracy of the simulation can be improved by this addition.

# 3 Background

This chapter serves as introduction to the topics that are covered in this thesis and provides the necessary background form different fields of study. It starts by explaining the identification of unknown systems and provides a brief introduction on machine learning. Further, a closer look is taken on the virtual assessment process and sensor models are explained.

## 3.1 System Identification of Black Box Systems

The identification and modeling of an unknown system forms the basis of the first research question of this thesis. Figure 3.1 shows a system $S$ with its input $u(t)$ and output $y(t)$. System identification describes the process of finding the dynamics of an unknown system, given only its inputs and corresponding outputs [26]. *White box* models are derived by first principle differential equations, such as physical laws and require knowledge about the systems composition and its parameters. In case no prior knowledge exists about a model and the description is based solely on measurements, the model is referred to as *black box*. A combination of those two classifications is labeled as *gray box* model and can be characterized using a combination of first principle equations and measurements [27]. This chapter sets its main focus on the identification of unknown black box systems.

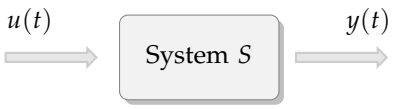



Figure 3.1: Unknown System $S$ with inputs $u(t)$ and outputs $y(t)$. (Based on [27])

An unknown system can be classified by the relation of its inputs to outputs into linear and nonlinear and further by its system description into parametric and nonparametric systems [27]. In the book by Stephen A. Billings [26], a nonlinear system is defined as any system that does not fulfill the superposition principle. This theorem states that the response of a system caused by a sum of stimuli, corresponds to the sum of responses that would have been caused by each stimulus individually. The theorem can be reformulated as principle of additivity (eq. 3.1) and homogeneity (eq. 3.2)

$$S(u_1 + u_2) = S(u_1) + S(u_2), \tag{3.1}$$

$$S(cu) = cS(u), \tag{3.2}$$

where $S(\cdot)$ represents a linear system with inputs $u$ and $c$ is an arbitrary constant. Opposed to linear systems, nonlinear system do not have a linear correlation between input and output.
A parametric system can be fully described by a finite set of parameters. For the identification of a parametric model, the underlying structure is usually known beforehand [27]. Basis for these structures are typically physical relations which require a lot of prior knowledge about the system [28]. Nonparametric systems do not assume a prior

definition of the model structure and require an infinite number of parameters for an exact description. A standard procedure for identifying nonparametric models analyzing the model output as response of a known input [26]. Finding this input-output relation is challenging when there exists no prior knowledge about the system itself. Identifying the behavior of a system is important for analyzing existing and developing new processes. An accurate system model can provide a better understanding of a systems behavior and can help to improve the efficiency of a process [27].

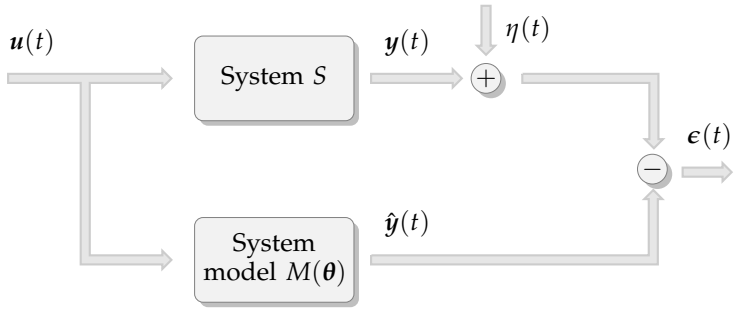



Figure 3.2: Identification of nonparametric systems. (Based on [27])

Figure 3.2 shows the process of system identification by approximating the systems behavior using a system model. The vector $\boldsymbol{u}(t) = [u_1(t), ..., u_n(t)]^T$ symbolizes the input parameters of the system. For the identification the input parameters (or a representative subset) are also fed into the system model $M$. The system model is further described by a parameter vector $\boldsymbol{\theta}$, so its approximated output can be expressed as $\hat{\boldsymbol{y}} = f(\boldsymbol{u}(t), \boldsymbol{\theta})$. The process output $\boldsymbol{y}(t)$ is usually disturbed by a source of noise $\eta(t)$ in a real-world application. For simplicity, Figure 3.2 only shows an additive source of noise $\eta(t)$. The model output $\hat{\boldsymbol{y}}(t)$ is compared to the disturbed system output $\boldsymbol{y}(t) + \eta(t)$, which yields the error signal $\boldsymbol{\epsilon}(t)$. The performance of the model is measured by the magnitude of error $\boldsymbol{\epsilon}(t)$. Minimizing $\boldsymbol{\epsilon}(t)$ by correctly adjusting the parameter vector $\boldsymbol{\theta}$ is usually the goal of a system identification process, since it leads to a better approximation of the system behavior. In order to do so, a set of fundamental design questions concerning the structure of the system model has to be answered.

- What subset of inputs is used for the system model?
- What model architecture is best suited for the problem?
- What is the necessary model complexity to approximate the system behavior?
- What are the choices for the parameters of the model?
- What methods are used for the validation of the result?

Chapter 4 describes the approach that is chosen for this thesis and provides answers for this design questions.

## 3.2 Machine Learning

Applying Machine Learning (ML) to problems that search for patterns in data has a long and successful history [29]. This section presents principles of this discipline that are applied in this thesis.

## History of Machine Learning

The section provides a brief historic roundup by pointing out milestones in Artificial Intelligence (AI) and ML, that are relevant for this thesis. The era of AI started in the year 1958 by the invention of Frank Rosenblatts basic perceptron [30] and the perceptron convergence theorem by Novikoff [31] five years later. The convergence theorem states that the perceptron learning algorithm converges in a finite amount of time for any linearly separable problem. This attribute lead researchers to make predictions about the future of AI and ML as solution to a variety of applications. Since computational capabilities were limited and high level applications such as computer vision demands huge amounts of resources, many problems were not solvable back then. This led to the public loosing interest in AI, since companies and scientists were not able to satisfy the high expectations they have set. The following 20 years became known as the first "AI winter" [32].
In the mid 1980's, the backpropagation algorithm was rediscovered and the Recurrent Neural Network (RNN) was introduced by Rumelhart et.al. [33]. His book also made the Multi Layer Perceptron (MLP) very popular, which until today is considered the standard form of a NN. With the introduction of the universal approximation theorem in 1989 by Cybenko [34], proof was found that NNs with at least one hidden layer, are universal approximators. The utilization of Graphic Processing Unit (GPU) in 2009 [35] to parallelize tensor operations brought an immense speedup to the processes of training NN and led to commercialization of the field.

## Deep Feedforward Networks

This section gives an introduction on Deep Neural Networks (DNNs) that use a feedforward architecture and explains their basic properties. Further references to topics about ML and the formal derivation of DNNs are provided in [29], [36]. Deep feedforward networks are the standard architecture of DNNs and applicable for numerous tasks [36]. The premise of DNNs is to approximate a function $f^*(\boldsymbol{x})$, by finding a mapping

$$\hat{y} = f(\boldsymbol{x}; \boldsymbol{\theta}) \approx f^*(\boldsymbol{x}), \tag{3.3}$$

where $\boldsymbol{x}$ labels the input vector of the network, $\boldsymbol{\theta}$ defines an adjustable parameter vector and $\hat{y}$ is the model output. In order to find a suitable mapping, the network needs to learn how to adjust $\boldsymbol{\theta}$ in a way, that results in the approximation of $f^*(\boldsymbol{x})$ with sufficient accuracy.

The term *feedforward* describes the underlying network structure. In a feedforward network, information flows in one direction, starting from the input layer and progressing towards the output through intermediate steps. These intermediate steps are known as *hidden layers* in the context of DNNs [36]. The model shown in Figure 3.3 shows an example of a feedforward network with a single hidden layer. The input layer is labeled as $\boldsymbol{x} = [x_0, ..., x_L]$, the hidden layer $\boldsymbol{h} = [h_0, ..., h_M]$ and the output layer $\boldsymbol{y} = [y_0, ..., y_N]$. Input and hidden units with an index 0 are set to a fixed value of 1 by default and their outgoing weight connections are commonly referred to as *bias*. These bias units do not depend on incoming connections, even when placed inside a hidden layer. The network shown in Figure 3.3 is *fully connected*, which means each cell in a layer has a connection to every cell (except the bias unit) in the successive layer. Every connection labels the scaling with a weight unit. The feedforward nomenclature strictly forbids feedback and cross connections within single layers. Feedforward DNNs are labeled as *networks* since they are a composition of different functions that are structured in an directed graph [36]. The most commonly used structure is chaining a set of functions

$f^{(1)}$, $f^{(2)}$, $f^{(3)}$ together to create a dependency between them e.g. $f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$. In that case, $f^{(1)}$ labels the first layer of the network, $f^{(2)}$ the second layer and so on. The term *deep learning* emerged from the process of stacking a number of layers on top of each other [29]. The number of stacked layers defines the *depth* of a DNN [36].

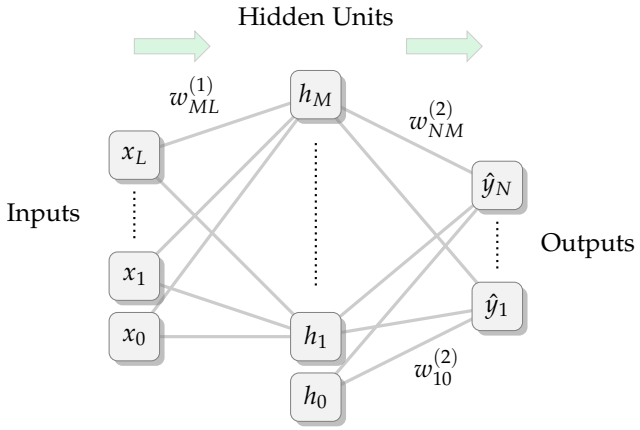



Figure 3.3: Network diagram showing the feed forward architecture of a DNN as a directed graph. The network consists of one hidden layer $\boldsymbol{h} = [h_0, ..., hM]$. (Based on [29])

*Training* a DNN describes the process of learning information from data by adjusting the network parameters and evaluating the output of a model. During training, the network tries to find the best approximation $f(\boldsymbol{x}; \boldsymbol{\theta})$ of the mapping $f^*(\boldsymbol{x})$. In *supervised* learning, the training data provides examples of $f^*(\boldsymbol{x})$, evaluated at different training points $\boldsymbol{x}^{(i)}$. Bishop [29] suggests the linear rescaling of every dimension of the input variables to a common interval such as $[0, 1]$. This avoids the increased significance that comes with dimensions of larger scale. Every example is accompanied by a corresponding label $y^{(i)} = f^*(\boldsymbol{x}^{(i)})$. Those labels specify what the output layer has to do when the network is exposed to a specific training example. However, labels do not define the behavior of the intermediate layers to achieve this mapping. Utilizing a *learning algorithm*, the network tries to find the correct adjustment of $\boldsymbol{\theta}$, that results in the a sufficient approximation of $f^*(\boldsymbol{x})$ [36]. Via *backpropagation*, the error between label $y$ and network prediction $\hat{y}$ is assigned to each weight parameter. This is done by calculating derivatives of the error with respect to each weight and then adjusting the weight with a scaled version of this gradient. The *learing rate* is a positive scaling factor for the gradient update in this context. A state of the art approach is to set the learning rate to a small constant value [36]. Other forms of training are *unsupervised* and *reinforcement* learning and not covered here. Hidden layers are called that way since they are not directly observable from the networks inputs and outputs [36]. The dimension of the hidden layers define the *width* of the network. Every layer consists of many smaller units acting in parallel, called *neurons*. The name neuron is based on their inspiration on neural brain cells. Similar to neural cells, every neuron inside a DNN receives input from many other cells and calculates its *activation*. The activation of a neuron defines its output, based on incoming data. Combining this information of every neuron in the network defines its overall output vector.

A common way to describe DNNs is to start with a model for linear regression and add parts that enable the model to overcome the limitation of strictly linear approximations [29]. A model for linear regression maps a set of multiple inputs

$\boldsymbol{x} = [x_0, ...x_L]$ to a single output $\hat{y}$

$$\hat{y} = f\left(\boldsymbol{x}^T \boldsymbol{w}\right) \approx f^*(\boldsymbol{x}) \quad \text{with } x_0 = 1. \tag{3.4}$$

$\boldsymbol{w}$ is the *weight* parameter and needs to be adjusted to find a solution for this equation. In case of linear regression, $f(\cdot)$ is the unit function. To extend the linear model and enable the approximation of nonlinear functions, $\boldsymbol{x}$ is transformed using a basic function $\phi(\boldsymbol{x}; \boldsymbol{\theta})$ and $f(\cdot)$ becomes a nonlinear activation function. The structure of the resulting model is defined by the sum of linear combinations of fixed basis functions $\phi(\boldsymbol{x}; \boldsymbol{\theta})$ in the form

$$\hat{y} = f\left(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w}\right) = f\left(\phi(\boldsymbol{x}; \boldsymbol{\theta})^T \boldsymbol{w}\right). \tag{3.5}$$

DNNs need to learn the properties of the basis function $\phi(\cdot)$ to find a suitable mapping between inputs and outputs. In the context of DNNs, $\phi(\cdot)$ resemble the hidden layers of the network. The parameters $\boldsymbol{\theta}$ are used to define $\phi(\boldsymbol{x}; \boldsymbol{\theta})$ and the weight parameters $\boldsymbol{w}$ are used to map from $\phi(\boldsymbol{x}; \boldsymbol{\theta})$ to the desired output [36]. The key idea of DNNs is to make the parameters $\boldsymbol{\theta}$ adjustable along with the coefficients $\boldsymbol{w}$ [29]. This can be done by applying Equation (3.5) to define the basis function itself. As a result, each basis function forms a nonlinear function consisting of a linear combination of inputs with adaptive coefficients. The output of the network shown in Figure 3.3 then calculates to

$$\hat{\boldsymbol{y}} = f^{(2)}\left(f^{(1)}\left(\boldsymbol{x}^T \boldsymbol{w}^{(1)}\right)^T \boldsymbol{w}^{(2)}\right). \tag{3.6}$$

### Recurrent Neural Networks

This section summarizes the properties of RNNs and highlights the differences to feedforward MLPs. RNNs are a family of NNs that are mainly used for processing sequential data [33]. The feedforward architecture of conventional MLPs is characterized by a structure that does not involve loops and for this reason only operates on fixed-sized input vectors. RNNs break with this convention and are able to work with input data that does not necessarily have to be of a fixed length. This key difference is achieved with recurrent connections that share parameters across the network [36]. In case of the vanilla RNNs, the shared parameters are the weights of the network. A recursive, dynamic system with a momentary system state $\boldsymbol{h}_t$ and a parameter-vector $\boldsymbol{\theta}$, is formulated as

$$\boldsymbol{h}_t = f\left(\boldsymbol{h}_{t-1}; \boldsymbol{\theta}\right), \tag{3.7}$$

where subscript $t$ represents the current time-step and $t-1$ the previous one. By adding a dependency on an input sequence $\boldsymbol{x}_t \in [\boldsymbol{x}_1, ..., \boldsymbol{x}_\tau]$, the reformulation of Equation (3.7) leads to the description of the hidden state of a RNN as

$$\boldsymbol{h}_t = f\left(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta}\right). \tag{3.8}$$

Figure 3.4 shows the *unfolding* of the computational graph. This representation clarifies the temporal structure of the RNN and visualizes the dependency of the previous system state for the calculation of the output. Based of Equation (3.6) the hidden state at time instance $t$ can be formulated as

$$\boldsymbol{h}_t = f\left(\boldsymbol{x}_t^T \boldsymbol{w}_x + \boldsymbol{h}_{t-1}^T \boldsymbol{w}_h\right). \tag{3.9}$$

According to Figure 3.4, the output at the same time $t$ can further be expressed as

$$\hat{\boldsymbol{y}}_t = f^{(2)}\left(f^{(1)}\left(\boldsymbol{x}_t^T \boldsymbol{w}_x + \boldsymbol{h}_{t-1}^T \boldsymbol{w}_h\right)^T \boldsymbol{w}_y\right). \tag{3.10}$$

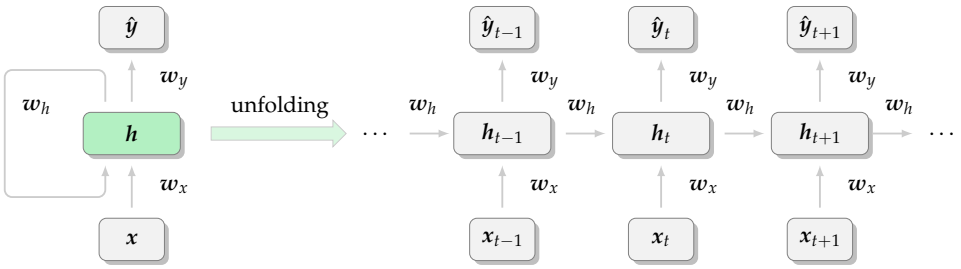



Figure 3.4: Unfolding the computational graph of a RNN. The left side shows the recurrent connection that is unfolded on the right side. (Based on [29])

RNNs use their internal state representation $h_t$ to store information on previously exposed data. The required length of the remembered history depends on the task. RNNs have limited capabilities to remember previous data due to the mapping of a sequence of arbitrary length to a fixed sized system state. Networks often have to make predictions based on long-term dependencies, as Graves [37] shows by generating sequences for different applications. In that case, the problem of *vanishing gradients* may arise, when RNNs operate on long sequences [38]. During training, the weights of a RNN are adjusted using an algorithm called Backpropagation Through Time (BPTT) [39]. Hereby the gradient of the loss function with respect to the network weights is propagated back in time. Small values in the weight matrices cause the gradient to decrease exponentially with each time step. For this reason, RNNs are often biased to prefer short-term dependencies in the training data.

## Long Short-Term Memory Networks

The Long Short-Term Memory (LSTM) network was introduced by Hochreiter and Schmidhuber in 1997 [40] to solve the problem of vanishing gradients and enable the forming of long-term dependencies in sequential data. They achieve this by maintaining a cell state that is separate from the output. Similar to other NN architectures, LSTM networks are structured in layers that consist of at leas one cell. The number of layers and cells influences the performance of the network and has to be chosen according to the task. LSTM networks utilize a structure of gated cells to selectively control the flow of information forward in time. In addition, these gated structures enable the uninterrupted flow of the gradient backwards to mitigate the vanishing gradient problem during BPTT. A gate describes a unit that is able to add or remove information to a signal, as shown in Figure 3.5. A LSTM cell is formed by combining these gated

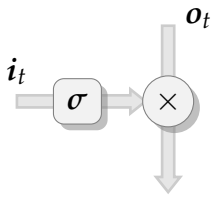



Figure 3.5: Components of a gate-cell. A sigmoid unit maps incoming data $(0,1)$ scales an output via element-wise multiplication.

structures to control the effect that previous seen information and newly added data has on the cell state. The structure of a single LSTM cell is depicted in Figure 3.6.

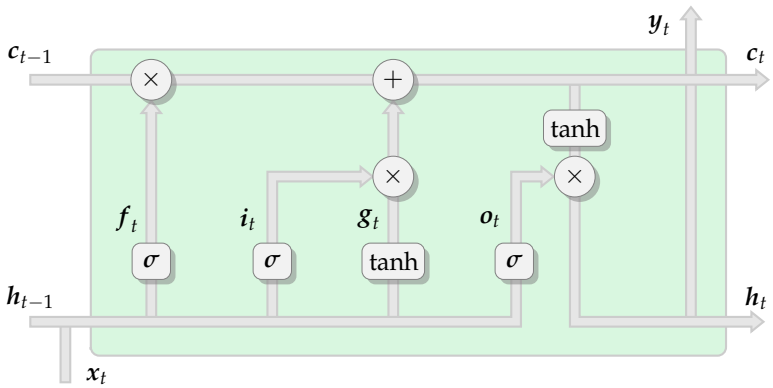



Figure 3.6: Structure of a LSTM-cell. Cell inputs are previous cell state $c_{t-1}$ and concatenated previous hidden state $h_{t-1}$ and input features $x_t$ as input. Four gate units update the cell state $c_t$ and output $\hat{y}_t$. Round blocks label element-wise operations. (Based on [41])

Training a LSTM network is similar to the training of RNNs. The update procedure affecting the forward path can be split into 4 steps:

- Forget irrelevant information (eq. 3.11).
- Store relevant, new information (eq. 3.12).
- Update the cell state (eq. 3.15).
- Output transformed cell state (eq. 3.16).

In mathematical terms, the updating of a cell state can be formulated as: $\forall t \in \boldsymbol{\tau} = [1, ..., \tau]$

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{x}_t^T \boldsymbol{w}_{xf} + \boldsymbol{h}_{t-1}^T \boldsymbol{w}_{hf}\right), \tag{3.11}$$

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{x}_t^T \boldsymbol{w}_{xi} + \boldsymbol{h}_{t-1}^T \boldsymbol{w}_{hi}\right), \tag{3.12}$$

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{x}_t^T \boldsymbol{w}_{xo} + \boldsymbol{h}_{t-1}^T \boldsymbol{w}_{ho}\right), \tag{3.13}$$

$$\boldsymbol{g}_t = \tanh\left(\boldsymbol{x}_t^T \boldsymbol{w}_{xg} + \boldsymbol{h}_{t-1}^T \boldsymbol{w}_{hg}\right), \tag{3.14}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \boldsymbol{g}_t, \tag{3.15}$$

$$\boldsymbol{y}_t = \boldsymbol{h}_t = \boldsymbol{o}_t \circ \tanh\left(\boldsymbol{c}_t\right). \tag{3.16}$$

Hereby $\boldsymbol{x}_t$ labels the input features at time-step $t$. The weight matrices $\boldsymbol{w}$ are shared across every LSTM cell state and therefore don't have a time index.

## 3.3 Virtual Assessment

This section provides a deeper insight into the virtual assessment process. Section 1.1 introduced virtual assessment as method to improve processes that are unsuitable for the assessment of autonomous driving functions, such as the V-Model. This is done by extending the test domain with virtualized versions of test cases.
The base data for a scenario is recorded by a test vehicle. SD is retrieved from the recorded onboard sensors of a prototype vehicle, whereas GT information about other Traffic Objects (TOs) is recorded with a high precision reference sensor system, such as LiDAR[24].

In the context of virtual assessment, the term scenario is used as unambiguous and consistent description of a test case. Therefore, the description of a scenario needs to fulfill a set of requirements [17], [42]

- **Restriction:** The Ego vehicle is free to perform in the described scenario in an unrestricted way after starting from a fixed condition.
- **Unambiguity:** The description of other TOs and the environment has to be unambiguous. There are no triggers, conditions or driver models involved.
- **Accuracy:** A scenario-description needs to be accurate enough, so it contains all relevant information for the AD function to base its decision on.
- **Consistency:** Describing a scenario has to be done in a consistent way, which means in case redundant information is present in the description, it has to be coherent in itself.

In their work, Wagner et al. [43] represent object traces of a scenario-description as a dynamic set of splines of cubic polynomials. Groh et.al [22] state, that a conversion from recorded data into a scenario can be based on either GT or SD. Both variants introduce a description error $\Delta$D, that will be noted as $\Delta D_{GT}$ and $\Delta D_{SD}$, respectively. Data that originates from multiple sensors can have a higher rate of uncertainty than each of the individual sensors, in case the sources provide inconsistent data [44], [45]. Since SD is comprised of information coming from different sources, it is hard to achieve a low conversion error in case of inconsistencies in the sensor readings. As opposed to this, using GT data as base for the scenario generation provides a high consistency and is therefore preferable over SD [22]. Wagner et al. [43] state that the description error $\Delta D_{GT}$ is only responsible for a small portion of the difference between real-world and simulated data. Other simplifications, such as a simplified vehicle model and an approximated environment model negatively affect the result.
A scenario needs to be unambiguously mapped to a single point in a scenario-space, which means the simulation cannot have any degree of freedom for the interpretation of the scenario, when compared to the real-world test case [9]. To justify the comparison of simulation results from different frameworks, an unambiguous description of a scenario is absolute necessary [43]. Wachenfeld and Winner [17] point out the high complexity of the scenario-space for the assessment of ADS, since it has to cover typical traffic-situations but also unforeseeable scenarios, that may arise with the introduction of autonomous vehicles. A scenario can be classified in terms of complexity, criticality and occurrence [46]. Certain traffic situations are highly unlikely but pose high risk for the safety of the occupants, whereas traffic situations with a high occurrence usually have lower risk [17].
The simulation framework is a SiL system that processes scenario-descriptions and simulates them. For this, it provides an environmental model that consists of all necessary information for the Ego vehicle to understand and current scene. Examples for the content of an environmental model are road maps, lane markings and other TOs [22]. To interpret a scene, the AD function needs to classify perceived objects and localize them inside the road model. The AD function then makes predictions about the movement of the TOs and proceeds with a decision on how to plan and act. The decision on a maneuver starts at a high level, such as lane-keeping or overtaking and proceeds on a lower level describing the desired trajectory. After that, the required actions are taken and returned back to the simulation framework. Figure 3.7 shows this behavior as a feedback-loop. The virtual assessment process compares scenarios that are recorded in different test domains, using quantities that are labeled as Key Performance Indices (KPIs) [42]. KPIs express all requirements on an AD function and are used to compare the results of FOTs and their corresponding simulations. The requirements of an AD function can be classified by measurements of accident risk,

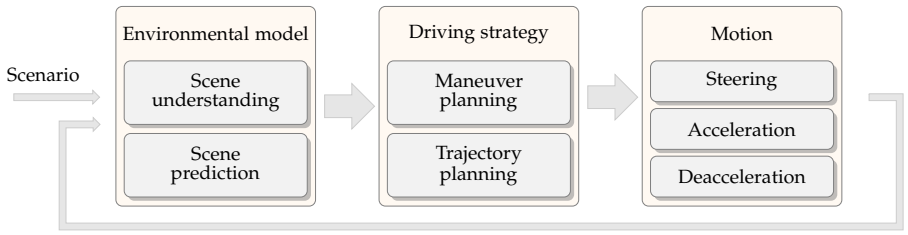



Figure 3.7: Representation of the simulation process. The environmental model is used to predict a trajectory for every TO. An action is taken and fed back into the environmental model. (Based on [42])

compliance to traffic rules and passenger comfort [9]. To compare two scenarios, binary classifications such as accident or no accident are not sufficient [22]. In the automotive domain, examples for KPIs are Time to Collision (TTC) and Time to React (TTR) [47]. Wagner et al. [48] state that uncertainties in human behavior have to be considered by the assessment method and that a solely focus on the most probable future path a TO might take does not fulfill that criteria. They therefore predict single variations around the most probable future path for every TO in the scene. Later, they use a stochastic approach based on TTR, which reveals a single risk value for every driving scene. Accumulating this risk value over the length of a scenario results in a risk over time signal.

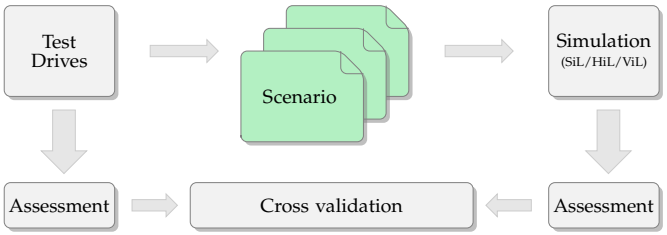



Figure 3.8: Explanation of the virtual assessment process. (Based on [9])

Figure 3.8 summarizes the virtual assessment procedure. Scenarios are generated from FOTs or Naturalistic Driving Studies (NDS) and characterized in an unambiguous way. The recorded scenarios are then simulated using in-the-loop tests. The scenarios from both paths are separately assessed via KPIs and then cross-validated. According to [9], in case the evaluated KPIs are comparable accurately enough, the simulated scenario can be used as a replacement for the real-world test. In addition, this procedure enables the creation of additional test kilometers by applying adequate small variations to the simulation of the scenario [42].

## 3.4 Sensor Models

Automated vehicles perceive the environment through sensors and therefore perceive a shifted version of the GT [46]. Via SF, information is extracted from raw sensor data and passed to the AD function of the vehicle. Based on this information, the vehicle decides on how to plan and act accordingly.
Hugh F. Durrant-Whyte [49] describes a SM as an abstraction of a physical sensing process. SMs have the purpose to approximate the ability of real sensors to extract information about their environment in terms of the information available to the sensor itself. In other words, a SM aims to reproduce the output of a real sensor with all its characteristics. SMs are used as probabilistic or quantitative approximations of real

sensors in simulated environments [49]. In the simulation, GT information coming from a scenario-description is used as input for the SM. The aim of the SM is then to reproduce SD from GT.

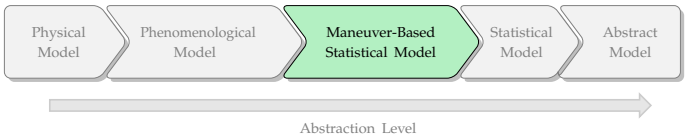



Figure 3.9: Order of a SMs according to its level of abstraction. The chosen model for this work, the *maneuver-based statistical model*, is highlighted in green. (Based on [46])

According to Notz et.al. [46], SMs can be classified by their level of abstraction. Figure 3.9 visualizes 5 types of models and sorts them by this criterion. An *abstract* SM usually simplifies the behavior of a sensor in an idealistic way, without any interfering noise [50]. *Statistical* models use statistical methods to estimate the probability distribution of an output variable [51]. A *phenomenological* model utilizes simplified physical effects and statistical properties to emulate the behavior of a sensor on signal level [52]. The *physical* model of a sensor-setup has the profoundest modeling depth. The sensor behavior is approximated by applying physical laws such as wave propagation via ray-tracing, which comes with a huge increase in computational complexity [53]. Based on the evaluation of their collected data and generation of GT, Groh et.al. [22] show a decrease of perceptual accuracy during highly dynamic maneuvers. In order to include the maneuver-based variance of the noise level and also keep the model complexity simple enough they propose the "*maneuver-based statistical model*" [46]. This model is a non-parametric approach and derives the correlation of sensor output and perceived objects in a statistical way by comparing SD and GT information.

## Soft Sensors

The term *soft sensor* combines "software" and "sensors" and refers to the estimation of unknown process variables based on mathematical models or empirical measurements. The Kalman filter [54] is a prominent model-based soft sensor for estimating the state of an object. In industrial processes, soft sensors are often proposed for estimating quality variables, in cases where online measurements are technically or economically infeasible. They can be applied as data-driven methods to model unknown black box systems, only using empirical observations of a process [55]. Soft sensors approximate quality variables in real-time, by finding patterns in related process variables, which are easier to measure. Recent publications in the field of industrial processes showed a successful application of RNNs based soft sensors on different real-world applications [56], [57]. The estimated quality variables often describe highly nonlinear, dynamic processes, which have similarities to vehicle dynamics in automotive domain.

# 4 Approach

This chapter gives an overview of the practical implementation of this thesis and provides detailed explanations of the design choices. To answer the first research question of this work, the implementation of a PM that is able to predict Model Sensor Data (MSD) from GT information, is necessary. For a successful experiment, the predicted output has to approximate the real SD with an error that is smaller than $\Delta S$. Beyond that, this thesis aims to decrease $\Delta S$ to a minimum. A data-driven black box approach is chosen for the PM and is covered in Section 4.1. The model is implemented in a Python[1] 3.6 framework, that includes the Tensorflow[2] library. The underlying operating system is Ubuntu [3] 20.04.

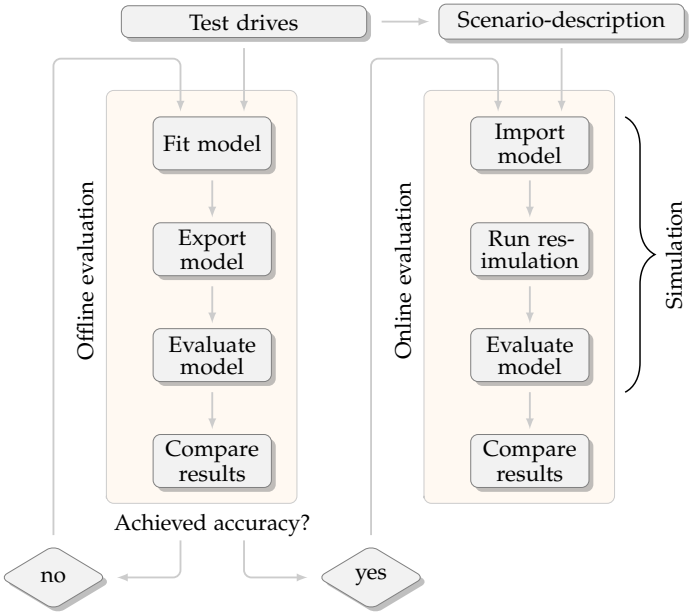



Figure 4.1: General approach of implementing and testing the PM.

For the full implementation and evaluation of the PM, a pipeline is established that is pictured in Figure 4.1. The development process starts with the analysis of the training data, which is covered in Section 4.2. The preprocessed data is the input for the offline evaluation that is shown on the left side of Figure 4.1. A configuration is chosen for the PM and during the training process, the prediction error between real-world SD and model output MSD is minimized. The trained model is exported and evaluated using a separate test-set. The evaluation results are compared to the sensor-error $\Delta S$. In case the accuracy of the model is insufficient, the training process starts from the beginning with a new set of hyperparameters, which is shown as feedback loop in Figure 4.1.

[1] `https://www.python.org/` (last access 15.02.2021)
[2] `https://www.tensorflow.org/` (last access 15.02.2021)
[3] `https://ubuntu.com/` (last access 15.02.2021)

In case the performance of the model is sufficient, the trained model is imported in the simulation framework for the online evaluation. This framework is implemented in C++ and uses the Tensorflow C++ API for importing the model. As an additional input for the simulation, the description of a scenario is loaded. While running the simulation, the PM predicts MSD in real-time. For the final evaluation, the simulation output is compared to the GT information.

This chapter describes the main contributions of this work and is structured in the following way. Section 4.1 compares the properties of the implemented PM with sensor models that are introduced in Section 3.4. Next, Section 4.2 analyzes the available training data and its preprocessing. Section 4.3 shows the application of the LSTM architecture to implement PMs and provides a connection between Sections 3.1 and 3.2. Section 4.4 highlights the properties of the training process and Sections 4.5 and 4.6 summarize two different approaches for the implementation of the PM.

## 4.1 Perception Model

This section points out the differences between the concept of a PM, which is used in this thesis and a SM, described in Section 3.4. It defines the scope of the model and points out the main advantages of a PM over individual SMs.

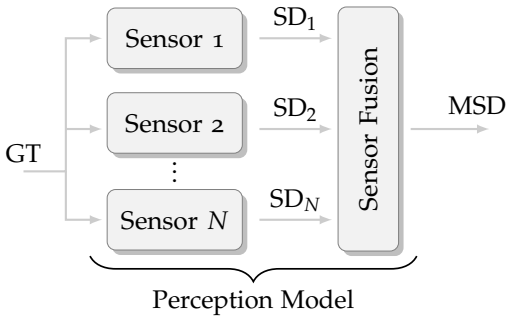



Figure 4.2: The PM is comprised of different sensors and a SF algorithm. The input is GT information, the output is labeled as MSD (Based on [24]).

Section 3.4 introduces SMs and categorizes them by their level of abstraction. The most suitable approach for this thesis is the *maneuver-based statistical model*, since it includes the decreasing perceptual accuracy that comes with dynamic driving maneuvers [22]. This decreased accuracy can be a high uncertainty of an object's position or the misjudgment of its velocity, which can result in a malfunction of the AD function, such as the delayed recognition of a critical situation. Figure 4.2 pictures a set of sensors $n = 1, ..., N$, that receive GT information, coming from a scenario-description, as input. The output of each sensor $n$ is labeled as $SD_n$ and contains sensor-errors $\Delta S$. In further processing steps a SF algorithm combines the sensor outputs $SD_{1,...,N}$ into an object-list that contains information about every perceived TO [24].
In oppose to a single SM, a PM comprises different SMs, as well as the SF algorithm. In this work, the PM covers the whole sensor setup of an Autonomous Vehicle (AV) [58]. Figure 4.2 shows that the GT information from the scenario-description is analogously used as input for the PM. The output of the model is labeled as MSD and is an approximation of the fused SD at object-list-level. The goal of the PM is to recreate SD solely based on the GT information coming from the scenario-description [24]. Its

scope includes interacting TOs during dynamic maneuvers and does not address static objects such as lane markings or static obstacles.
A major advantage of this approach over individual SMs is expected to come with the inclusion of all sensors into one model. This makes the PM independent of single SMs that originate from different suppliers [24]. In that case, a change in the sensor setup does not necessarily lead to a change in the PM. The inclusion of the SF algorithm into the PM brings an additional advantage. In case of a deviation of the simulation output during cross-verification, that might emerge from a single malfunctioning SM, the error does not have to be tracked back through the SF algorithm [24].
The sensor-error $\Delta S = GT - SD$ can be split into a static and dynamic component $\Delta S = \Delta S_{static} + \Delta S_{dynamic}$. Thereby, the static error depends on the objects position and orientation, relative to the Ego vehicle. An AV is equipped with different sensors that are mounted at several locations around the vehicle. Those sensors have varying precision, which affects the static error $\Delta S_{static}$. To evaluate the influence of the static error on the overall $\Delta S$, Sigl et al. [24] use a discretization model containing the positions around an AV based on its sensor setup. The dynamic error $\Delta S_{dynamic}$ is linked to the dynamic state of the perceived TO, such as its velocity and acceleration. Groh et.al showed a correlation of the error magnitude and the dynamic state of the vehicle [22]. For the conversion of GT data into MSD and therefore an approximation of the fused SD, the PM has to model both errors accordingly. For this, the PM receives a set of inputs that consists of static and dynamic features of the Ego vehicle and other TOs and is further described in Section 4.2.

## 4.2 Training Data Analysis

The dataset used in this thesis contains three different test-drives on German highways of approximately 60 minutes. The weather condition ranges from sunny to cloudy and the road is dry. The dataset contains recorded onboard sensors SD from a prototype vehicle and reference GT information from a high precision LiDAR sensor.

### Data extraction

To model sensor-errors during changing dynamics of TOs, the dataset is split into *traces* for further processing. A trace describes the trajectory of a TO and is dismissed if it is too short or does not contain any dynamic transitions. During a maneuver such as overtaking or lane changing, the dynamic properties of a TO change, which means the vehicle is accelerating, braking or steering [24]. Since traced vehicles do not change their dynamics for the majority of a highway drive, dismissing this data leads to a more equal distribution of dynamic and non-dynamic sections. Every traced TO in the training set should perform at least one significant change of motion. Discarding data by this criteria removes objects that are not recognized as TOs, such as traced traffic signs or guard rails. For further processing, the GT data coming from the reference systems and the SD coming from the onboard sensors of the Ego vehicle are matched to receive traces that contain both information. The effective duration of the training data after the preselection is approximately 60 minutes of single object traces. During the test drive multiple objects can be traced at once, so the extracted traces may overlap in time.

GT information is recorded with a high precision LiDAR sensor and contains relative distance $d_{rel}(GT)$ and relative heading $h_{rel}(GT)$ between TO and Ego vehicle and the Ego vehicle's pitch $p_{ego}(GT)$. In addition, the Ego vehicle uses its onboard sensors to

measure its own absolute velocity $v_{ego}(GT)$. The relative velocity $v_{to}(GT)$ of a TO is calculated from $v_{ego}(GT)$ and the derivative of the relative distance $\frac{d}{dt}d_{rel}(GT)$.
All features that are used for the implementation of the perception model are listed in Table 4.1. In this listing, quantities that are marked as SD, are derived from the onboard recordings of the test vehicle, whereas GT information originates from a reference system, with the exception of $v_{ego}(GT)$. Every quantity is based on the local coordinate frame of the Ego vehicle.

| Feature | Unit | Description |
|---|---|---|
| $v_{to}(GT)$ | m/s | Absolute velocity of the TO measured by a reference system |
| $v_{to}(SD)$ | m/s | Absolute velocity of the TO measured by onboard sensors |
| $a_{to}(GT)$ | m/s$^2$ | Absolute acceleration of the TO |
| $v_{ego}(GT)$ | m/s | Absolute velocity of the Ego vehicle |
| $d_{rel}(GT)$ | m | Absolute distance between the Ego vehicle and the TO |
| $h_{rel}(GT)$ | rad | Angle between the Ego vehicle and the TO |
| $p_{ego}(GT)$ | rad | Pitch of the Ego vehicle |

Table 4.1: Input features for the implementation of the perception model.

Further progressing, this thesis only concentrates on the $x$ component of the vehicles motion, which always points in the lateral direction of the Ego vehicle.

## Exclusion of lateral direction

Figure 4.3 shows the correlation between the $v_{x,to}(GT)$, $v_{x,to}(SD)$ and $v_{y,to}(GT)$, $v_{y,to}(SD)$ of 2.5% randomly sampled datapoints from the full dataset. The upper plot shows the correlation of the longitudinal, the lower plot the correlation of the lateral components of the same quantity.

The Pearson correlation factor $R$ [59] is a quantity that measures the statistical relationship between two continuous variables and ranges from $-1$ to 1. A value of 0 implies that there is no linear correlation, a value close to the boundaries implies perfect linear correlation between both variables.

$$R = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2(y_i - \bar{y})^2}} \tag{4.1}$$

$\bar{x}$ and $\bar{y}$ represent mean over all samples of a signal. The GT and SD components in longitudinal direction correlate more than the components in lateral direction. The omnidirectional correlation between $v_{y,to}(GT)$, $v_{y,to}(SD)$ show the weak connection of between the GT and SD components of the velocity in lateral direction. Figure 4.4 compares the lateral components of the measured velocities $v_{y,to}(GT)$, $v_{y,to}(SD)$, with the errors between GT and SD of the same quantity.

The error components $\epsilon v_x$ and $\epsilon v_x$, that are shown in Figure 4.4 are calculated as

$$\epsilon v_x = v_{x,to}(GT) - v_{x,to}(SD), \tag{4.2}$$
$$\epsilon v_y = v_{y,to}(GT) - v_{y,to}(SD). \tag{4.3}$$

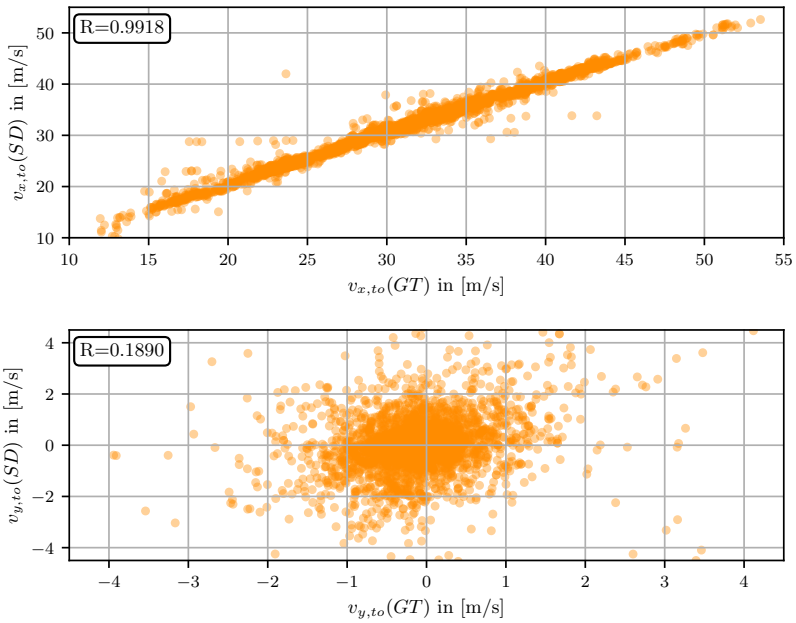



Figure 4.3: Correlation of GT and SD velocity in longitudinal and lateral direction.

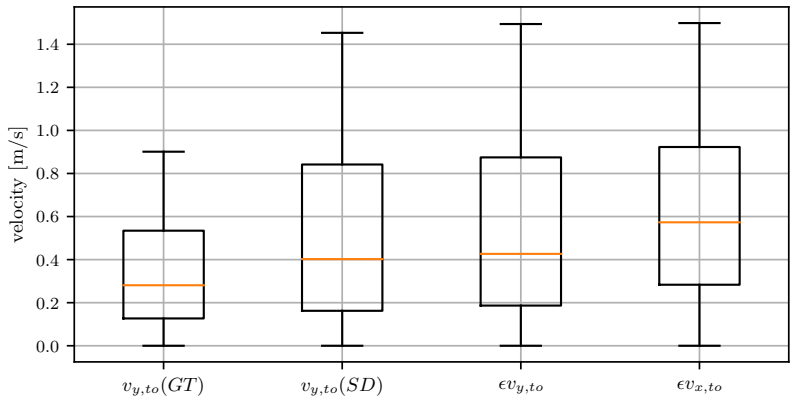



Figure 4.4: Comparison of the error between GT and SD velocity in longitudinal and lateral direction.

This representation of the error $\epsilon v_y$ between SD and GT shows that the margin of the error is in a comparable range of the actual values of those quantities. As summary for the decision to exclude the lateral component from the prediction model can be stated with two bullet points.

- The weak correlation between the GT and SD component of the velocity $v_{y,to}$
- The error magnitude $\epsilon v_y$ being in the same range as the measured values

## Restriction of distance values

An additional restriction criteria on the dataset is the relative object distance $d_{rel}$. Only objects that are within a relative distance of 120m are considered as part of the dataset. Whenever a vehicle leaves this range during a maneuver, the corresponding data is discarded. The grey area in Figure 4.5 shows the cumulative number of samples from the whole dataset as function of relative distance. The value is scaled in a range between 0 and 1. The number of samples with a smaller distance than 120m corresponds to approximately 87% of the whole dataset.

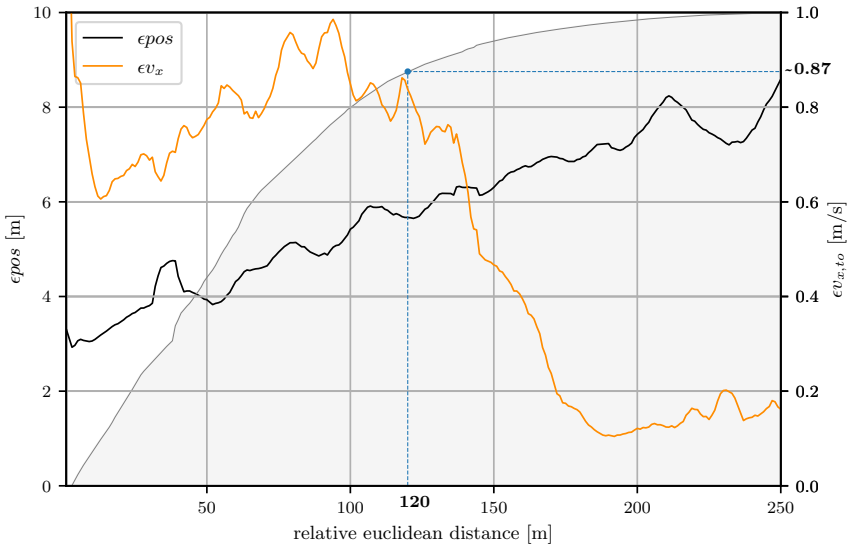



Figure 4.5: Comparison of position and velocity error.

In addition, Figure 4.5 shows the distribution of the position error $\epsilon_{pos}$ and the velocity error $\epsilon v_x$. The decrease of $\epsilon v_x$ leads to the conclusion of a biased dataset for ranges greater than 120m. This can be interpreted as the internal data association algorithm of the test vehicle being only able to match traced objects when the certainty is disproportional high. In addition, optical onboard sensors have a limited range which mostly assumes perfect weather conditions [60]–[63]. To guarantee the generalization of the implemented PM, a restriction to distance values under 120m is appropriate.

## Application specific preprocessing

To use sequential maneuvers as input for the PM, described in Sections 4.5 and 4.6, a number of preprocessing steps is necessary. In general, LSTM networks use sequential data as input vectors to make predictions about a quantity. Thereby, sequences of different length can be split into overlapping chunks of a fixed size according to Figure 4.6.

In this thesis, all input traces are split into vectors that have a length of $\tau = 25$ and overlap by 24 samples. The value of $\tau$ is derived by a set of experiments and represents 1s of data at the Ego vehicle's mean data rate of 25Hz. The input vector $\boldsymbol{x}_t = \left[x_{t-(\tau-1)}, ..., x_t\right]$ is used to make a prediction about the SD at time instant $t$.

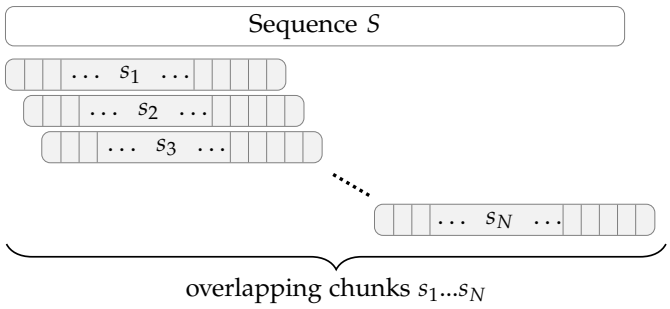



Figure 4.6: Splitting a sequence $S$ into overlapping chunks $s_1, ..., s_N$.

Every sample at time-step $t$ contains a subset of values, described in Table 4.1. For the training process, those slices are randomly shuffled. In addition, the chosen input features are linearly interpolated to an equidistant interval of 0.04s $\widehat{=}$ 25Hz to guarantee equivalent input vectors for the PM. As a further preprocessing step, every strictly positive input feature is scaled to a range of $[0, 1]$ and every other feature to a range of $[-1, 1]$.

## 4.3 Network Architecture

Applying neural networks to system identification is a well established and tested procedure [27]. Section 3.2 points out that neural networks are capable of realizing any continuous function by learning patterns in training data. Chen et al. [64] studied the application of single layer neural networks for the identification of discrete-time nonlinear systems. In their work, they prove a successful modeling of simulated and real-world systems, using machine learning methods.

Section 3.4 defines *soft sensors* as method for estimating unknown process variables via empirical measurements [55]. The concept of a PM, described in Section 4.1, can also be interpreted that way, when established as data-driven model. In their work [56], Ke et al. apply LSTM networks as approximators for highly nonlinear, dynamic, quality variables in chemical processes. In a data-driven approach, they use a two-layer LSTM and achieve improved results compared to a standard RNN approach. Yuan et al. [57] apply LSTM networks to the identification of a nonlinear, dynamic process. To factor in the high temporal correlation of the process variables, they include the output of the intermediate LSTM layers as inputs of their successors.
Using LSTM networks for generating sequences is a widely applied field. Alexander Graves [37] applies LSTM networks to the generation of different complex sequences, such as text and online handwriting, by predicting one datapoint at a time. Salinas et al. [65] implemented a framework for time series prediction called DeepAR. In their approach they predict the parameters of a normal distribution $\sigma_t, \mu_t$ and draw samples $\hat{y}_t$ from it at every time step $t$. They further use $\hat{y}_t$ as additional input for the successive time step $t + 1$, which they label as *autoregressive* approach.

This work uses a LSTM network to implement a PM, which is used in the virtual assessment of AD functions. Literature suggests that the LSTM network structure is suitable for system identification and time series prediction problems [56], [57], [65]. Further, this thesis uses inspirations from aforementioned literature to answer the first research question outlined in Section 1.3.

The goal of the implemented network is to predict MSD at every time step $t$, that fulfills $MSD_t \approx SD_t$. Thereby the difference between MSD and SD has to be smaller than $\Delta S$, as a minimum requirement for the PM. The predicted MSD estimates the longitudinal velocity of a tracked TO. Figure 4.7 shows the abstraction of the network model with its input and output.

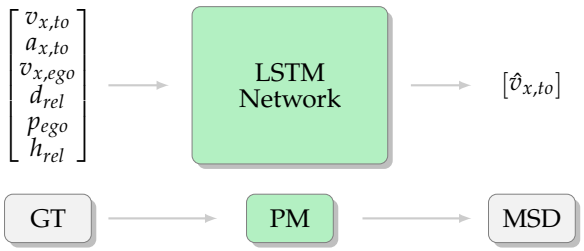



Figure 4.7: LSTM network as PM implementation for the prediction of MSD.

For the prediction process, the network uses previous samples $[t-\tau, ..., t]$ to update its cell states and output MSD. In addition, the network is supposed to approximate the dynamic noise behavior of the SD.
The proposed network architecture uses a feature vector $\boldsymbol{x} = [x_{t-\tau}, ..., x_t]$ as input to predict the components of a probability distribution $\hat{\mu}_t, \hat{\sigma}_t$. Further, $\hat{\mu}_t, \hat{\sigma}_t$ are used to draw samples from this distribution. This procedure implies the assumption of a Gaussian distributed error around the predicted MSD. In their work, Sigl et al. [24] model the perception error of an AV using a Gaussian distribution, as it fits the measured distribution of the sensor values. To prevent numerical instabilities, that arise with very small or negative values for the variance, $\hat{\sigma}_t$ is transformed to

$$\hat{\sigma}'_t = \ln\left(1 + e^{\hat{\sigma}_t}\right). \tag{4.4}$$

This transformation does not affect the prediction process itself but ensures numerical stability, since it avoids negative values for the variance. After this transformation, the probability distribution is formulated using Gaussian properties

$$\hat{y}_t = \mathcal{N}(\hat{\mu}_t, \hat{\sigma}'_t) = \frac{1}{\hat{\sigma}'_t\sqrt{2\pi}}\, e^{-\frac{1}{2}\left(\frac{\hat{y}_t - \hat{\mu}_t}{\hat{\sigma}'_t}\right)^2}, \tag{4.5}$$

where $\hat{y}_t$ is the sampled prediction value for time step $t$.
This thesis implements two variants of the prediction LSTM network, which are explained in Sections 4.5 and 4.6.

## 4.4 Training Properties

For the training process, the available dataset of approximately 60 minutes of traced TOs is split into three parts. The training set consists of 80 % of the available data samples. Validation and test-set each contain 10 % of the data samples. The test-set is not used for the training process and is only applied for the final evaluation. Iterating over all training examples labels one *epoch*. Every epoch, the validation set is used to calculate a *validation loss*, which is further used to optimize the hyperparameters of the network [29]. In general, tuning the hyperparameters of a neural network using a separate validation set increases its generalization ability.

The implemented LSTM network uses Mini-Batch Gradient Descent as method to process the input data [36]. Mini-Batch Gradient Descent has been proven to provide a robust convergence and a better utilization of computational resources than non batched variants [36]. A minibatch is a subset of the training set, containing $N$ entries that are propagated through the network at once. A single loss is calculated by averaging over the whole batch. In this thesis the Mean Squared Error (MSE) of a single batch $i$ is chosen as loss function.

$$\mathcal{L}(y_{t,j}^{(i)}, \hat{y}_{t,j}^{(i)}) = \frac{1}{N} \sum_{j=1}^{N} \left( y_{t,j}^{(i)} - \hat{y}_{t,j}^{(i)} \right)^2 \tag{4.6}$$

$\hat{y}_{t,j}^{(i)}$ is the $j$th predicted value and $y_{t,j}^{(i)}$ is the $j$th true label of batch $i$ at time step $t$. To ease further reading, batch indices are omitted.
Kingma and Ba [66] introduced the Adam algorithm for optimizing DNNs, which is applied in this thesis. The Adam algorithm optimizes the networks weights by maintaining individual, adaptive learning rates for each separate function parameter.

## 4.5 Approach: Non Autoregression

The Non Autoregression (NA) approach uses a vanilla LSTM network for the implementation of the PM. The network uses a feature vector $\boldsymbol{x}_t = [x_{t-\tau}, ..., x_t]$ as input, which describes a time sequence of GT features. It predicts the parameters of a Gaussian distribution $\hat{\mu}_t, \hat{\sigma}'_t$ for every time step $t$ and samples the final output value $\hat{y}_t$ from this distribution.

### Training

For training, the network uses its input features to estimate the output $\hat{y}_t$. The output is then compared to the true label $y_t$ and the loss value $\mathcal{L}(y_t, \hat{y}_t)$ is calculated based on Equation (4.6). The gradient of the loss function with respect to the network parameters is propagated backwards through time using the BPTT algorithm. This procedure adjusts the network weights in a way that minimizes the overall loss. Figure 4.8 shows the forward path of the training algorithm as unfolded in time.

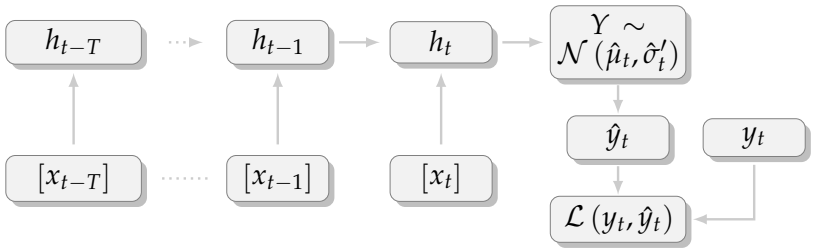



Figure 4.8: Training process of NA approach.

The Gaussian distribution $Y \sim \mathcal{N}(\hat{\mu}_t, \hat{\sigma}'_t)$ is parameterized by the predicted mean $\hat{\mu}_t$ and variance $\hat{\sigma'}_t$, as shown in Equation (4.5).

### Prediction

After training, the network is able to predict MSD based on incoming data. The flow of information during the prediction is similar to the training processes. The LSTM propagates the input sequence $\boldsymbol{x}_t = [x_{t-\tau}, ..., x_t]$ through the network and outputs the sampled value $\hat{y}_t$.

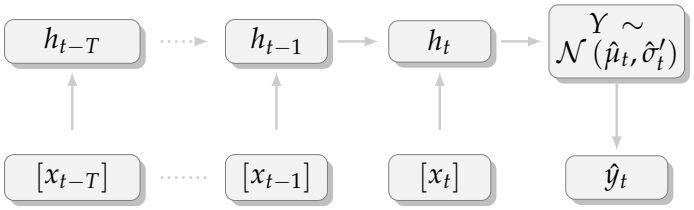



Figure 4.9: Prediction process of NA approach.

Figure 4.9 shows the unfolded network structure during prediction. The network estimates one sample at a time, which is collected to receive a full object trace.

## 4.6 Approach: Single Component Autoregression

In oppose to the approach discussed in Section 4.5, the Single Component Autoregression (SC) approach uses the output of the previous time step $\hat{y}_{t-1}$ as additional input feature for the network.

### Training

During training, the LSTM network uses feature vector $\boldsymbol{x}_t = [x_{t-\tau}, ..., x_t]$, as well as the true label of the previous time step $y_{t-1}$ as input. The network then proceeds similar to the NA variant by estimating the parameters $\hat{\mu}_t, \hat{\sigma}'_t$ and further sampling the predicted value $\hat{y}_t$. Figure 4.10 shows the information flow of the SC approach during training.

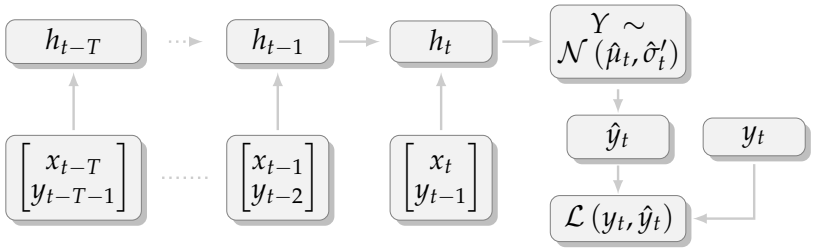



Figure 4.10: Training process of SC approach.

After sampling $\hat{y}_t$, the network calculates the batch MSE based on Equation (4.6) and adjusts its weight accordingly via BPTT.

## Prediction

For the prediction, the network needs a settling time of length $\tau$ before it is able to estimate the first value. This is because the prediction queue, shown in Figure 4.11, must be filled with previous network predictions beforehand. To achieve a realistic behavior during the settling time of the network, the prediction queue is initialized the following way

$$\hat{\mu}_t = \mathrm{GT}_t, \tag{4.7}$$

$$\hat{\sigma}'_t = \eta. \tag{4.8}$$

$\mathrm{GT}_t$ refers to the GT value at time $t$ and $\eta$ is a constant that is defined offline in an empirical way by calculating the global variance of all input traces.

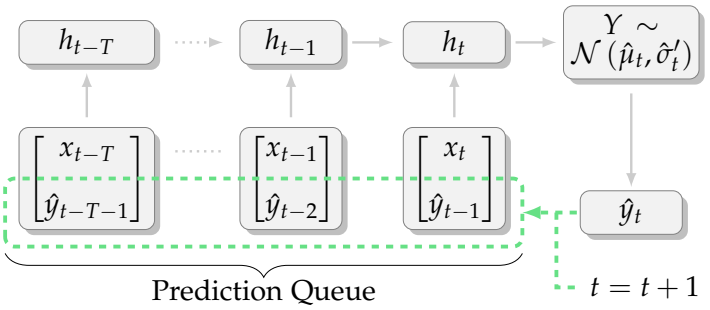



Figure 4.11: Prediction process of SC approach.

After the initialization phase, the network is able to predict the parameters of a normal distribution $\hat{\mu}_t, \hat{\sigma}'_t$, based on previous exposed data. The sampling process of $\hat{y}_t$ is similar to the described procedure in Section 4.5. To use the most recent prediction as input for the next time step, $\hat{y}_t$ is added to the prediction queue.

# 5 Results

This chapter summarizes the assessment of the implemented PM and concludes the final results of this thesis. The purpose of the offline evaluation, is the determination and optimization of the network configuration. The resulting model must be able to reproduce the SD of an AV, with a reasonable accuracy. Further on, the online evaluation integrates the optimized model into a simulation framework, to test the PM in an open-loop.

## 5.1 Offline Evaluation

The offline evaluation is performed separately from the simulation environment and serves the purpose of finding and optimizing the parameters of the LSTM network. It consists of all steps that are pictured on the left side of Figure 4.1 and is performed as an iterative procedure. This section provides the results of this evaluation and the conclusions that are drawn from it.

### Hyperparameter Search

Neural networks try to find the optimal mapping of the provided input data to the corresponding output label by reducing the cost between network prediction and label. The base architecture of the network that is used in this thesis consists of two different approaches, labeled as NA and SC. For each approach, the network itself has to be parameterized by a set of *hyperparameters*, which define the network and training properties. Since finding a suitable set of hyperparameters in an analytic way is a challenging task, the development process usually resorts to the training of multiple models and the comparison of their performance on a validation set. An unsuitable set of parameters can lead to a non-converging loss function or an insufficient accurate prediction.
The following hyperparameters for the LSTM network are empirically optimized in the context of this thesis. The Number of Training Epochs (NE) defines the number of iterations that the full input data-set is processed during training and the Learning Rate (LR) is a measurement for the step-width, the training algorithm takes in the direction of the steepest descent. The Number of Network Layers (NL) defines the number of hidden layers in the LSTM network and the Number of Cells per Layer (NC) defines the number of LSTM cells per layer.

### Training Epochs and Learning Rate

The first part of the hyperparameter search aims to find a suitable combination of the NE and the LR. For this evaluation, all input features, as listed in Table 4.1, are chosen as input for the network, with the exclusion of $v_{to}(SD)$ in case of the NA approach. The training set consists of samples that are preprocessed according to Section 4.2. An applicable combination of LR and NE must lead to a decreasing validation loss during the training process. To confirm this, a test series of models with different hyperparameters is trained on the same dataset and the convergence behavior of the

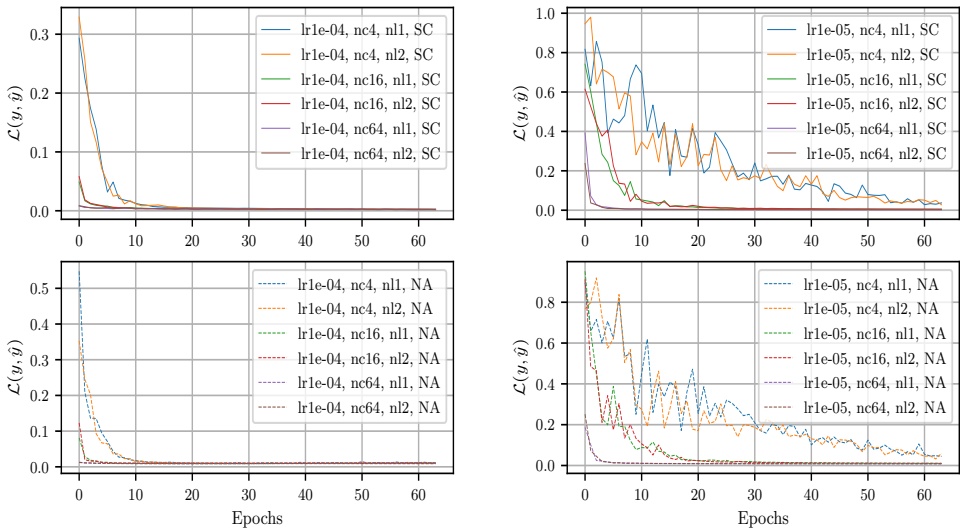
networks validation loss $\mathcal{L}(y, \hat{y})$ is monitored during this process. All combinations of the following hyperparameters are tested for both network approaches, resulting in a total amount of 36 combinations.

$$\text{LR} \in [1e-04, 5e-05, 1e-05],\ \text{NL} \in [1, 2],\ \text{NC} \in [4, 16, 64]$$

Each set of parameters is trained with 5 different models and the resulting validation loss $\mathcal{L}_{1:5}(y, \hat{y})$ is averaged at every time-step. Further on, a *test-set* describes a set of models that have an equal parameterization and are used to verify the consistency of the predictions. Equation (5.1) shows the calculation of the averaged validation loss $\mathcal{L}(y, \hat{y})$ for every test-set.

$$\mathcal{L}(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{1:5}(y, \hat{y}) \tag{5.1}$$

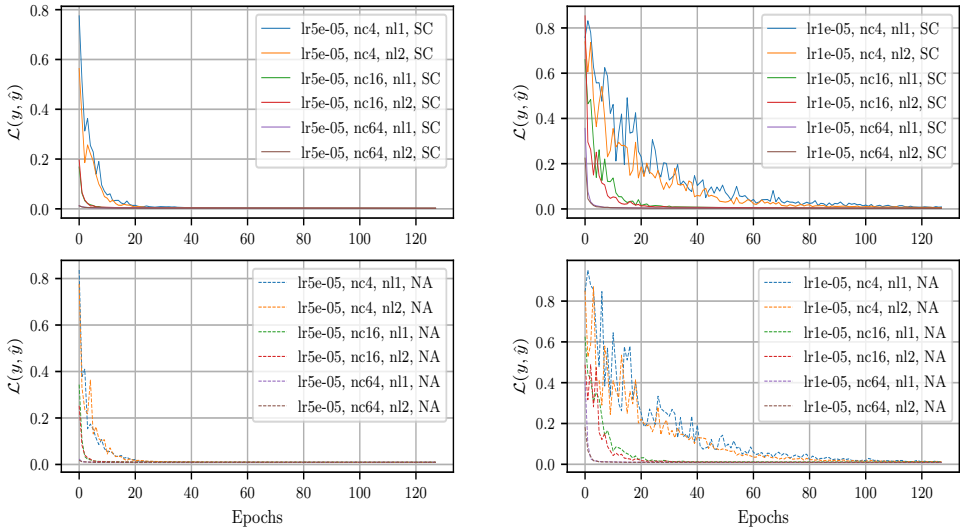
The upper plots in Figure 5.1 show the convergence of $\mathcal{L}(y, \hat{y})$ for the SC approach, the lower plots for the NA approach. The loss function of every test-set in Figure 5.1a converges at latest after 20 epochs. This indicates that with the chose combination of LR and NE, the network is able to find a solution for the given task. On the contrary, the number of epochs is not sufficient for the training of the test-sets showed in Figure 5.1b, especially for configurations that use a low NC. This is shown as the loss function $\mathcal{L}(y, \hat{y})$ does not converge after the specified amount of epochs for these test-sets. In addition, the convergence behavior contains more fluctuations and starts at a higher offset than the functions with LR=1e − 04. For this reason, the evaluation of test-sets using smaller LRs is repeated with 128 training epochs.




(a) Fixed parameters LR=1e − 04, NE=64.

(b) Fixed parameters LR=1e − 05, NE=64.

Figure 5.1: Convergence of validation loss $\mathcal{L}(y, \hat{y})$ after 64 training epochs.




(a) Fixed parameters LR=5e − 05, NE=128.

(b) Fixed parameters LR=1e − 05, NE=128.

Figure 5.2: Convergence of validation loss $\mathcal{L}(y, \hat{y})$ after 128 training epochs.

The training with an increased epoch count of 128 is shown in Figure 5.2. Again, the upper plots show the convergence of $\mathcal{L}(y, \hat{y})$ for the SC approach and the lower plots for the NA approach. The loss function of all parameter configurations converges after the specified amount of training epochs, which confirms the update of the NE for these LR configurations.

To further evaluate the convergence behavior of the training process, the variance of $\mathcal{L}(y, \hat{y})$ for each test-set is determined after the last training epoch, as shown in Equation (5.2). This parameter is expected to provide information about the consistency of the training procedure, as small values indicate a similar performance.

$$\sigma^2\left(\mathcal{L}(y, \hat{y})\right) = \frac{1}{N} \sum_{i=1}^{N} \left(\mathcal{L}_i(y, \hat{y}) - \mathcal{L}(y, \hat{y})\right)^2 \tag{5.2}$$

Table 5.1 compares the values of $\mathcal{L}(y, \hat{y})$ and $\sigma^2\left(\mathcal{L}(y, \hat{y})\right)$ for all evaluated test-sets after training. The top 6 rows list the results of the NA approach, the bottom 6 rows the results of the SC approach. For this comparison, the models using LR=1e − 04 are trained 64 epochs, the models using LR=5e − 05 and LR=1e − 05 are trained for 128 epochs.

All configurations lead to values of $\mathcal{L}(y, \hat{y})$ with comparable magnitude and contain small values for the variance. The highest average validation loss $\mathcal{L}(y, \hat{y}) = 0.0126$ is about 7.5 times higher than the lowest achieved loss value of $\mathcal{L}(y, \hat{y}) = 0.0017$. Those values represent the MSE between normalized quantities, which means they do not represent the actual physical difference but a scaled version. For that reason, the results shown in Table 5.1 are only used to evaluate relative trends that occur during training. In case of the SC approach, the prediction quality of all test-sets with LR=1e − 04 is positively influenced by an increase NL and NC. In addition, the variance of the

| $\mathcal{L}(y,\hat{y})$ $(\sigma^2(\mathcal{L}(y,\hat{y})))$ | | | |
|---|---|---|---|
| Configuration | LR=1e − 04 | LR=5e − 05 | LR=1e − 05 |
| NA, NC=4, NL=1 | 0.0110 (1.0e − 06) | 0.0122 (4.3e − 08) | 0.0097 (4.2e − 06) |
| NA, NC=4, NL=2 | 0.0098 (2.0e − 07) | 0.0126 (8.5e − 08) | 0.0093 (1.4e − 06) |
| NA, NC=16, NL=1 | 0.0108 (6.1e − 06) | 0.0109 (1.0e − 07) | 0.0090 (3.2e − 06) |
| NA, NC=16, NL=2 | 0.0088 (1.5e − 08) | 0.0095 (1.3e − 07) | 0.0089 (1.8e − 07) |
| NA, NC=64, NL=1 | 0.0106 (2.2e − 06) | 0.0087 (4.2e − 07) | 0.0091 (8.4e − 08) |
| NA, NC=64, NL=2 | 0.0107 (7.1e − 07) | 0.0088 (1.7e − 07) | 0.0091 (1.6e − 07) |
| SC, NC=4, NL=1 | 0.0026 (5.5e − 07) | 0.0088 (5.5e − 08) | 0.0021 (1.3e − 05) |
| SC, NC=4, NL=2 | 0.0022 (2.5e − 07) | 0.0059 (5.9e − 08) | 0.0020 (1.2e − 06) |
| SC, NC=16, NL=1 | 0.0020 (3.5e − 07) | 0.0031 (1.0e − 07) | 0.0019 (6.0e − 07) |
| SC, NC=16, NL=2 | 0.0024 (1.9e − 07) | 0.0031 (7.2e − 08) | 0.0020 (2.7e − 07) |
| SC, NC=64, NL=1 | 0.0020 (1.2e − 07) | 0.0023 (7.6e − 09) | 0.0017 (3.0e − 08) |
| SC, NC=64, NL=2 | 0.0019 (4.5e − 08) | 0.0023 (2.8e − 08) | 0.0017 (8.9e − 08) |

Table 5.1: Comparison of validation loss after training. Every cell contains the mean loss and the variance of the test-set in brackets.

loss function decreases when using larger architectures. The NA approach shows no recognizable trend relative to the NL and NC. The increasing performance that comes with an increasing NL and NC continues for both approaches in case of LR=5e − 05, as again the validation loss decreases with an increasing NL and NC. Both approaches show the most promising results with LR=1e − 05, which is highlighted in green colour. Decreasing the LR leads to an improved prediction ability of the model for almost every case of this validation. The evaluation of the first test series shows the influence that the hyperparameters have on the performance of the network. The effect of the LR on the training process and further the approximation ability of the networks is pointed out in this section and reasons for the preference of the combination of LR=1e − 05 and NE=128 are provided. To further investigate the influence of the model structure on the prediction performance, the trained models are benchmarked on a selected, separate set of Test Tracks (TTs).

## Hyperparameter Evaluation

To find the most suitable set of hyperparameters, the previously trained test-sets are applied on the prediction task of a separate number of TTs. This independent set of TTs is chosen to estimate the generalization ability of the trained models and consists of 5 exemplary object traces $TT_{1:5}$, as summarized in Table 5.2. In oppose to the training and validation set, the input slices for the network are not shuffled, to kept their temporal order for this evaluation.

To verify the prediction performance of the networks, the MSE between the rescaled MSD and SD is used as baseline. The MSE measures the offset error $\epsilon = SD - MSD$ and is as single performance indicator not sufficient for the evaluation of the prediction quality, since it provides little information about signal characteristics. Therefore, the smoothness of the error signal $\epsilon$ is also taken into account by calculating the variance $\sigma^2(\epsilon)$ and using it as second performance indicator. This procedure is expected to lead to the best suited configuration of the hyperparameters for the final evaluation. Tables 5.3-5.8 show the MSE between SD and MSD per test-set, for every TT. Every table shows the evaluation of a fixed LR. Colored cells highlight the best performing set of hyperparameters per approach and TT, solely based on the MSE.

| Test-track | t [s] | $\bar{v}_{x,to}$ [m/s] | Summary |
|---|---|---|---|
| $TT_1$ | 35 | 35 | Slow acceleration after breaking |
| $TT_2$ | 120 | 33 | Hard breaking and acceleration |
| $TT_3$ | 42 | 28 | Breaking with spike in sensor values |
| $TT_4$ | 32 | 28 | Long breaking maneuver |
| $TT_5$ | 26 | 36 | Alternating breaking and acceleration |

Table 5.2: Description and basic properties of $TT_{1:5}$. The velocity $\bar{v}_{x,to}$ is averaged over the whole track and rounded towards the nearest integer.

Figure 5.3 visualizes the results from Tables 5.3-5.8. The left column represents the NA approach, the right column the SC approach. Every row represents one of $TT_{1:5}$ in increasing order. In this comparison, the NA approach shows a rather inconsistent behavior regarding the correlation of model complexity and performance. Tracks $TT_3$ and $TT_5$ benefit from larger models, since their level of performance rises with increasing model complexity. Whereas larger model architectures affect the performance on tracks $TT_1$ and $TT_2$ in a negative way for all learning rates. In addition, there is no clear preference for the value of the LR. On the contrary, the SC approach tends to perform better using a combination of low model complexity and small LR. The orange curve, representing $LR = 1e-05$, results in the overall best performance.

To further analyze the prediction performance of the test-sets, all TTs are plotted and the parameter $\sigma^2(\epsilon)$ is evaluated. Since this evaluation leads to a large amount of plots (i.e. 2 approaches x 18 hyperparameter configurations x 5 TTs), only one representative example is presented in this section to demonstrate occurring trends. Figures 5.4-5.6 show the predictions for $TT_2$, made by test-sets with respective $LR = 1e-04$ and $LR = 1e-05$. Every subplot shows the evaluation of the labeled test-set using the SC approach at the top and the NA approach at the bottom. A large value of the MSE between $v_{x,to}(SD)$ and $v_{x,to}(MSD)$ mostly corresponds to a large offset error, as the network is not able to approximate the correct velocity value. The gray outlines visualize the $\pm 2\sigma$ distance for each test-set. A large value of $\sigma^2(\epsilon)$ corresponds to a large difference in the noise behavior between $v_{x,to}(SD)$ and $v_{x,to}(MSD)$. This effect is particularly visible in Figures 5.4c-5.4d.
In general, the NA approach is significantly more consistent in its result than the SC approach, as the average values for $\pm 2\sigma$ is lower for almost every test-set. High model complexities tend to result in higher loss values for both approaches. In case of the SC approach, the model predictions also become increasingly inconsistent with larger model architectures, which is represented by a large $\pm 2\sigma$ distance. In that case, the model is not capable of finding a suitable minimum um $\mathcal{L}(y, \hat{y})$, as shown in Figures 5.5a-5.6b and 5.6c-5.6d. The combination of a small network architecture and a large value for the LR leads to a good approximation of the noise behavior. In contrast, combining small network architectures with small LRs increases the uncertainty. Evaluating all test-tracks concludes that the best results are achieved with following combination of hyperparameters

$$\text{LR} = 1e-05,\ \text{NL} = 1,\ \text{NC} = 16.$$

This set of hyperparameters has the overall lowest values for the MSE, while keeping $\sigma^2(\epsilon)$ at a reasonable low level for both approaches. In addition, the consistency of the prediction, shown as small $\pm 2\sigma$ distance, has a promising low level. All further networks are therefore trained with this combination of hyperparameters.

| $MSE\{v_{x,to}(SD), v_{x,to}(MSD)\}$ | | | | | | |
|---|---|---|---|---|---|---|
| TT | NL=1, NC=4 | NL=2, NC=4 | NL=1, NC=16 | NL=2, NC=16 | NL=1, NC=64 | NL=2, NC=64 |
| $TT_1$ | 0.4614 | 0.4834 | 0.4885 | 0.4987 | 0.5594 | 0.7696 |
| $TT_2$ | 0.1251 | 0.1425 | 0.1247 | 0.1604 | 0.1853 | 0.2046 |
| $TT_3$ | 0.0923 | 0.0790 | 0.0711 | 0.0757 | 0.1086 | 0.1253 |
| $TT_4$ | 1.2153 | 1.7957 | 1.4143 | 1.1391 | 1.1672 | 1.1104 |
| $TT_5$ | 0.1900 | 0.1737 | 0.1185 | 0.1198 | 0.1473 | 0.1566 |

Table 5.3: Evaluation of grid search for NA approach and LR=1e − 04.

| $MSE\{v_{x,to}(SD), v_{x,to}(MSD)\}$ | | | | | | |
|---|---|---|---|---|---|---|
| TT | NL=1, NC=4 | NL=2, NC=4 | NL=1, NC=16 | NL=2, NC=16 | NL=1, NC=64 | NL=2, NC=64 |
| $TT_1$ | 0.4085 | 0.6299 | 0.6166 | 0.6510 | 0.4203 | 0.5158 |
| $TT_2$ | 0.1128 | 0.1991 | 0.1965 | 0.1431 | 0.1038 | 0.1592 |
| $TT_3$ | 0.0988 | 0.0853 | 0.0971 | 0.0708 | 0.0764 | 0.0778 |
| $TT_4$ | 1.2527 | 1.5433 | 1.5144 | 1.3155 | 2.0977 | 2.0237 |
| $TT_5$ | 0.1506 | 0.1209 | 0.1003 | 0.0936 | 0.1204 | 0.1085 |

Table 5.4: Evaluation of grid search for NA approach and LR=5e − 05.

| $MSE\{v_{x,to}(SD), v_{x,to}(MSD)\}$ | | | | | | |
|---|---|---|---|---|---|---|
| TT | NL=1, NC=4 | NL=2, NC=4 | NL=1, NC=16 | NL=2, NC=16 | NL=1, NC=64 | NL=2, NC=64 |
| $TT_1$ | 0.4401 | 0.5742 | 0.5174 | 0.5685 | 0.5857 | 0.6031 |
| $TT_2$ | 0.1476 | 0.1109 | 0.1541 | 0.1858 | 0.2324 | 0.1723 |
| $TT_3$ | 0.1885 | 0.1165 | 0.0846 | 0.0770 | 0.0933 | 0.0759 |
| $TT_4$ | 1.3680 | 1.2953 | 2.0615 | 1.5570 | 1.5585 | 1.4041 |
| $TT_5$ | 0.4025 | 0.3660 | 0.1446 | 0.1658 | 0.0729 | 0.0812 |

Table 5.5: Evaluation of grid search for NA approach and LR=1e − 05.

| $MSE\{v_{x,to}(SD), v_{x,to}(MSD)\}$ | | | | | | |
|---|---|---|---|---|---|---|
| TT | NL=1, NC=4 | NL=2, NC=4 | NL=1, NC=16 | NL=2, NC=16 | NL=1, NC=64 | NL=2, NC=64 |
| $TT_1$ | 0.5681 | 0.4664 | 1.0714 | 0.5532 | 0.4147 | 0.5248 |
| $TT_2$ | 0.2364 | 0.0912 | 0.1511 | 0.2097 | 0.9385 | 0.3936 |
| $TT_3$ | 0.0834 | 0.0905 | 0.1447 | 0.1170 | 0.4233 | 0.0974 |
| $TT_4$ | 1.2354 | 2.4748 | 1.9201 | 1.8050 | 2.0759 | 2.4100 |
| $TT_5$ | 0.7114 | 0.2718 | 0.1792 | 0.3501 | 0.3773 | 0.2216 |

Table 5.6: Evaluation of grid search for SC approach and LR=1e − 04.

| $MSE\{v_{x,to}(SD), v_{x,to}(MSD)\}$ | | | | | | |
|---|---|---|---|---|---|---|
| TT | NL=1, NC=4 | NL=2, NC=4 | NL=1, NC=16 | NL=2, NC=16 | NL=1, NC=64 | NL=2, NC=64 |
| $TT_1$ | 0.6122 | 0.7996 | 0.6199 | 0.5212 | 0.5722 | 0.4857 |
| $TT_2$ | 0.1345 | 0.1918 | 0.2678 | 0.1878 | 0.1232 | 0.2122 |
| $TT_3$ | 0.1983 | 0.1532 | 0.0947 | 0.0950 | 0.2150 | 0.0501 |
| $TT_4$ | 2.4048 | 3.3452 | 2.9948 | 3.2621 | 2.0977 | 2.0237 |
| $TT_5$ | 0.5748 | 0.3381 | 0.3686 | 0.3277 | 0.2376 | 0.2604 |

Table 5.7: Evaluation of grid search for SC approach and LR=5e − 05.

| $MSE\{v_{x,to}(SD), v_{x,to}(MSD)\}$ | | | | | | |
|---|---|---|---|---|---|---|
| TT | NL=1, NC=4 | NL=2, NC=4 | NL=1, NC=16 | NL=2, NC=16 | NL=1, NC=64 | NL=2, NC=64 |
| $TT_1$ | 0.3189 | 0.4376 | 0.5299 | 0.4927 | 0.5579 | 0.5467 |
| $TT_2$ | 0.0778 | 0.1905 | 0.1190 | 0.1282 | 0.3910 | 0.2292 |
| $TT_3$ | 0.1172 | 0.1344 | 0.0639 | 0.0639 | 0.0838 | 0.0571 |
| $TT_4$ | 1.1213 | 1.2273 | 1.4854 | 1.9845 | 3.5642 | 3.9202 |
| $TT_5$ | 0.2383 | 0.2518 | 0.2074 | 0.2059 | 0.3949 | 0.3418 |

Table 5.8: Evaluation of grid search for SC approach and LR=1e − 05.

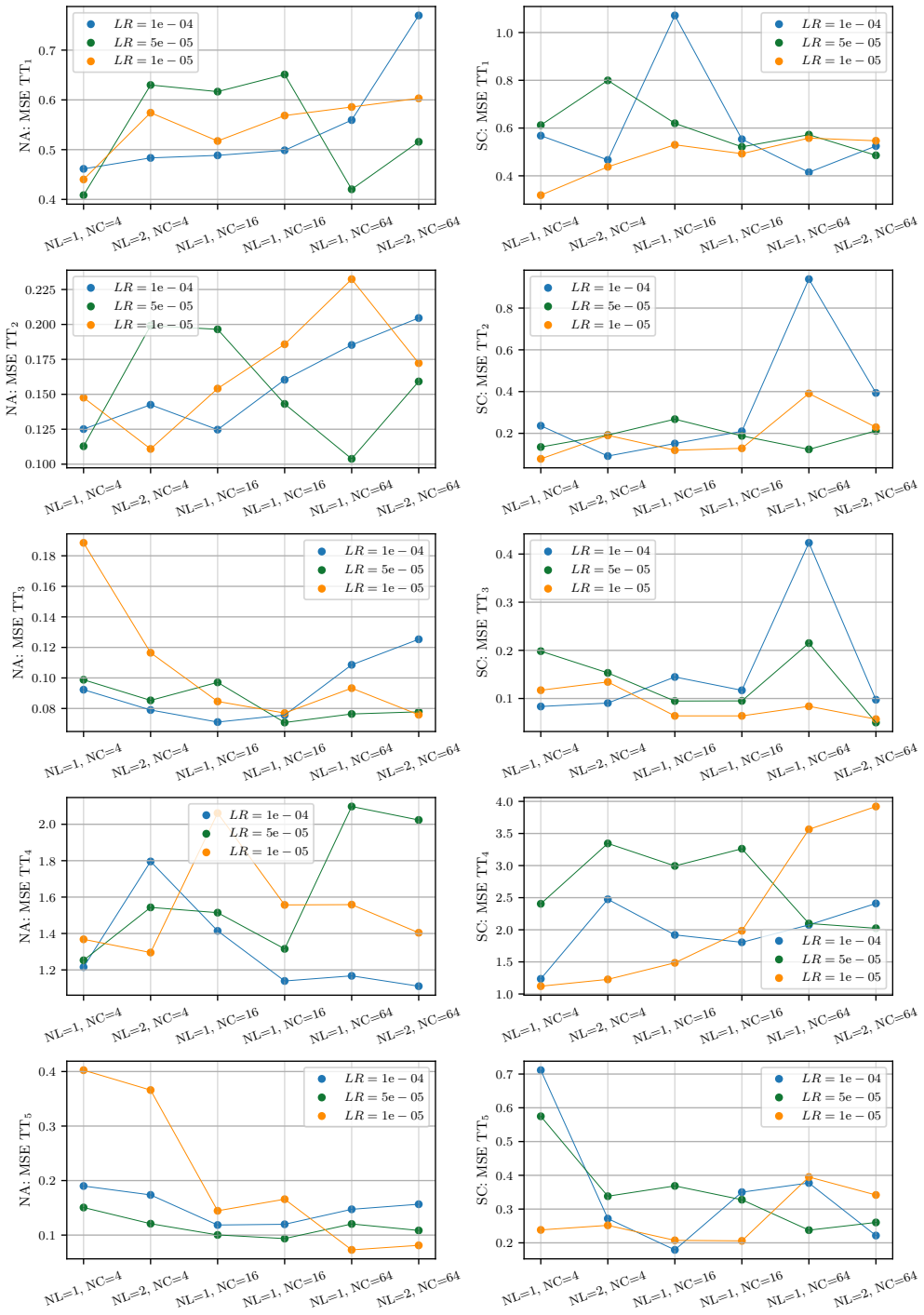
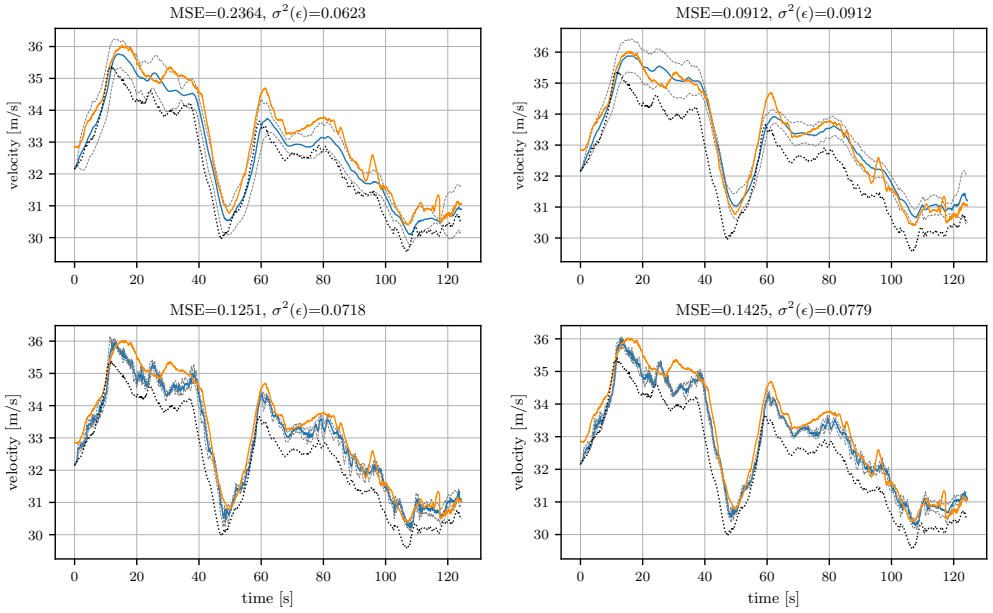
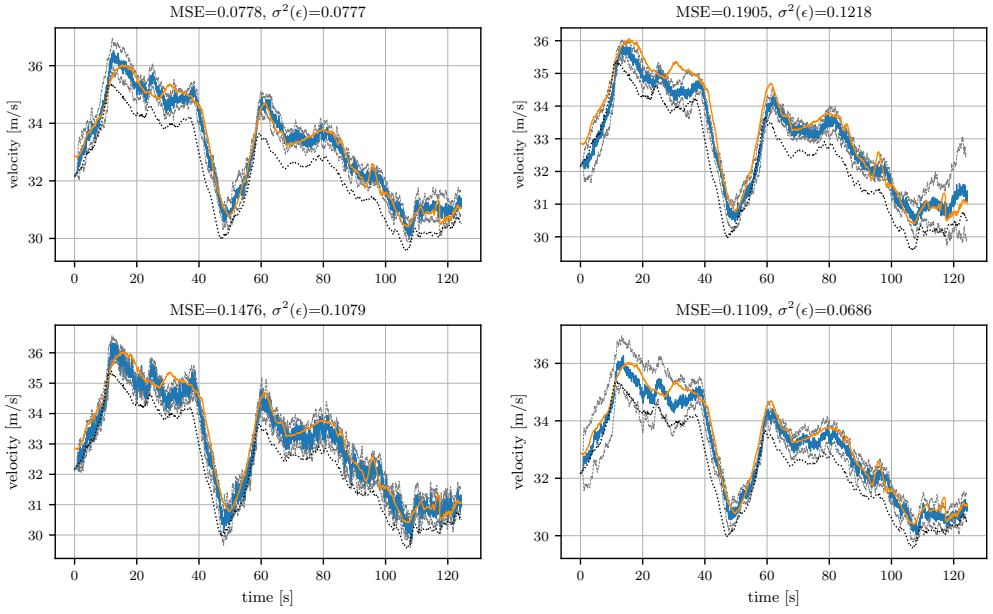



Figure 5.3: Comparison of $MSE\{v_{x,to}(SD), v_{x,to}(MSD)\}$ based on the feature selection. The left side shows the NA approach, the right side the SC approach. Every row represents one of $TT_{1:5}$.




(a) Fixed parameters LR=1e − 04, NL=1, NC=4.

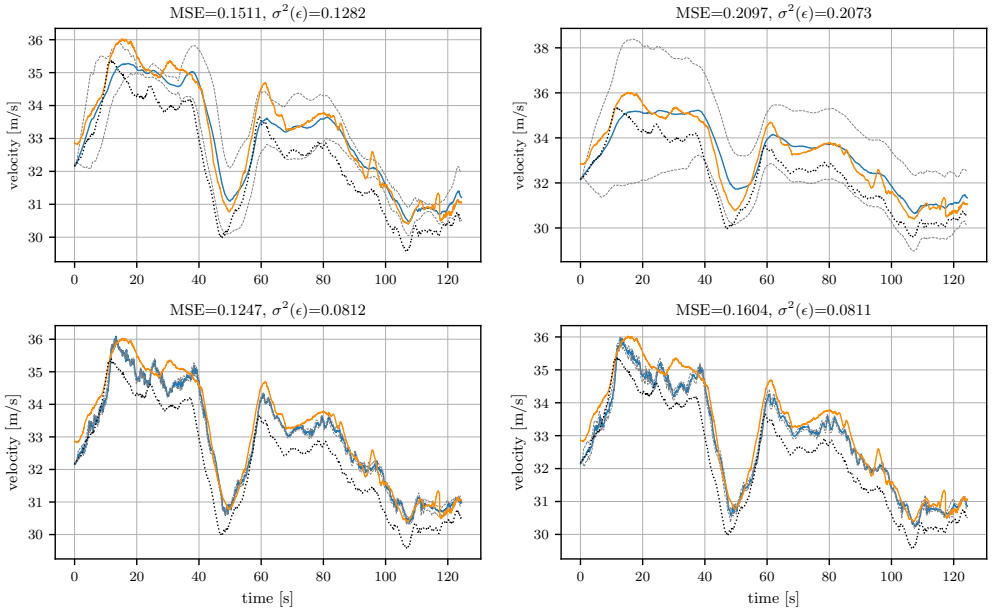
(b) Fixed parameters LR=1e − 04, NL=2, NC=4.




(c) Fixed parameters LR=1e − 05, NL=1, NC=4.

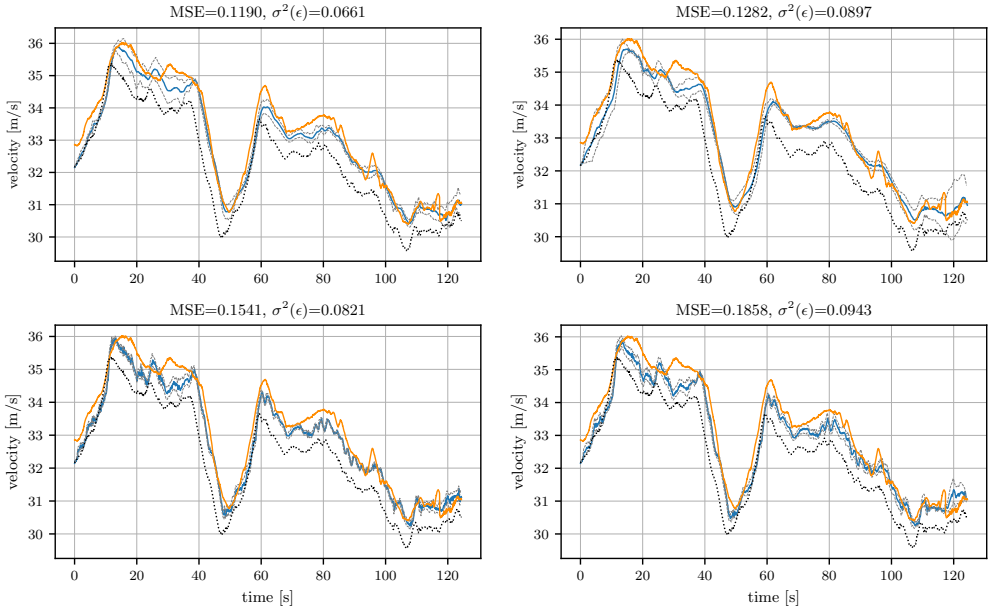
(d) Fixed parameters LR=1e − 05, NL=2, NC=4.

Figure 5.4: Comparison of $TT_2$ using LR $\in [1e-04, 1e-05]$, NL $\in [1, 2]$ and NC $= 4$ for the training. The dotted curve labels $v_{x,to}(GT)$, the orange curve $v_{x,to}(SD)$ and the blue curve $v_{x,to}(MSD)$. The gray, dashed outlines show the $\pm 2\sigma$ distance for each test-set.




(a) Fixed parameters LR=1e − 04, NL=1, NC=16.

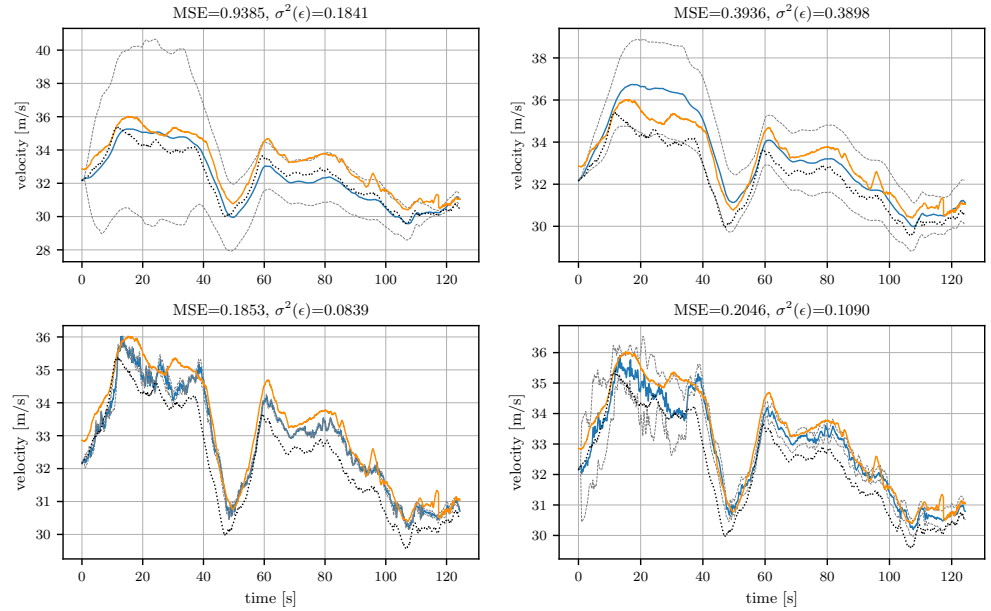
(b) Fixed parameters LR=1e − 04, NL=2, NC=16.




(c) Fixed parameters LR=1e − 05, NL=1, NC=16.

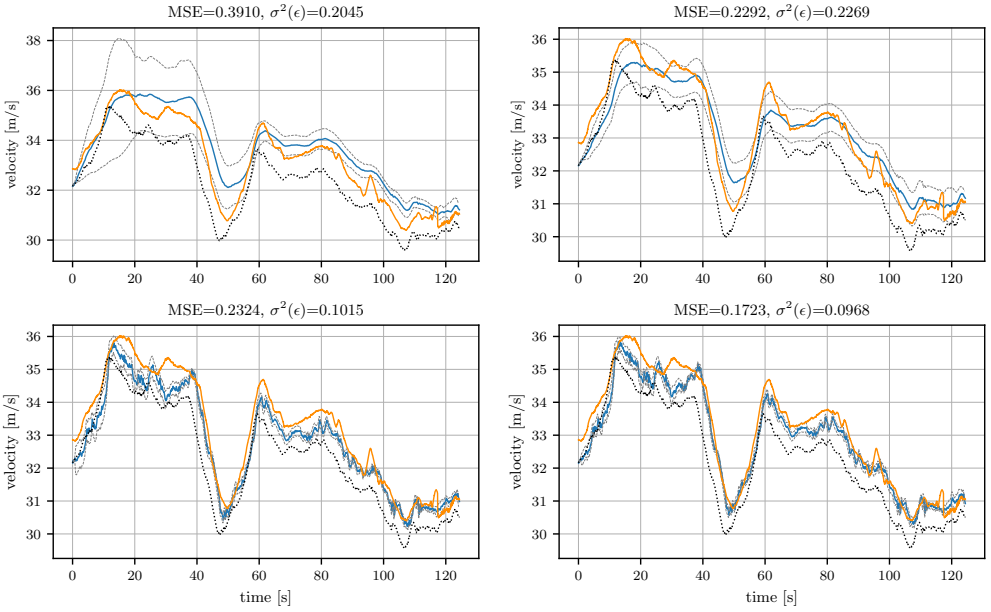
(d) Fixed parameters LR=1e − 05, NL=2, NC=16.

Figure 5.5: Comparison of $TT_2$ using LR $\in [1e-04, 1e-05]$, NL $\in [1, 2]$ and NC $= 16$ for the training. The dotted curve labels $v_{x,to}(GT)$, the orange curve $v_{x,to}(SD)$ and the blue curve $v_{x,to}(MSD)$. The gray, dashed outlines show the $\pm 2\sigma$ distance for each test-set.




(a) Fixed parameters LR=1e − 04, NL=1, NC=64.

(b) Fixed parameters LR=1e − 04, NL=2, NC=64.




(c) Fixed parameters LR=1e − 05, NL=1, NC=64.

(d) Fixed parameters LR=1e − 05, NL=2, NC=64.

Figure 5.6: Comparison of $TT_2$ using LR $\in [1e-04, 1e-05]$, NL $\in [1, 2]$ and NC $= 64$ for the training. The dotted curve labels $v_{x,to}(GT)$, the orange curve $v_{x,to}(SD)$ and the blue curve $v_{x,to}(MSD)$. The gray, dashed outlines show the $\pm 2\sigma$ distance for each test-set.

## Choice of Feature-Set

The final part of the offline evaluation investigates the influences, that the choice of input features has on the models performance. The training of all previous models uses the whole available data-set and every input feature listed in Table 4.1. The usage of a subset of input features can lead to an increased performance, since potentially wrong conclusions, that are drawn from inconsistencies in redundant information, can be avoided. An examples for this can be inconsistencies between the change of $v_{x,to}(GT)$ and the timely corresponding values of $a_{x,to}(GT)$. In addition, removing irrelevant parts from the full input set can be beneficial for the predictive ability of the models. As previously cited, the error of the perceived SD depends amongst other things on the performed maneuver and has a higher magnitude during track segments with increased vehicle dynamics. For this reason, the input set is pruned into four dynamic levels, based on the acceleration of the perceived TO. A set of three metrics is established to quantify this pruning. On the example of $02-17-25$, an input track must have a variance $\sigma^2(a_{x,to}(GT)) \geq 0.2$ (first value) and an absolute value of $|a_{x,to}(GT)| \geq 1.7\text{m/s}^2$ for at least 2.5s (second and third value). The four levels of dynamics and the resulting number of input tracks after pruning are listed in Table 5.9.

| | full data | $02-17-25$ | $03-21-31$ | $04-25-36$ |
|---|---|---|---|---|
| Nr. of Traces | 164 | 24 | 18 | 11 |

Table 5.9: Pruning of input data based in dynamic properties.

Table 5.10 shows the combination of input features, that are used with the pruned data-sets for the final evaluation of this chapter.

| Abbreviation | Base Features | Additional Features |
|---|---|---|
| $\mathcal{F}_{all}$ | $v_{x,to}(GT), d_{x,rel}(GT)$ | $a_{x,to}(GT), v_{ego}(GT), p_{ego}(GT), h_{rel}(GT)$ |
| $\mathcal{F}_{vra}$ | $v_{x,to}(GT), d_{x,rel}(GT)$ | $a_{x,to}(GT)$ |
| $\mathcal{F}_{vrv}$ | $v_{x,to}(GT), d_{x,rel}(GT)$ | $v_{ego}(GT)$ |
| $\mathcal{F}_{vrp}$ | $v_{x,to}(GT), d_{x,rel}(GT)$ | $p_{ego}(GT$ |
| $\mathcal{F}_{vrh}$ | $v_{x,to}(GT), d_{x,rel}(GT)$ | $h_{rel}(GT)$ |

Table 5.10: Used input features and their abbreviations.

To summarize, the final evaluation of this section consists of [2 approaches x 4 dynamic levels x 5 feature selections], which leads to a total number of 40 combinations of training parameters. Every combination is trained with 10 models, resulting in 400 trained models for this test.

Tables 5.12-5.15 show the best performing feature-sets for every approach, evaluated on all four dynamic levels of the data-set. For enhanced readability, the values inside the curly brackets are omitted for the labeling of $MSE_{all}$ and $MSE_{test}$. Every row lists the averaged MSE between $v_{x,to}(SD)$ and $v_{x,to}(MSD)$, based on the selected input set and each cell shows the result of a single model $m$. The first value in every cell represents $MSE_{all}$, which is calculated between all available tracks. This value represents the ability of the models to correctly approximate the provided input data. The second value in brackets shows the $MSE_{test}$, which is calculated and averaged over the five test tracks only. The green colored cells represent the overall best result for each table and green colored text highlights the best candidate of each row, based on $MSE_{test}$.
For every approach, the feature-set containing the model with the lowest value for $MSE_{test}$ is selected and compared to the same approach using the $\mathcal{F}_{all}$ feature-set.

The overall best performing feature-set for the NA approach is the $\mathcal{F}_{vrp}$ set. The best performing set for the SC approach is the $\mathcal{F}_{vrh}$ set. This comparison provides insight on the influence that single features have on the prediction performance of the trained models. The results from Tables 5.12-5.15 are additionally visualized in Figures 5.7-5.8 as boxplots. Here, the orange line marks the median and the lower and upper boundaries mark 25% and 75% of the test-data, respectively. The whiskers have a length of 1.5 times the interquartile range. All boxplots show the best three candidates for every dynamic level of data and the highlighted feature-sets. The result is then compared to the error between SD and GT.
The columns of Figures 5.7-5.8 show the effect that the dynamic based pruning of the data-set has on the prediction performance. Test-sets using the $\mathcal{F}_{all}$ feature-set show a clear trend towards a decreased accuracy for both approaches on the pruned data-set. Reducing the amount of training data decreases the performance of the models drastically and leads to an increased MSE. This effect is also visible in case of the $\mathcal{F}_{vrp}$ and $\mathcal{F}_{vrh}$ feature-set, but in a reduced form. Compared to test-sets using the $\mathcal{F}_{all}$ feature-set, the usage of a smaller amount of features stabilizes the performance of the models when applied on the pruned training data. Although both approaches show the best performance when using the whole data-set for training, using a small number of input features can still lead to results with reasonable accuracy, in case of a reduced data-set.

Overall, the NA approach leads to a more consist performance and models using the SC architecture can lead to better results, but show less consistency. The SC approach, trained on the full data-set consisting of the $\mathcal{F}_{vrh}$ feature-set, stands out as the test-set with the highest predictive accuracy and its best candidate is chosen as model for the online evaluation. Figure 5.9 shows the final result of the offline evaluation on the example of $TT_2$, using the best performing models $m_{na,vrp,3}$ for the NA approach and $m_{sc,vrh,5}$ for the SC approach. Hereby, $m_{na,vrp,3}$ is the third model that is implemented using the NA approach and trained on the $\mathcal{F}_vrp$ features-set.
Because the model shown in Figure 5.9a has no information about its previous predictions, it mainly tries the minimize the offset between $v_{x,to}(GT)$ and $v_{x,to}(SD)$, in order to find a good approximation. This approach does not capture the full characteristics of a the SD. In oppose to that, the SC approach reproduces the timely progression of the SD very accurately. This is especially visible during the breaking and re-accelerating maneuver at time $40-60$s. Both approaches lack the ability of predicting the correct SD of $TT_2$, in sections where the GT and SD signals progress in different directions. An example can be seen during time $20-40$s. Table 5.11 compares $MSE_{test}\{v_{x,to}(SD), v_{x,to}(GT)\}$ and $MSE_{test}\{v_{x,to}(SD), v_{x,to}(MSD)\}$ of the best candidate $m_{sc,vrh,5}$. The implemented PM shows a superior performance on every TT, compared to the error between SD and GT.

| | $TT_1$ | $TT_2$ | $TT_3$ | $TT_4$ | $TT_5$ |
|---|---|---|---|---|---|
| $MSE_{test}\{v_{x,to}(SD), v_{x,to}(GT)\}$ | 1.0222 | 0.8717 | 0.3860 | 1.4779 | 0.3613 |
| $MSE_{test}\{v_{x,to}(SD), v_{x,to}(MSD)\}$ | 0.3711 | 0.0721 | 0.0890 | 0.8168 | 0.3467 |

Table 5.11: Comparison of $MSE_{test}$ for every TT. The MSD consists of the predicted values of the best candidate $m_{sc,vrh,5}$.

This result answers the first research question of this thesis and confirms that the implemented PM is capable of offline reproducing the sensor-error $\Delta S$, so that

$$MSE_{test}\{v_{x,to}(SD), v_{x,to}(MSD)\} < MSE_{test}\{v_{x,to}(SD), v_{x,to}(GT)\} \; \forall \; TT_{1:5} \tag{5.3}$$

| $MSE_{all}\{v_{x,to}(SD), v_{x,to}(MSD)\}$ ($MSE_{test}\{v_{x,to}(SD), v_{x,to}(MSD)\}$) | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| data set | $m_{na,all,0}$ | $m_{na,all,1}$ | $m_{na,all,2}$ | $m_{na,all,3}$ | $m_{na,all,4}$ | $m_{na,all,5}$ | $m_{na,all,6}$ | $m_{na,all,7}$ | $m_{na,all,8}$ | $m_{na,all,9}$ |
| all, full data | 0.32 (0.35) | 0.29 (0.31) | 0.31 (0.40) | 0.32 (0.44) | 0.31 (0.32) | 0.31 (0.37) | 0.32 (0.44) | 0.30 (0.39) | 0.31 (0.30) | 0.31 (0.37) |
| all, 02-17-25 | 2.38 (1.14) | 0.99 (0.67) | 1.87 (1.38) | 1.41 (0.74) | 1.16 (0.85) | 1.46 (0.78) | 1.32 (0.95) | 1.68 (0.83) | 1.11 (0.90) | 1.74 (1.02) |
| all, 03-21-31 | 5.46 (2.32) | 8.43 (3.42) | 10.62 (3.13) | 3.69 (2.69) | 3.45 (2.06) | 175.81 (41.27) | 11.12 (2.88) | 4.49 (1.13) | 20.58 (5.59) | 6.73 (3.04) |
| all, 04-25-36 | 193.56 (73.23) | 240.02 (74.25) | 237.64 (66.18) | 333.36 (108.45) | 153.61 (39.41) | 121.86 (31.25) | 106.08 (41.18) | 263.82 (110.71) | 128.81 (52.78) | 317.57 (87.79) |

Table 5.12: NA approach: Final evaluation of models $m_{na,all,1:10}$, using the $\mathcal{F}_{all}$ feature-set.

| $MSE_{all}\{v_{x,to}(SD), v_{x,to}(MSD)\}$ ($MSE_{test}\{v_{x,to}(SD), v_{x,to}(MSD)\}$) | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| data set | $m_{na,vrp,0}$ | $m_{na,vrp,1}$ | $m_{na,vrp,2}$ | $m_{na,vrp,3}$ | $m_{na,vrp,4}$ | $m_{na,vrp,5}$ | $m_{na,vrp,6}$ | $m_{na,vrp,7}$ | $m_{na,vrp,8}$ | $m_{na,vrp,9}$ |
| vrp, full data | 0.40 (0.27) | 0.38 (0.26) | 0.36 (0.27) | 0.39 (0.25) | 0.37 (0.26) | 0.38 (0.26) | 0.41 (0.28) | 0.39 (0.27) | 0.38 (0.27) | 0.39 (0.27) |
| vrp, 02-17-25 | 0.98 (0.94) | 0.83 (0.89) | 0.67 (0.52) | 0.61 (0.45) | 0.99 (0.88) | 0.88 (0.68) | 0.80 (0.65) | 0.65 (0.69) | 0.66 (0.55) | 0.69 (0.48) |
| vrp, 03-21-31 | 0.88 (0.39) | 0.88 (0.56) | 1.33 (0.72) | 0.92 (0.83) | 0.89 (0.53) | 0.61 (0.31) | 0.85 (0.57) | 1.46 (0.96) | 1.64 (1.11) | 1.50 (1.07) |
| vrp, 04-25-36 | 9.98 (11.00) | 12.46 (13.84) | 7.27 (5.91) | 11.52 (11.90) | 4.43 (4.09) | 14.20 (19.09) | 6.71 (3.39) | 8.73 (8.49) | 12.83 (12.56) | 5.32 (5.09) |

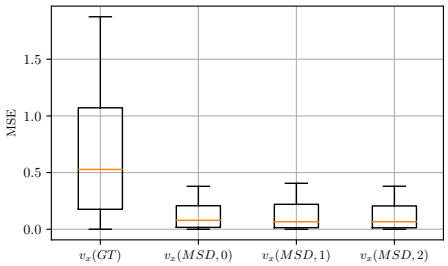
Table 5.13: NA approach: Final evaluation of models $m_{na,vrp,1:10}$, using only $\mathcal{F}_{vrp}$ feature-set.

| $MSE_{all}\{v_{x,to}(SD), v_{x,to}(MSD)\}$ ($MSE_{test}\{v_{x,to}(SD), v_{x,to}(MSD)\}$) | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| data set | $m_{sc,all,0}$ | $m_{sc,all,1}$ | $m_{sc,all,2}$ | $m_{sc,all,3}$ | $m_{sc,all,4}$ | $m_{sc,all,5}$ | $m_{sc,all,6}$ | $m_{sc,all,7}$ | $m_{sc,all,8}$ | $m_{sc,all,9}$ |
| all, full data | 0.35 (0.37) | 0.31 (0.51) | 0.40 (0.60) | 0.30 (0.33) | 0.33 (0.44) | 0.27 (0.36) | 0.28 (0.32) | 0.30 (0.33) | 0.30 (0.28) | 0.33 (0.41) |
| all, 02-17-25 | 1.21 (0.90) | 25.05 (0.73) | 0.99 (0.57) | 2.76 (1.34) | 2.52 (1.16) | 1.99 (1.04) | 2.56 (1.60) | 1.42 (0.75) | 5.41 (3.77) | 4.09 (0.69) |
| all, 03-21-31 | 5.97 (3.51) | 4.02 (2.23) | 11.78 (3.83) | 21.94 (8.74) | 68.41 (35.51) | 84.00 (4.62) | 40.99 (10.58) | 3.31 (1.66) | 46.27 (15.55) | 19.08 (5.65) |
| all, 04-25-36 | 302.99 (90.27) | 288.36 (231.36) | 33.87 (5.90) | 429.79 (154.04) | 318.55 (154.16) | 234.59 (76.73) | 487.56 (153.25) | 104.20 (28.66) | 121.57 (46.30) | 137.46 (16.78) |

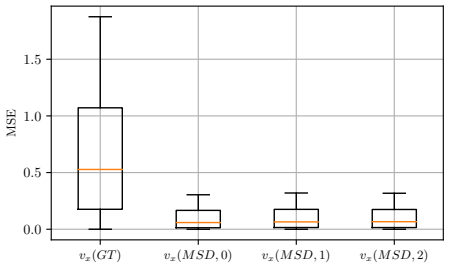
Table 5.14: SC approach: Final evaluation of models $m_{sc,all,1:10}$, using the $\mathcal{F}_{all}$ feature-set.

| $MSE_{all}\{v_{x,to}(SD), v_{x,to}(MSD)\}$ ($MSE_{test}\{v_{x,to}(SD), v_{x,to}(MSD)\}$) | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| data set | $m_{sc,vrh,0}$ | $m_{sc,vrh,1}$ | $m_{sc,vrh,2}$ | $m_{sc,vrh,3}$ | $m_{sc,vrh,4}$ | $m_{sc,vrh,5}$ | $m_{sc,vrh,6}$ | $m_{sc,vrh,7}$ | $m_{sc,vrh,8}$ | $m_{sc,vrh,9}$ |
| vrh, full data | 0.43 (0.39) | 0.42 (0.26) | 0.43 (0.25) | 0.45 (0.26) | 0.41 (0.29) | 0.40 (0.23) | 0.41 (0.25) | 0.44 (0.28) | 0.44 (0.25) | 0.57 (0.61) |
| vrh, 02-17-25 | 0.91 (0.93) | 1.04 (0.66) | 1.44 (2.48) | 0.63 (0.43) | 1.55 (3.64) | 0.67 (0.37) | 0.47 (0.27) | 0.63 (0.48) | 0.80 (1.87) | 0.69 (0.40) |
| vrh, 03-21-31 | 1.76 (3.39) | 1.53 (1.96) | 0.99 (1.36) | 1.01 (2.02) | 0.87 (0.50) | 1.33 (1.83) | 0.91 (0.75) | 1.27 (0.94) | 1.00 (0.82) | 0.69 (0.48) |
| vrh, 04-25-36 | 1.51 (0.93) | 3.76 (8.42) | 2.83 (1.85) | 4.02 (2.39) | 4.48 (16.32) | 2.91 (2.69) | 2.94 (3.72) | 6.37 (128.25) | 3.02 (10.40) | 3.30 (5.78) |

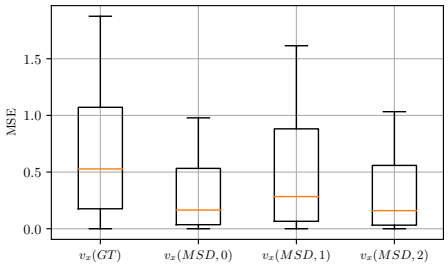
Table 5.15: SC approach: Final evaluation of models $m_{sc,vrh,1:10}$, using the $\mathcal{F}_{vrh}$ feature-set.



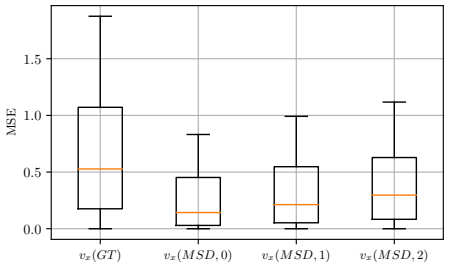

(a) NA approach with feature-set: $\mathcal{F}_{all}$, full data



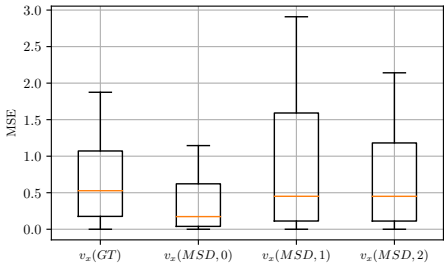

(b) NA approach with feature-set: $\mathcal{F}_{vrp}$, full data



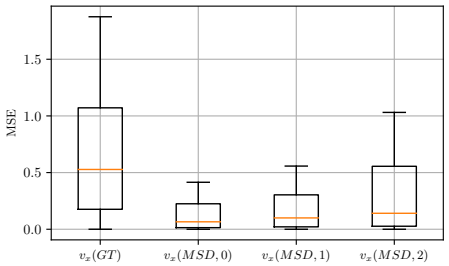

(c) NA approach with feature-set: $\mathcal{F}_{all}$, 02-17-25



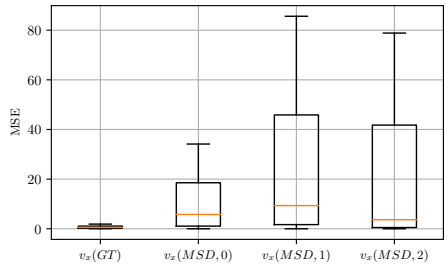

(d) NA approach with feature-set: $\mathcal{F}_{vrp}$, 02-17-25



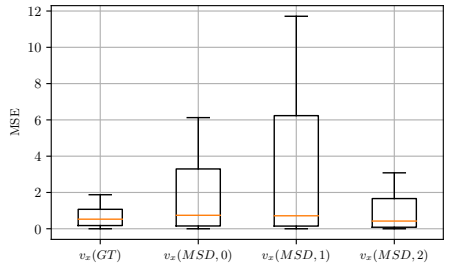

(e) NA approach with feature-set: $\mathcal{F}_{all}$, 03-21-31




(f) NA approach with feature-set: $\mathcal{F}_{vrp}$, 03-21-31

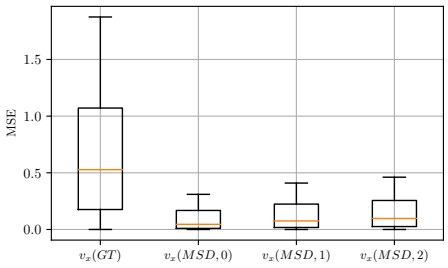



(g) NA approach with feature-set: $\mathcal{F}_{all}$, 04-25-36

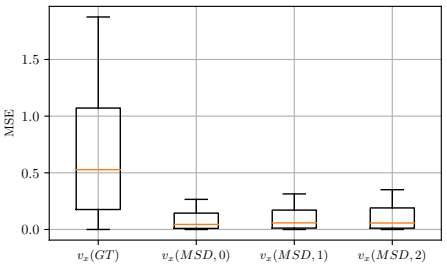



(h) NA approach with feature-set: $\mathcal{F}_{vrp}$, 04-25-36

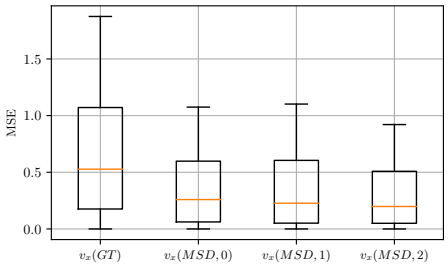
Figure 5.7: NA approach: Comparison of feature selection.



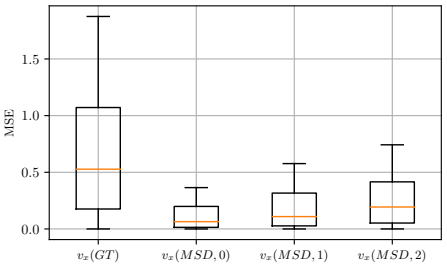

(a) SC approach with feature-set: $\mathcal{F}_{all}$, full data



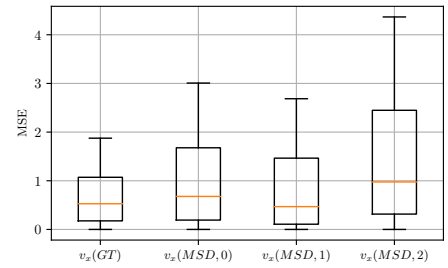

(b) SC approach with feature-set: $\mathcal{F}_{vrh}$, full data



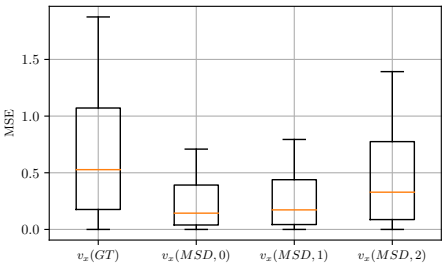

(c) SC approach with feature-set: $\mathcal{F}_{all}$, 02-17-25



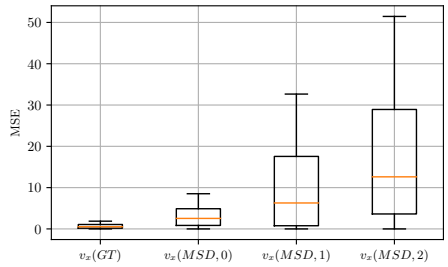

(d) SC approach with feature-set: $\mathcal{F}_{vrh}$, 02-17-25



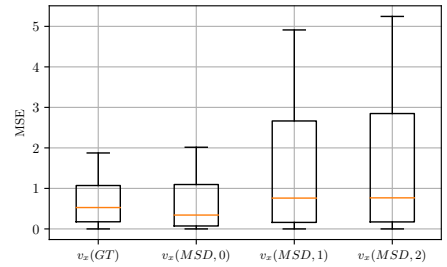

(e) SC approach with feature-set: $\mathcal{F}_{all}$, 03-21-31




(f) SC approach with feature-set: $\mathcal{F}_{vrh}$, 03-21-31

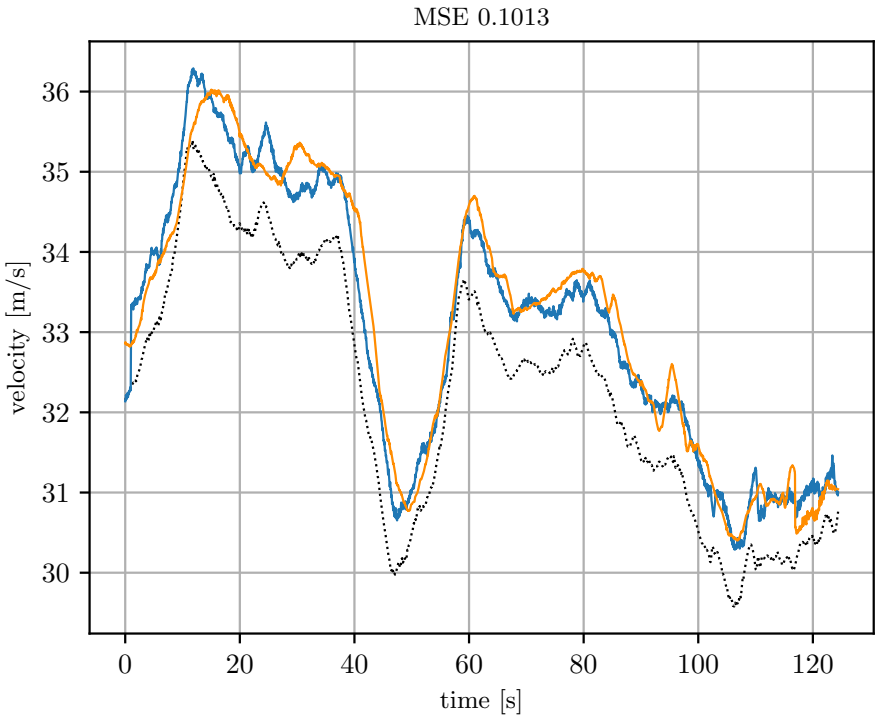



(g) SC approach with feature-set: $\mathcal{F}_{all}$, 04-25-36

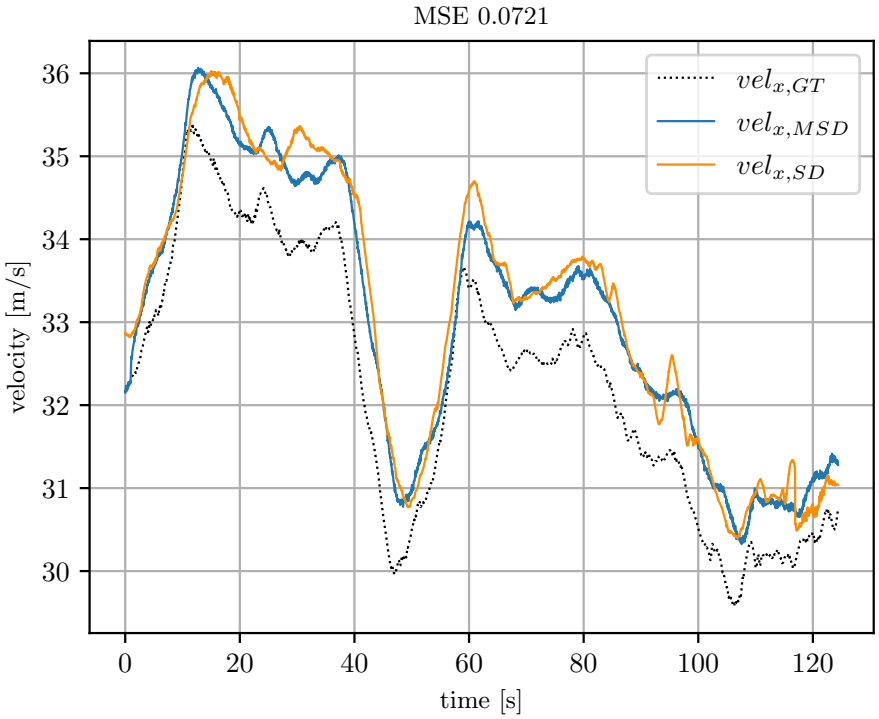



(h) SC approach with feature-set: $\mathcal{F}_{vrh}$, 04-25-36

Figure 5.8: SC approach: Comparison of feature selection.




(a) Model $m_{na,vrp,3}$ as best candidate of the NA approach.




(b) Model $m_{sc,vrh,5}$ as best candidate of the SC approach.

Figure 5.9: Prediction of $TT_2$ using the best candidates of both approaches. The dotted curve labels $v_{x,to}(GT)$, the orange curve $v_{x,to}(SD)$ and the blue curve $v_{x,to}(MSD)$.

## 5.2 Online Evaluation

The online evaluation is performed without the Ego vehicle's driving function in a virtualized environment and consists of the steps that are shown on the right side of Figure 4.1. This process takes a scenario-description and a pre-trained PM as input to simulate the scenario as SiL. The goal of this procedure is the online estimation of the sensor-error $\Delta S$, which is done by processing GT information from the scenario-description with the PM to approximate SD. As in Section 5.1, the PM is trained on a training data-set before it can be used in the online evaluation. The input-scenario for the simulation is based on a real-world test-drive on a German highway and is approximately 10 minutes long. For further usage, the data is reprocessed to extract the scenario-description. Through this process a description error $\Delta D_{GT}$ is introduced to the data-set, as cited in Section 3.3. Further on, all reprocessed variables, that are affected by $\Delta D_{GT}$, are labeled as $GT_r$.

Splitting the 10 minute long input-scenario results in 245 separate object traces. These tracks consist of TOs that move in the direction of the Ego vehicle, but also static objects and vehicles approaching from the opposite direction. The model is trained on a data-set, which only contains information about vehicles moving in a positive direction. A data-driven model is in general not capable of producing meaningful results, when evaluated on features that deviate too much from the training data. The generalization ability of the PM does not cover categorical differences, such as the movement of a TO in the opposite direction. For this reason, tracks are filtered for the evaluation by the following criteria to ensure comparable results. TOs that have a negative-oriented absolute velocity $v_{x,to}(GT_r)$ are removed from the data-set, since they are not represented by the training data. In addition, TOs with a relative distance $d_{x,rel}(GT_r)$ greater than 120 meters are removed from the set, as listed in Section 4.2. To receive reasonable estimates for models that use the SC approach, a settling time of at least one second must be ensured to fill the prediction queue of the PM. For this reason, traces that are shorter than four seconds are removed from the set. Applying these criteria prunes the list of tracks to a total number of 31 traces, with an average length of 14 seconds.

The model that leads to the best offline prediction is selected to evaluate the performance of the PM in the online prediction. The best candidate is based on the SC approach and uses NL = 1, NC = 16 as configuration of the LSTM network. This model is trained on the previously used input-data with the $\mathcal{F}_{vrh}$ feature-set, for a total amount 128 epochs, while using a LR = $1e-05$.

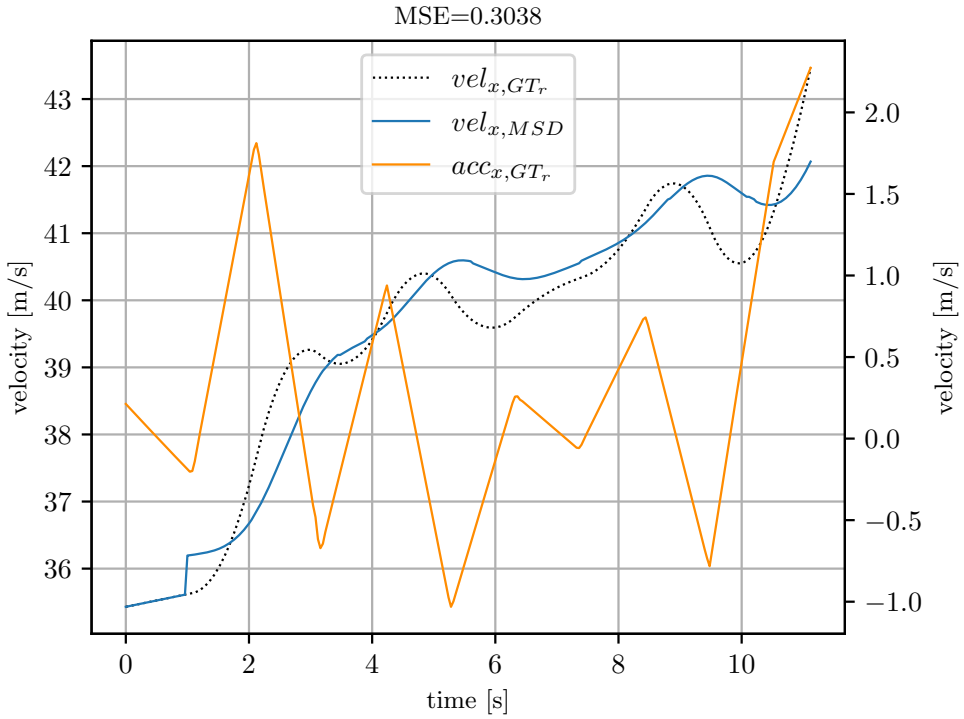
For the evaluation of the online prediction capabilities of the PM, two examples of the resulting model output are chosen to analyze the occurring effects. The selected predictions are shown in Figure 5.10. The black dotted and orange functions represent the reprocessed parameters $v_{x,to}(GT_r)$ $acc_{x,to}(GT_r)$ and the blue function labels the predicted $v_{x,to}(MSD)$. The MSE is calculated between $v_{x,to}(GT_r)$ and $v_{x,to}(MSD)$ to quantify the difference. The reprocessed velocity $v_{x,to}(GT_r)$ and acceleration $acc_{x,to}(GT_r)$ in Figure 5.10 show a non-naturalistic behavior. $v_{x,to}(GT_r)$ is represented as an artificially smooth progressing function and $acc_{x,to}(GT_r)$ consists of only piece-wise linear segments. As cited in Section 3.3, $v_{x,to}(GT_r)$ is the extracted version of $v_{x,to}(GT)$ and is approximated as dynamic set of splines. This set of splines does not contain any recognizable fluctuations. The source of the piece-wise linear progression of $acc_{x,to}(GT_r)$ can not be clearly identified. One possible explanation for the origin of $acc_{x,to}(GT_r)$ is that is calculated by taking the derivative over time of $v_{x,to}(GT_r)$. A function of $v_{x,to}(GT_r)$, that

is represented as splines of quadratic polynomials, would therefore lead to a piece-wise linear progression of $acc_{x,to}(GT_r)$.

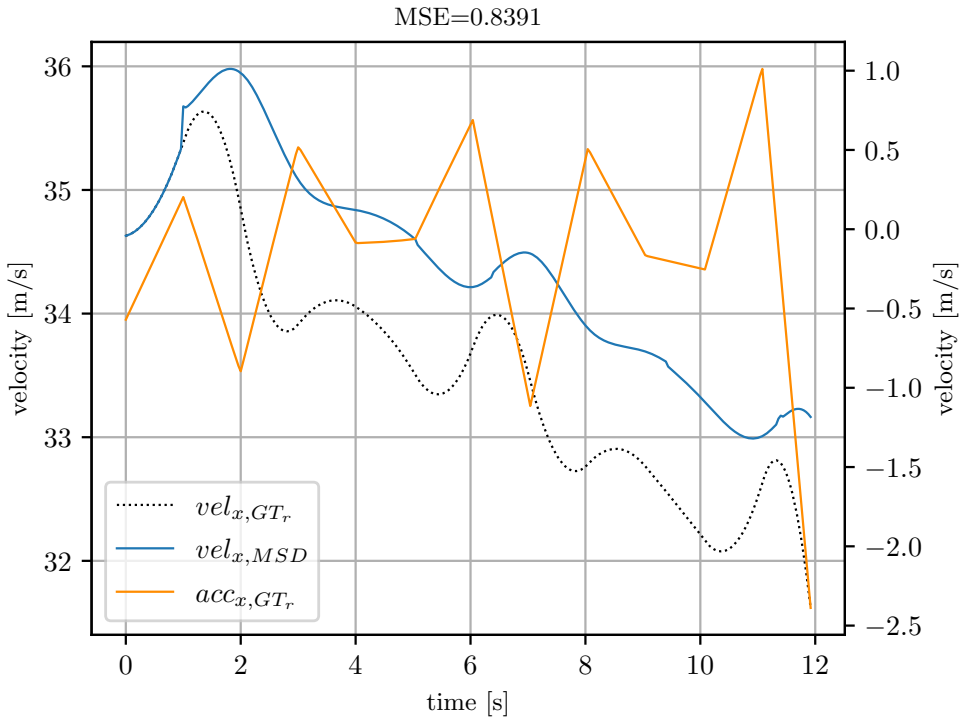
To assess the online evaluation, the comparison with real-world or reprocessed SD is crucial, similar to the offline evaluation. At the time of this thesis, no corresponding SD is available for this comparison. Using a full scenario-description, that consists of multiple object traces, as input for the PM, leads to a cumulative error that affects the positioning of the Ego vehicle. As cited in Section 3.3, the description error $\Delta D_{GT}$ is only responsible for a small portion of the difference between real-world and simulated data. Other simplifications, such as a simplified model for the Ego vehicle and an approximated environment model also affect this misalignment. Since this error affects the positioning of the Ego vehicle, the matching of corresponding SD and MSD is challenging. One possible solution to reduce the influence of this cumulative error is the generation of a separate scenario-description for every single object trace. This comes with a major increase in effort that exceeds the time scope of this thesis. In addition, the introduction of a matching algorithm, that is able to fit SD to the timely corresponding GT frames and therefore the predicted MSD, can help to avoid this problem. A comparison of MSD and SD is therefore not possible.

To qualitatively the performance of the PM, the same scenario is simulated with five separate models, that use different training configurations. Due to limitations in the simulation, models that use the $\mathcal{F}_{vrp}$ and $\mathcal{F}_{all}$ feature-set are excluded from the evaluation, since the required pitch of the Ego vehicle $p_{ego}(GT_r)$ is not accessible in the simulation environment. The resulting predictions of this test-series confirm, that models that are trained on real-world data struggle with the synthesized approximations. The noise-free inputs always result in a noise-free prediction of the SD, even though the PMs are trained on real-world data. In addition, not being able to access the SD, that corresponds to the predicted MSD, makes a quantitative statement of the assessment procedure difficult. Using the implemented PM for the online estimation can lead to promising results, since the system is able to process successively incoming data to make prediction on the corresponding SD. Also, the predicted MSD shown in Figure 5.10 follows the GT input and shows characteristics, such as a small time delay and an overshooting of changes in the velocity, similar to the MSD that is presented in Figure 5.9.

The online evaluation shows the analysis of the implemented PM in the simulation environment and examines its performance in a qualitatively assessment. The achieved results are promising, but the implemented solution cannot be evaluated to an extent, that would be sufficient for fully answering the second research question of this thesis. Therefore, a statement on the improvement of the virtual assessment process by including the PM cannot be provided entirely and only an outlook on the improvement abilities of this PM is presented.




(a) Accelerating maneuver in online evaluation.




(b) Breaking maneuver in online evaluation.

Figure 5.10: Comparison of two tracks from the online evaluation. The black dotted line labels $v_{x,to}(GT_r)$, the orange line labels $acc_{x,to}(GT_r)$ coming from the scenario description. The blue function labels the estimated $v_{x,to}(MSD)$.

# 6 Conclusions and Outlook

This Section summarizes the results that are obtained while working on this thesis and provides possible extensions to further develop this project. The goal of this work is the implementation of a PM, that is capable of estimating the perceived SD of an AV, when provided with GT information about a scenario. The PM is implemented using machine learning methods in the form of a LSTM network. Two different approaches are designed to find a solution for this task and a series of models is compared to optimize the parameterization of the network. After being trained on real-world data, the resulting PM is able to reproduce SD with an accuracy that exceeds the model requirements. In the offline evaluation, the PM manages to reproduce the signal characteristics of the SD, solely based on GT information. Due to difficulties in the evaluation process, the online evaluation can only provide a qualitative statement on the performance of the model inside the simulation framework.

At first, this thesis investigates the influence, that the composition and size of the training set has on the performance of the LSTM network. For this, the data-set is pruned by its underlying levels of dynamic. Both NA and SC approaches show a clear trend towards a better performance, when provided with a larger data-set for training. This evaluation focuses on the predictive capabilities, that the implemented PM has during highway drives. Even in this narrow field of traffic scenarios, the addition of more training data is beneficial for the overall performance of the PM. Neural networks are capable of finding correlations in large amounts of data and the LSTM architecture has proven to be a suitable solution for the processing of sequenced data. This thesis uses a combination of all available input features or single additional feature to GT velocity and relative distance between the Ego vehicle and traced TO. The result of this analysis is that a reduced number of features can lead to better predictions than using the full data-set. A different selection of feature-combinations can therefore be beneficial for the performance of the PM and could be investigated further. Data analysis tools such as Principal Component Analysis (PCA) can help to find a more sophisticated selection of input features, by maximizing the variance of the data set.

To improve the assessment process, stricter requirements on the performance of the PM can be beneficial. This work aims to implement a PM that is able to reproduce SD with a higher accuracy than the direct comparison with GT data. The offline evaluation shows that this minimum requirement can be exceeded by both chosen network architectures. A more detailed definition of test cases can also improve the assessment during different situations and can expose potential weaknesses of the model. Since this work concentrates on common maneuvers during highway-drives, such as following a vehicle or overtaking, a detailed assessment of specific traffic situations cannot be provided. Therefore, a more detailed description of tested scenarios can help the comparison with other model architectures and opens up the way of potential hybrid solutions. The comparison with classic solutions from control theory can further investigate the suitability of the implemented PM on this prediction task.

Even though this thesis covers a variety of training configurations, a more elaborate inspection of those possibilities can improve the performance of the model. The most

promising SC approach uses a prediction queue to feed previous outputs back into the network. The prediction queue is implemented using a static length of 25 values, which correspond to one second of test-data. The introduction of a dynamic queue that changes its size depending on the performed maneuver could not only have a positive influence on the computational performance, but also improve the accuracy of the PM. Since the dynamic change of the network architecture during the prediction phase is not possible, a solution for this approach is not trivial. In addition, the inclusion of more than one time-step as input for each layer of the LSTM can be beneficial.

To receive results for the online evaluation that are comparable to those coming from the offline evaluation, the access to all input parameters during the simulation is crucial. The most promising candidate of the first evaluation step uses the pitch of the Ego vehicle to make a prediction about the SD. Adding this feature to the simulation environment enables a broader selection of PMs. The most important improvement of the online testing procedure is the addition of a trace-matching algorithm, that is able to match corresponding SD and MSD. Only with a profound comparison of these quantities, the second research question of this thesis can be answered to a full extent. To bring additional benefits to the virtual assessment of automated driving, a closed-loop evaluation using the implemented PM is advisable. The goal of this procedure is the replication of all decisions an AV takes during a real-world drive inside the simulation environment. A congruent base for this decision enables the replacement of real-world test with a virtualized version.

This thesis shows the successful development and implementation of a PM, that is able to approximate the SD of an AV. The implemented software pipeline consists of all components that are necessary to prepare the input sequences for the model, train the neural network for the prediction task and import the fully trained model into the provided simulation framework. Therefore, two different network structures are designed and optimized. The predicted MSD shows similar characteristics as the real-world SD and results in a smaller estimation error than the corresponding GT. The influence of different training properties to increase the performance of the PM are reviewed and improved. Furthermore, a framework for the online estimation of SD inside the provided simulation environment has been prepared and tested. Future work can use the results of this thesis as a baseline to further improve the prediction capabilities of PMs.

# Acronyms

**ACC** Active Cruise Control

**AD** Autonomous Driving

**ADAS** Advanced Driver Assistant System

**ADS** Autonomous Driving System

**AI** Artificial Intelligence

**AV** Autonomous Vehicle

**BPTT** Backpropagation Through Time

**DNN** Deep Neural Network

**ERSO** European Road Safety Observatory

**FOT** Field Operating Test

**GPU** Graphic Processing Unit

**GT** Ground Truth

**HiL** Hardware in the Loop

**KPI** Key Performance Index

**LKAS** Lane Keeping Assistant System

**LR** Learning Rate

**LSTM** Long Short-Term Memory

**ML** Machine Learning

**MLP** Multi Layer Perceptron

**MSD** Model Sensor Data

**MSE** Mean Squared Error

**NA** Non Autoregression

**NC** Number of Cells per Layer

**NDS** Naturalistic Driving Study

**NE** Number of Training Epochs

**NL** Number of Network Layers

**NN** Neural Network

**PCA** Principal Component Analysis

**PM** Perception Model

**RNN** Recurrent Neural Network

**SAE** Society of Automotive Engineers

**SC** Single Component Autoregression

**SD** Sensor Data

**SF** Sensor Fusion

**SiL** Software in the Loop

**SM** Sensor Model

**TO** Traffic Object

**TT** Test Track

**TTC** Time to Collision

**TTR** Time to React

# Bibliography


[1] ERSO, "Annual Accident Report 2018." [Online]. Available: https://ec.europa.eu/transport/road_safety/sites/roadsafety/files/pdf/statistics/dacota/asr2018.pdf (cit. on p. 1).

[2] D. Watzenig and M. Horn, *Automated Driving: Safer and More Efficient Future Driving*. Springer, Sep. 23, 2016, 619 pp., ISBN: 978-3-319-31895-0. Google Books: 1FAiDQAAQBAJ (cit. on pp. 1, 2).

[3] Đ. Petrović, R. Mijailović, and D. Pešić, "Traffic Accidents with Autonomous Vehicles: Type of Collisions, Manoeuvres and Errors of Conventional Vehicles' Drivers," *Transportation Research Procedia*, vol. 45, pp. 161–168, 2020, ISSN: 23521465. DOI: 10.1016/j.trpro.2020.03.003. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S2352146520301654 (visited on 01/11/2021) (cit. on p. 1).

[4] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: Opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, Jul. 2015, ISSN: 09658564. DOI: 10.1016/j.tra.2015.04.003. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0965856415000804 (visited on 01/11/2021) (cit. on p. 1).

[5] T. Winkle, "Sicherheitspotenzial automatisierter Fahrzeuge: Erkenntnisse aus der Unfallforschung," in *Autonomes Fahren*, Springer Vieweg, Berlin, Heidelberg, 2015, pp. 351–376 (cit. on p. 1).

[6] F. Liu, F. Zhao, Z. Liu, and H. Hao, "Can autonomous vehicle reduce greenhouse gas emissions? A country-level evaluation," *Energy Policy*, vol. 132, pp. 462–473, Sep. 2019, ISSN: 03014215. DOI: 10.1016/j.enpol.2019.06.013. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0301421519303830 (visited on 01/11/2021) (cit. on p. 1).

[7] H. Claypool, A. Bin-Nun, and J. Gerlach, "Self-Driving Cars: The impact on people with disabilites," p. 35, (cit. on p. 1).

[8] "J3016B: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International." (), [Online]. Available: https://www.sae.org/standards/content/j3016_201806/ (visited on 01/06/2021) (cit. on p. 1).

[9] S. Wagner, A. Knoll, K. Groh, T. Kühbeck, D. Watzenig, and L. Eckstein, "Virtual Assessment of Automated Driving: Methodology, Challenges, and Lessons Learned," vol. 2, no. 4, p. 16, 2019 (cit. on pp. 1–3, 16, 17).

[10] "BMW says self-driving car to be Level 5 capable by 2021," Automotive News. (2017), [Online]. Available: https://www.autonews.com/article/20170316/MOBILITY/170319877/bmw-says-self-driving-car-to-be-level-5-capable-by-2021 (visited on 01/05/2021) (cit. on p. 2).

[11] F. Lambert. "Tesla CEO Elon Musk: 'self-driving will encompass all modes of driving by the end of next year'," Electrek. (Mar. 11, 2018), [Online]. Available: https://electrek.co/2018/03/11/tesla-ceo-elon-musk-self-driving-next-year/ (visited on 01/05/2021) (cit. on p. 2).

[12] P. Kafka, "The Automotive Standard ISO 26262, the Innovative Driver for Enhanced Safety Assessment & Technology for Motor Cars," *Procedia Engineering*, vol. 45, pp. 2–10, 2012, ISSN: 18777058. DOI: 10.1016/j.proeng.2012.08.112. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877705812031244 (visited on 01/05/2021) (cit. on p. 2).

[13] W. Wachenfeld and H. Winner, "Virtual Assessment of Automation in Field Operation A New Runtime Validation Method," p. 10, (cit. on p. 2).

[14] H. Winner, S. Hakuli, and G. Wolf, *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort : mit 550 Abbildungen und 45 Tabellen*. Springer-Verlag, Jan. 1, 2009, 732 pp., ISBN: 978-3-8348-0287-3. Google Books: UqlYIaso3yoC (cit. on p. 2).

[15] H. Martin, K. Tschabuschnig, O. Bridal, and D. Watzenig, "Functional Safety of Automated Driving Systems: Does ISO 26262 Meet the Challenges?" In Sep. 1, 2017, pp. 387–416, ISBN: 978-3-319-31893-6. DOI: 10.1007/978-3-319-31895-0_16 (cit. on p. 2).

[16] V. Gebhardt, G. M. Rieger, J. Mottok, and C. Gießelbach, *Funktionale Sicherheit Nach ISO 26262: Ein Praxisleitfaden Zur Umsetzung*. dpunkt. verlag, 2013, ISBN: 3-86491-339-X (cit. on p. 2).

[17] W. Wachenfeld and H. Winner, "Die freigabe des autonomen fahrens," in *Autonomes Fahren*, Springer Vieweg, Berlin, Heidelberg, 2015, pp. 439–464 (cit. on pp. 2, 16).

[18] T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, "A decade of agile methodologies: Towards explaining agile software development," *Journal of Systems and Software*, vol. 85, no. 6, pp. 1213–1221, Jun. 2012, ISSN: 01641212. DOI: 10.1016/j.jss.2012.02.033. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0164121212000532 (visited on 01/13/2021) (cit. on p. 2).

[19] T. Helmer, L. Wang, K. Kompass, and R. Kates, "Safety Performance Assessment of Assisted and Automated Driving by Virtual Experiments: Stochastic Microscopic Traffic Simulation as Knowledge Synthesis," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, Sep. 2015, pp. 2019–2023. DOI: 10.1109/ITSC.2015.327 (cit. on p. 2).

[20] K. Fowler, *Developing and Managing Embedded Systems and Products: Methods, Techniques, Tools, Processes, and Teamwork*. Elsevier, 2014, ISBN: 0-12-405863-9 (cit. on p. 3).

[21] S. Ulbrich, T. Menzel, A. Reschka, F. Schuldt, and M. Maurer, "Defining and Substantiating the Terms Scene, Situation, and Scenario for Automated Driving," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, Gran Canaria, Spain: IEEE, Sep. 2015, pp. 982–988, ISBN: 978-1-4673-6596-3. DOI: 10.1109/ITSC.2015.164. [Online]. Available: http://ieeexplore.ieee.org/document/7313256/ (visited on 01/13/2021) (cit. on p. 2).

[22] K. Groh, S. Wagner, T. Kuehbeck, and A. Knoll, "Simulation and Its Contribution to Evaluate Highly Automated Driving Functions," presented at the WCX SAE World Congress Experience, Apr. 2, 2019, pp. 2019-01–0140. DOI: 10.4271/2019-01-0140. [Online]. Available: https://www.sae.org/content/2019-01-0140/ (visited on 01/05/2021) (cit. on pp. 2, 3, 16–18, 20, 21).

[23] P. Cao, W. Wachenfeld, and H. Winner, "Perception sensor modeling for virtual validation of automated driving," *it - Information Technology*, vol. 57, no. 4, Jan. 28, 2015, ISSN: 1611-2776, 2196-7032. DOI: 10.1515/itit-2015-0006. [Online]. Available: https://www.degruyter.com/doi/10.1515/itit-2015-0006 (visited on 01/12/2021) (cit. on p. 3).

[24] M. Sigl, C. Schütz, S. Wagner, and D. Watzenig, "Modeling Perception Errors of Automated Vehicles," presented at the 2021 (submitted to) IEEE Vehicular Technology Conference (VTC), Apr. 2021, pp. 1–6 (cit. on pp. 3, 4, 15, 20, 21, 26).

[25] A. Piazzoni, J. Cherian, M. Slavik, and J. Dauwels. "Modeling Sensing and Perception Errors towards Robust Decision Making in Autonomous Vehicles." arXiv: 2001.11695 [cs]. (Jan. 31, 2020), [Online]. Available: http://arxiv.org/abs/2001.11695 (visited on 01/05/2021) (cit. on p. 3).

[26] S. A. Billings, *Nonlinear System Identification: NARMAX Methods in the Time, Frequency, and Spatio-Temporal Domains*. John Wiley & Sons, 2013, ISBN: 1-119-94359-0 (cit. on pp. 9, 10).

[27] O. Nelles, *Nonlinear System Identification: From Classical Approaches to Neural Networks, Fuzzy Models, and Gaussian Processes*. Cham: Springer International Publishing, 2020, ISBN: 978-3-030-47438-6 978-3-030-47439-3. DOI: 10.1007/978-3-030-47439-3. [Online]. Available: http://link.springer.com/10.1007/978-3-030-47439-3 (visited on 01/16/2021) (cit. on pp. 9, 10, 25).

[28] J. Sjöberg, H. Hjalmarsson, and L. Ljung, "Neural Networks in System Identification," *IFAC Proceedings Volumes*, vol. 27, no. 8, pp. 359–382, Jul. 1994, ISSN: 14746670. DOI: 10.1016/S1474-6670(17)47737-8. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1474667017477378 (visited on 02/05/2021) (cit. on p. 9).

[29] C. M. Bishop, *Pattern Recognition and Machine Learning*, ser. Information Science and Statistics. New York: Springer, 2006, 738 pp., ISBN: 978-0-387-31073-2 (cit. on pp. 10–14, 26).

[30] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958, ISSN: 1939-1471, 0033-295X. DOI: 10.1037/h0042519. [Online]. Available: http://doi.apa.org/getdoi.cfm?doi=10.1037/h0042519 (visited on 01/16/2021) (cit. on p. 11).

[31] A. B. Novikoff, "On convergence proofs for perceptrons," STANFORD RESEARCH INST MENLO PARK CA, 1963 (cit. on p. 11).

[32] S. Russell and P. Norvig, "IN ARTIFICIAL INTELLIGENCE," *Artificial Intelligence*, p. 2145, (cit. on p. 11).

[33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ San Diego La Jolla Inst for Cognitive Science, 1985 (cit. on pp. 11, 13).

[34] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989 (cit. on p. 11).

[35] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, pp. 873–880 (cit. on p. 11).

[36] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ser. Adaptive Computation and Machine Learning. Cambridge, Massachusetts: The MIT Press, 2016, 775 pp., ISBN: 978-0-262-03561-3 (cit. on pp. 11–13, 27).

[37] A. Graves. "Generating sequences with recurrent neural networks." arXiv: 1308.0850. (2013) (cit. on pp. 14, 25).

[38] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998 (cit. on p. 14).

[39] P. Werbos, "Backpropagation through time: What it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct./1990, ISSN: 00189219. DOI: 10.1109/5.58337. [Online]. Available: http://ieeexplore.ieee.org/document/58337/ (visited on 01/18/2021) (cit. on p. 14).

[40] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997 (cit. on p. 14).

[41] C. Olah. "Understanding LSTM Networks – colah's blog." (), [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 01/27/2021) (cit. on p. 15).

[42] K. Groh, T. Kuehbeck, B. Fleischmann, M. Schiementz, and C. C. Chibelushi, "Towards a Scenario-Based Assessment Method for Highly Automated Driving Functions," p. 7, (cit. on pp. 16, 17).

[43] S. Wagner, K. Groh, T. Kühbeck, and A. Knoll, "Towards Cross-Verification and Use of Simulation in the Assessment of Automated Driving," in *2019 IEEE Intelligent Vehicles Symposium (IV)*, Jun. 2019, pp. 1589–1596. DOI: 10.1109/IVS.2019.8814268 (cit. on p. 16).

[44] D. L. Hall and J. Llinas, "An introduction to multisensor data fusion," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 6–23, Jan. 1997, ISSN: 1558-2256. DOI: 10.1109/5.554205 (cit. on p. 16).

[45] W. A. Abdulhafiz and A. Khamis, "Handling Data Uncertainty and Inconsistency Using Multisensor Data Fusion.," *Advances in Artificial Intelligence (16877470)*, 2013 (cit. on p. 16).

[46] D. Notz, M. Sigl, T. Kuhbeck, S. Wagner, K. Groh, C. Schütz, and D. Watzenig, "Methods for Improving the Accuracy of the Virtual Assessment of Autonomous Driving," in *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*, Graz, Austria: IEEE, Nov. 2019, pp. 1–6, ISBN: 978-1-72810-142-2. DOI: 10.1109/ICCVE45908.2019.8965040. [Online]. Available: https://ieeexplore.ieee.org/document/8965040/ (visited on 01/04/2021) (cit. on pp. 16–18).

[47] J. Hillenbrand, A. M. Spieker, and K. Kroschel, "A Multilevel Collision Mitigation Approach—Its Situation Assessment, Decision Making, and Performance Tradeoffs," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 4, pp. 528–540, Dec. 2006, ISSN: 1558-0016. DOI: 10.1109/TITS.2006.883115 (cit. on p. 17).

[48] S. Wagner, K. Groh, T. Kuhbeck, M. Dorfel, and A. Knoll, "Using time-to-react based on naturalistic traffic object behavior for scenario-based risk assessment of automated driving," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2018, pp. 1521–1528, ISBN: 1-5386-4452-5 (cit. on p. 17).

[49] H. F. Durrant-Whyte, "Sensor Models and Multisensor Integration," p. 17, (cit. on p. 17).

[50] M. Holder, P. Rosenberger, H. Winner, T. D'hondt, V. P. Makkapati, M. Maier, H. Schreiber, Z. Magosi, Z. Slavik, O. Bringmann, and W. Rosenstiel, "Measurements revealing Challenges in Radar Sensor Modeling for Virtual Validation of Autonomous Driving," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Nov. 2018, pp. 2616–2622. DOI: 10.1109/ITSC.2018.8569423 (cit. on p. 18).

[51] N. Hirsenkorn, T. Hanke, A. Rauch, B. Dehlink, R. Rasshofer, and E. Biebl, "A non-parametric approach for modeling sensor behavior," in *2015 16th International Radar Symposium (IRS)*, Dresden, Germany: IEEE, Jun. 2015, pp. 131–136, ISBN: 978-3-95404-853-3. DOI: 10.1109/IRS.2015.7226346. [Online]. Available: http://ieeexplore.ieee.org/document/7226346/ (visited on 01/14/2021) (cit. on p. 18).

[52] M. Herrmann and H. Schön, "Efficient Sensor Development Using Raw Signal Interfaces," in *Fahrerassistenzsysteme 2018*, Springer, 2019, pp. 30–39 (cit. on p. 18).

[53] C. van Driesten and T. Schaller, "Overall Approach to Standardize AD Sensor Interfaces: Simulation and Real Vehicle," in *Fahrerassistenzsysteme 2018*, Springer, 2019, pp. 47–55 (cit. on p. 18).

[54] G. Welch and G. Bishop, "An introduction to the Kalman filter," 1995 (cit. on p. 18).

[55] P. Kadlec, B. Gabrys, and S. Strandt, "Data-driven Soft Sensors in the process industry," *Computers & Chemical Engineering*, vol. 33, no. 4, pp. 795–814, Apr. 2009, ISSN: 00981354. DOI: 10.1016/j.compchemeng.2008.12.012. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0098135409000076 (visited on 02/11/2021) (cit. on pp. 18, 25).

[56] W. Ke, D. Huang, F. Yang, and Y. Jiang, "Soft sensor development and applications based on LSTM in deep neural networks," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, 2017, pp. 1–6, ISBN: 1-5386-2726-4 (cit. on pp. 18, 25).

[57] X. Yuan, L. Li, and Y. Wang, "Nonlinear dynamic soft sensor modeling with supervised long short-term memory network," *IEEE transactions on industrial informatics*, vol. 16, no. 5, pp. 3168–3176, 2019 (cit. on pp. 18, 25).

[58] A. Suhre and W. Malik, "Simulating object lists using neural networks in automotive radar," in *2018 19th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE)*, IEEE, 2018, pp. 1–5, ISBN: 1-5386-2359-5 (cit. on p. 20).

[59] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise Reduction in Speech Processing*, Springer, 2009, pp. 1–4 (cit. on p. 22).

[60] M. Kutila, P. Pyykönen, W. Ritter, O. Sawade, and B. Schäufele, "Automotive LIDAR sensor development scenarios for harsh weather conditions," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, IEEE, 2016, pp. 265–270, ISBN: 1-5090-1889-1 (cit. on p. 24).

[61] S. M. Patole, M. Torlak, D. Wang, and M. Ali, "Automotive radars: A review of signal processing techniques," *IEEE Signal Processing Magazine*, vol. 34, no. 2, pp. 22–35, 2017 (cit. on p. 24).

[62] M. Khader and S. Cherian, "An Introduction to Automotive LIDAR," *Texas Instruments*, 2018 (cit. on p. 24).

[63] M. E. Warren, "Automotive LIDAR technology," in *2019 Symposium on VLSI Circuits*, IEEE, 2019, pp. C254–C255, ISBN: 4-86348-720-7 (cit. on p. 24).

[64] S. Chen, S. A. Billings, and P. M. Grant, "Non-linear system identification using neural networks," *International journal of control*, vol. 51, no. 6, pp. 1191–1214, 1990 (cit. on p. 25).

[65] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski, "DeepAR: Probabilistic forecasting with autoregressive recurrent networks," *International Journal of Forecasting*, vol. 36, no. 3, pp. 1181–1191, 2020 (cit. on p. 25).

[66] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization." arXiv: 1412.6980. (2014) (cit. on p. 27).