Universität Duisburg-Essen · Campus Duisburg

Fakultät für Ingenieurwissenschaften

Abteilung Maschinenbau

Lehrstuhl für Mechatronik

Masterarbeit

zur Erlangung des Grades eines

Master of Science Wirtschaftsingenieurwesen (Vertiefung Mechatronik)

Thema: **Erstellung eines Radaufhängungsmodells mithilfe von neuronalen Netzen**

Neural Network-Based Wheel Suspension Modeling

vorgelegt von: **Jan Seeger**
geb. am 31.07.1996 in Duisburg

Erstgutachter: **Prof. Dr.-Ing. Dr. h. c. Dieter Schramm**

Zweitgutachter: **Dr.-Ing. Niko Maas**

Betreuer: **Dr.-Ing. Frédéric Kracht**

Duisburg, 16.07.2020

Abstract

Wheel suspension models relying on differential algebraic equation solvers for simulations are computationally resourceful and often not hard real-time capable. In this work, a neural network approach is developed that is based on the NARX model. A multibody vehicle simulation in ADAMS CAR and a high-precision object-oriented SIMULINK-model are utilized to obtain adequate datasets for NN training and testing purposes. Different settings of the model are tried. The final NARX model, containing 5 hidden nodes, is trained in open- and closed-loop configuration by BAYESIAN Regularization. The generalization ability is proven in several different, driving-situation-based test instances. It is shown that the NN can model wheel carrier acceleration and spring / damper reaction force effectively while requiring significantly lower computation time than a DAE-based simulation.

Keywords: wheel suspension modeling, double wishbone suspension, vehicle dynamics, real-time, kinematic loop, neural network modeling, recurrent neural network, NARX

List of Contents

List of Figures

List of Tables

List of Abbreviations

AR	Autoregressive
ARMA	Autoregressive moving average
BP	Backpropagation
BPTT	Backpropagation through time
CM	Center of mass
CL	Closed loop
CAD	Computer aided design
DoF	Degree of freedom
DAE	Differential Algebraic Equations
FEM	Finite element method
GPU	Graphic processing unit
GUI	Graphic user interface
HPO	Hyperparameter optimization
HP	Hyperparameter
ISO	International Organization for Standardization
LMBP	LEVENBERG-MARQUARDT Backpropagation
LSTM	Long short-term memory
ML	Machine learning
MAE	Mean absolute error
MSW	Mean square weights
MSE	Mean squared error
MSEreg	Mean squared error regularized
MA	Moving average

MBS	Multibody system
MIMO	Multiple input multiple output
NN	Neural network
NVH	Noise, vibration, harshness
NARMA	Non-linear autoregressive moving average
NARMAX	Nonlinear autoregressive moving average with exogenous inputs
NARX	Nonlinear autoregressive with exogenous inputs
nMAE	Normalized mean absolute error
OL	Open loop
ODE	Ordinary differential equation
PSD	Power spectral density
PID	Proportional-integral-derivative
RNN	Recurrent neural network
e.V.	Registered association in Germany
SISO	Single input single output
SLN	Single layer network
SUV	Sport utility vehicle
wrt	With respect to

List of Symbols

$\boldsymbol{a}$	Acceleration error
$a(k)$	Delay block output at timestep k
c_D	Damper constant
d_u	Input delay
d_y	Feedback delay
E	Total error
$E(k)$	Error at timestep k
e_i	DoF of p bodies
$e(k)$	Independent error input to output nodes
f_G	Degree of freedom
F	Transfer function
$\boldsymbol{F}$	Force vector
$\boldsymbol{F}_G$	Gravitational force (at CM)
G_d	Displacement PSD
G_e, G_s, G_a	Road roughness modeling factors
h_{max}	Maximum step size
$I_{CM,i}$	Moment of inertia of a body i with respect to CM
J	Number of joints
K	Number of timesteps
k	Timestep
k_S	Spring stiffness coefficient
L	Kinematic loop
l	Spring length

l_0	Free spring length
$\boldsymbol{M}$	Torque vector
M_x	Overturning moment
M_y	Rolling resistance
M_z	Aligning torque
m_i	Mass of body i
m	Hidden node index
Mu	BP control parameter
N	Number of nodes in hidden layer
N_u	Number of input nodes
N_y	Number of output nodes
n_u	Maximum input delay
n_y	Maximum feedback delay
P	Prismatic joint
p	Number of bodies
p_{NARX}	Number of optimizable parameters
q	Number of independent mounts and joints
R	Revolute joint
R^2	Coefficient of determination
$\boldsymbol{r}$	Position vector
S	Spherical joint
s	Steering rack displacement
$T(k)$	Target at timestep k

t	Time
$u(k)$	System input at timestep k
$\boldsymbol{v}$	Velocity vector
$\boldsymbol{w}$	Weight vector as input to a node
w_0	Bias input
w_i	Weight of node input i
$\boldsymbol{x}$	Generalized coordinate vector
$\boldsymbol{x}(k)$	Node input vector at timestep k
$\mathrm{y}(k)$	System output at timestep k
y_{PAST}	Previous system output
z^{-1}	One-step delay

Greek alphabet:

$\boldsymbol{\alpha}$	Angle acceleration vector
β	Generalized coordinate
ξ	Regularization performance parameter
Φ_m	Activation function of hidden node m
π	Ratio of a circle's circumference to its diameter
τ	Time constant
$\ddot{\psi}_i, \ddot{\theta}_i, \ddot{\varphi}_i$	Kardan accelerations

1 Introduction

The problem of modeling a dynamical system is to create an idealized mathematical description of the observed real-world system. Once this model is obtained, it can be used for *simulations* which are widely established in automotive engineering in order to keep prototyping efforts and costs low. In the industry, vehicle models serve as virtual replacement of prototypes and make efficient design and development on full vehicle-, subsystem- or component level possible. For the modeling of *vehicle dynamics*, theoretical modeling approaches are typically applied by formulating the model based on physical laws. In contrast, far less popular experimental modeling identifies the model structure based on the input-to-output-relationship of experimental data [1]. Being not only known for their system identification abilities, *neural networks (NNs)* nowadays receive a lot of attention among various fields of research due to their flexibility and success in many scientific and commercial applications. Their basic mathematical concept has been studied for decades. Still, excessive computational resources are required for many tasks that only recently have become more accessible and therefore offer new possibilities in many research areas such as suspension modeling.

Due to the *kinematic loops* in their topological structure, suspension models often require numerical methods to iteratively compute an exact analytic solution for implicit differential equations. Using these computationally intensive though very accurate models in simulations, the *real-time capability* is insufficient for the use in real-time applications such as driving simulators [2]. Instead, initially trained neural networks efficiently compute their outputs in a configuration of multiple parallel simple processing units that essentially employ a pre-defined function of the sum of their inputs. At the same time, this parallel computing architecture that is opaque for the user – in other words results in a *black-box* – is considered the biggest flaw of NNs [3].

The behavior of *multibody systems* can be very complex and hard to describe mathematically. For non-linear system an analytical model cannot be straightforward transformed into a non-linear neural network model. The task of modeling a non-linear dynamical system is to find and apply a model that can represent the inherent non-linear dynamics of the underlying system. At best, it can reproduce the input-to-output-behavior of the time-series for all of the data's interest range [4]. Ultimately, to integrate the neural network suspension model into a complete vehicle model, it is advantageous to obtain a block representation in a system modeling software like SIMULINK.

An object-oriented analytical model of a double wishbone suspension [5] has been recently developed at the CHAIR OF MECHATRONICS at the UNIVERSITY DUISBURG-ESSEN and resembles the suspension of the FORMULA STUDENT Racecar A40-02. A validated reference model of the complete vehicle exists in the multibody simulation environment ADAMS CAR [6–8].

To ensure the proper simulation ability of the neural network model, the network's parameters need to be optimized by presenting *input* and *target* (true output) data to a *learning algorithm*. After the network is fully trained, new input can be applied and the output is compared to the test targets. However, this implies that adequate data for both *learning* and *testing* purposes is to be generated beforehand.

Many neural network applications have been developed for the automotive industry in the past. New computing capabilities make learning of larger datasets and more complex modeling techniques possible. In manufacturing, development and research, NN technology is mostly used as a complement to incumbent methods because of its black-box nature. For in-vehicle deployment, limited computational resources impose even more constraints on the application of NNs [9].

Some research has been conducted regarding the use of NN in various topics of vehicle dynamics. A few examples shall be named: YIM AND OH [10] use NNs for the prediction of the kinematic state of the complete vehicle from real measurements. ALEXA ET. AL. [11] model the vehicle's velocity based on the road irregularities, employing a recurrent neural network. On component level, DYE AND LANKARANI [12] explore modeling tires from with the help of NN from experimental testing data. Regarding the suspension dynamics, many papers have been published concerned with the modeling or control of semi-active or active dampers [13–16]. Moreover, BASH's [17] neural network predicts the response of single discretized suspension arm. The research on modeling complete suspension (sub-)systems is performed to a smaller extent: In [18], ALI AND FRIMPONG successfully model the accelerations on the operator seat as result of external forces applied to large dump trucks using hydro-pneumatic struts with NNs, fuzzy logic and hybrid models. MORGAN AND YAO [19] apply a feed-forward NN to model the behavior of an active sprung mass trailer suspension used in static tests. GUARNERI ET. AL. [20] design a NARX network that takes measured road cleats, damping levels and speed as input to compute the force transmitted to the chassis. The established model includes tire and suspension dynamics with varying damper configuration. The NN results exceed those of the multibody simulation.

Outside of automotive engineering, some efforts are made to replace analytic model containing kinematic loops with NNs. For instance, SALEHINIA ET. AL. [21] deal with the forward kinematics of a parallel manipulator. A recurrent network is used to map limb lengths to the position and angles of the manipulator platform.

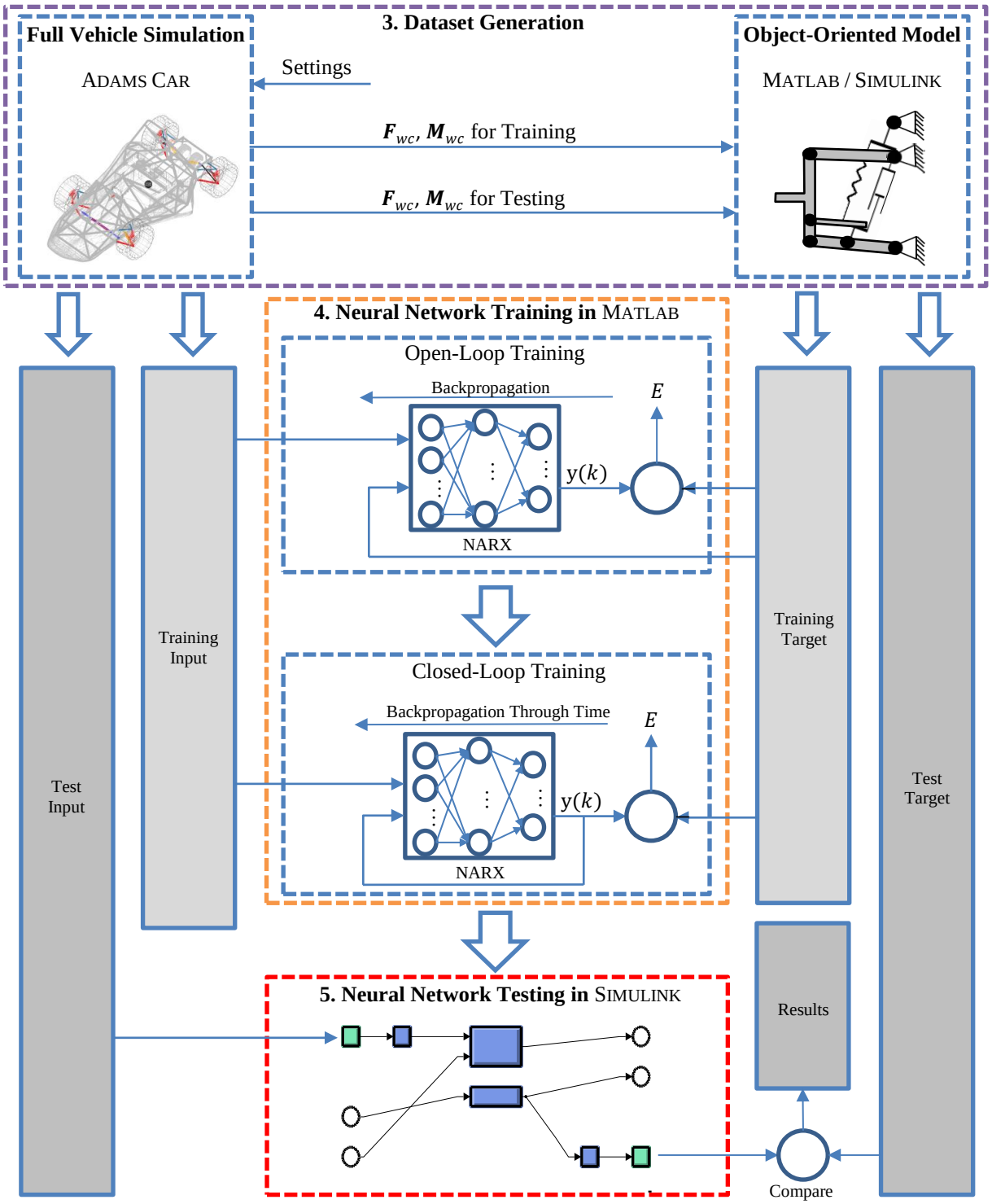


Figure 1.1 Workflow of the thesis. Source: Object-oriented model based on [12]. NARX representations based on [13].

Next, the contents of the following chapters are summarized.

In chapter 2, an overview of the two modeling techniques is given. In section 2.1, the basics of wheel suspensions and especially of the double wishbone wheel suspension are introduced. The kinematics of this architecture and the causes of the real-time-problematic DAEs in simulations are outlined, followed by a breakdown of multibody simulations used in automotive engineering. Section 2.2 covers the background of neural networks. The most basic building block of neural networks, the neuron model, and its usage in various NN architectures and learning algorithms is shown. Afterwards it becomes clear which NN layouts and training procedures are suitable for the given problem of non-linear time series modeling. The workflow of chapters 3 to 5 is displayed in Figure 1.1.

To generate data for the training and testing of the NN, chapter 3 presents the actions taken to extract time series from the multibody vehicle simulation software ADAMS CAR, and the subsequent use of the analytical object-oriented model implemented in MATLAB / SIMULINK, which the neural network shall replace. The two acquired datasets serve as input and target data to the network. The training dataset aims to train the NN by providing a widespread range of typical driving situations, bundled into two racetrack runs. Testing situations included in the training dataset and situations that are unseen by the NN, various driving events are simulated. This way, generalization (the ability to handle new data) can be tested.

Chapter 4 covers the implementation of the neural network model and it's training in MATLAB. First, the reasoning behind the selection of the *NARX (non-linear autoregressive exogenous)* network architecture is given. The computations inside the model are explained as well as its open- and closed-loop configurations. Then, the training procedure is followed: pre-processing prepares the data for input of the network before the training commences. The training phase is executed in two different steps, open- and closed-loop training that use different learning techniques. To improve results of the training phase, hyperparameters are optimized. Last, the post-processing is addressed.

In Chapter 5, results from training and testing are shown. After discussing the training performances, three networks are chosen for the embedding in the modeling environment SIMULINK. Here, the datasets collected from the driving maneuvers in chapter 3 are tested. Additionally, the topic of computational effort and the possibility of parallel computing is examined.

The acquired results are discussed in Chapter 6 in terms of modeling accuracy, generalization ability of the neural network and computation time.

Finally, the performed research is summarized in chapter 7 and an outlook for further research is given.

2 Theory

In this chapter, the theoretical basics are briefly introduced. After a short introduction to suspension systems in general, the double wishbone suspension and its kinematics are explained, followed by a summary of vehicle simulations that are used to model such suspensions and other multibody systems. Second, an overview of neural networks that are used particularly for the modeling of time series, e. g. the result of the aforementioned simulations, is given. The basic neuron model as well as its integration in various types of neural networks is explored. Finally, learning and measuring quality in neural networks is addressed.

2.1 Double Wishbone Suspension Modeling

In a wheel suspension, the wheel is mounted free to rotate on a wheel carrier. The wheel carrier is connected to the vehicle frame with typically one to five control arms. The hardpoints, which represent the interface of the suspension bodies to the frame, define the basics of every wheel suspension. On the front axle, steering input is fed to the suspension from the steering rack that is typically located in front or behind the center of the wheel on the axle subframe. Assuming no rear steering is used, the wheel on the front axle has an additional *degree of freedom* (DoF) regarding the steering motion while the spring deflection is the only DoF for the wheels on the rear. An optional anti-roll bar connects both suspensions on one axle to reduce rolling of the vehicle frame in corners. Rubber bushings are often used to minimize vibrations that are passed from the road to the interior. Wheel suspensions are built symmetrically to the longitudinal vehicle center line. It is therefore possible to examine a single wheel suspension, a quarter vehicle, and derive implications for the complete vehicle dynamics [22].

The *double wishbone suspension* assembly is one of the most common suspension topologies in automotive engineering, often used in upper class sedans, SUVs and sport cars. It belongs to the class of independent suspensions, in other words: the wheel can be controlled individual and independent of other suspension / wheel behavior [22]. It is a popular choice for formula-type racing cars [23].

In a double wishbone suspension, the wheel carrier is guided with laterally placed control arms. The upper control arm is linked to the upper end of the wheel carrier with a spherical joint. The lower control arm is mounted to the wheel carrier in the same way at the bottom end. Upper and lower control arms are connected to the chassis via revolute joints. The spring / damper element connects the lower control arm to the chassis and allows for the spring deflection (1st degree of freedom) [1]. The spring deflection is required for the wheel to follow the roughness of the road and to enable the isolation of vibration from the interior [22].

Linked to the wheel carrier with a spherical joint, the tie rod controls the steering of the wheel (2nd degree of freedom). It transfers the steering displacement from the steering rack into the

rotation of the wheel about the steering axis and is mounted to the steering rack with another spherical joint. Therefore, the tie rod is free to rotate, creating an isolated degree of freedom (3rd DoF) [1].

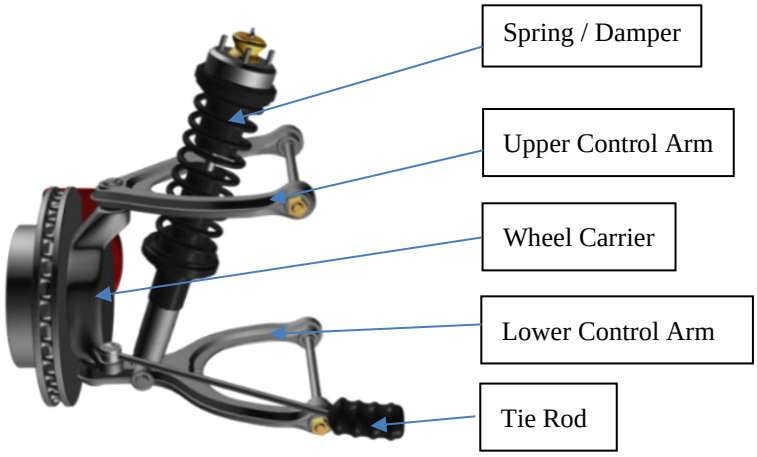


Figure 2.1 Components of a double wishbone suspension. Source: [5].

Advantages of this wheel suspension layout include the relatively free configurability of parameters like track width, toe- and camber angle as well as the suitability for powering. On the other hand, with its five bodies, it is a large and complex structure that requires a comprehensive elasto-kinematic set-up [1, 23].

2.1.1 Kinematics of a Double Wishbone Suspension

Kinematics is a part of technical dynamics. It is concerned with the description of movements of bodies. Essential for every mechanism and independent of its modeling technique is the computation of degrees of freedom (DoF). The number of DoF is defined by the constraints and therefore joints with other bodies. A rigid three-dimensional body in three dimensional space has six degrees of freedom which are constrained by the choice of joints they are connected with [24]. The state of the joints is usually characterized by generalized coordinates [22].

Examining wheel suspensions, the rotation of the wheel is left out of this consideration since it is part of every topology. Furthermore, the isolated rotation of the tie rod is neglected since it does not affect the movement of the wheel. The components of the wheel suspension therefore need to reduce the DoF of the system from six to one so that the wheel can only move vertically by the spring deflection, or to two DoF, if steering is possible [22].

All suspension types can be modeled by kinematic chains [22], which can be clustered into two topologies [1]:

- *Open topology:* Bodies are connected in a tree structure. Every body in this chain has only one preceding body.

- *Closed topology:* Kinematic loops exist that result in differential algebraic equations (DAE) that are difficult to solve or unsolvable. In some cases, special methods can be used to convert DAEs into ordinary differential equations (ODE) [5].

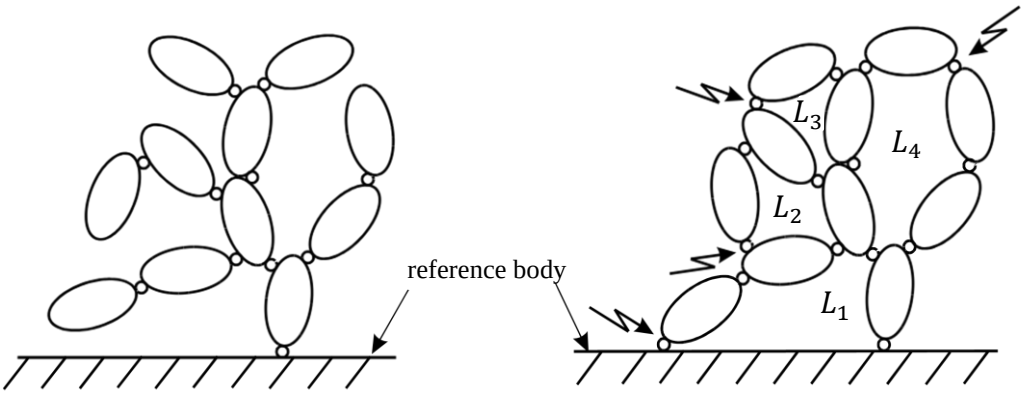


Figure 2.2 Open kinematic chain (left) vs. closed kinematic chain / kinematic loop (right). Source: [25].

From now on, a front suspension is considered, where a steering input is necessary. As outlined above, the double wishbone suspension consists of six bodies (no anti-roll bar). These are linked by 2 revolute, 1 prismatic, and 4 spherical joints that imply 3 DoF [5]:

$$f_G = \sum_{i=1}^{p} e_i - q = \underbrace{4 \cdot (6)}_{\text{4 bodies}} - \underbrace{3 \cdot (6-3)}_{\text{3 spherical j.}} - \underbrace{2 \cdot (6-1)}_{\text{2 revolute j.}} - \underbrace{1 \cdot (6-4)}_{\text{1 spherical prismatic j.}} = \underbrace{1}_{\text{spring deflection}} + \underbrace{1}_{\text{steering motion}} + \underbrace{1}_{\text{isolated DoF}} \quad (2\text{-}1)$$

In equation (2-1) e_i accounts to the DoF of all p bodies and q to the independent joints and mounts to other bodies and mounts to the reference body. Those three DoF are made up by the lifting of the wheel carrier (deflection of the spring / damper), the steering input and the free rotation of the tie rod [5].

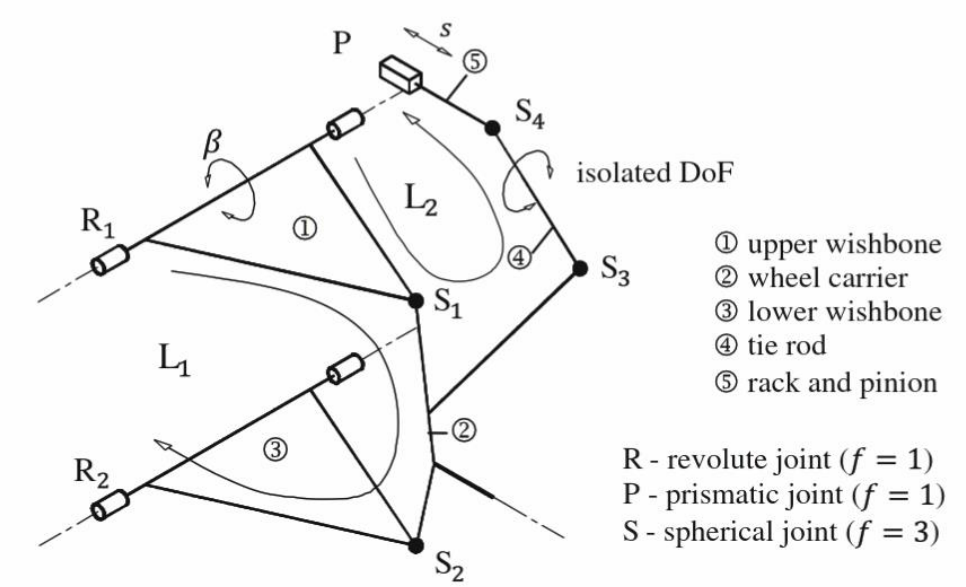


Figure 2.3 Kinematic structure of a double wishbone suspension. Source: [1].

Known as the topological method, the kinematic system is analyzed with regards to its kinematic loops. One can note from Figure 2.3 that two independent kinematic loops are connected via revolute joint R_1 and spherical joint at the upper control arm S_1. Loop 1 consists of the wheel carrier and the upper and lower control arm while loop 2 also contains the upper control arm, the wheel carrier and additionally, the tie rod and the steering rack. In terms of joints, the loops are related via the spherical joint S_1 which implies that the steering input s is independent of spring deflection.

2.1.2 Time Series as Result of Vehicle Simulations

For the development and configuration of vehicle dynamics, *multibody simulations* (MBS) are often used. These software systems can also be applied to general mechanical problems outside of automotive engineering. Often included, part libraries and toolsets for specific applications help to reduce the development costs and time. In terms of graphic user interfaces (GUI), a pre-processor helps to create the model. Post-processors support the user in evaluating results. Besides general multibody simulation tools, special software systems that are dedicated to vehicle dynamics are also commercially available, using integrated multibody simulations. In addition to the MBS functionalities, they allow the user to work with extra features, such as driver models or road profiles [1].

It exists a wide range of benefits of using simulations in the field of vehicle development: tests are reproducible and comparable because the test environment remains unchanged in each test iteration. In some scenarios, simulations can replace costly prototypes or make testing in earlier development phases possible when there is no prototype available. Critical driving maneuvers can be conducted safely [1].

MBS are usually used to examine movement of complex multibody systems, which are made up of moving bodies, joints that link the bodies, springs / dampers, bushings, etc. In terms of double wishbone suspensions, externally introduced loads, called *applied forces and torques* can be seen as the input to the MBS system which can be described by a set of differential-algebraic equations (DAE). Calculation of these DAEs requires immense computational effort, constraining the real-time capability of such simulations. Reaction forces and torques are processed internally but can also be tracked and exported from the software. ADAMS CAR is frequently used for the development of wheel suspensions since it houses a database of templates for different wheel suspension layouts, such as the double wishbone wheel suspension. Nowadays, MBS are also able to cover elasticities of bodies. During highly dynamical driving maneuvers, elasticities can have a significant impact on the vehicle dynamics when not considered. For example, a suspension linked to a perfectly rigid vehicle body causes higher reaction force peaks than would normally occur in real-world driving scenarios with flexible bodies. Another factor to consider are production tolerances that are often not accommodated in MBS. MBS become especially powerful tools for development, if combined with FEM (finite elements method), material strength-, stiffness-, durability or NVH (noise, vibration, harshness) analysis [22].

A full vehicle simulation is beneficial for development of components and complete vehicle because dynamical behavior of every component can be easily examined. Arbitrary maneuvers on virtual test track are possible. If focused on the suspension dynamics, a good understanding of the tire models is necessary since it heavily influences the applied forces at the wheel center. Load-time data is usually key to evaluate in most full vehicle suspension simulations along with a power spectral density analysis that focuses on the lower frequencies up to 30 Hz (typical resonance frequencies of suspension systems) [22].

Although forces and other variables that play a role in simulations are continuous, they must be measured at chosen timesteps. This process is called sampling and the vector of values at a given timestep is called *sample* [26]. Exported from the simulation software, the numeric simulation results can essentially be thought of as a sequence of vectors dependent on time, called a *time series* [27].

2.2 Neural Networks for Time Series Modeling

Artificial neural networks or simply *neural networks* (NNs) consist of (non-)linear computational elements that are arranged in parallel and try to mimic biological neural connections. In fact, in the early years of neural network research, scientists were studying the brain and its parallel computing ability to develop brain-like mathematical models that reproduce those processing abilities [3]. However, today's NNs that are used in research and commercial applications are not developed with those biological ideals in mind anymore [28].

Neural networks have become attractive areas of research and have been applied to many different topics. For example, pattern recognition, signal processing or data analysis strongly profited from the rise of neural networks [4].

To understand the internal processing of a neural network, one must look at an individual element of a network first: the neuron, also called node, which will be further discussed in the following section. The mathematical concept is rather simple. Classical neural networks only consist of multiple neurons that are essentially summation devices which individually put out a single signal.

A clear drawback to neural networks is its black box representation, meaning that the network itself is opaque. Basically, it hides the rule that it was trained on [4]. In a classical neural network, rules cannot be extracted from the network's internal parameters without much effort. On the other hand, black box modeling implies that no expert knowledge about the input to output relationship is needed beforehand (*a priori*) [27, 28]. Because of the simplicity of their concept, neural networks are easy to use while being relatively fault tolerant, as being able to handle new types of input [4], just like the human brain [29].

Neural networks can be applied to a wide variety of problems. Most of them can be categorized into classification or function approximation / regression problems. Classification machine learning, in which the output is to be associated with a certain category, uses many different methods that cannot be applied to function approximation and vice versa. Besides system modeling, function approximation contains a large set of objectives such as system identification, prediction, control, signal processing and many more [29], of which time series forecasting is the most popular [26].

In neural network system modeling, the basic procedure can be separated into two steps. First, acquire a model of the underlying physical system. After training the neural network to approximate the system, the task of applying new data to it has become a simulation problem. This is the second step. A system response (output) is generated by the mathematical model (neural network) that is given an input [4].

2.2.1 The Neuron Model

Neural networks are generally made up out of *nodes* (also called neurons) that use *weights* that define their connection strength. Neurons are essentially building blocks of neural networks that are simple computational devices themself. The connections between the nodes make computation in a network possible [30]. The value of weights is changed during the training process to increase the performance of the network. The most basic node (see Figure 2.4) takes N inputs with N constant weights to calculate the dot-product and pass it through a (non)-linear activation function that is static and typically monotone and continuous [28].

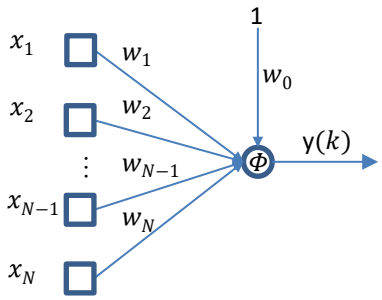


Figure 2.4 Neuron model. Source: Based on [28].

The activation function Φ is taking the input vector

$$\boldsymbol{x} = [x_1, \dots, x_N, 1]^T \quad (2\text{-}2)$$

can be the result of neurons from a previous layer or simply the input to the neural network and the weight vector

$$\boldsymbol{w} = [w_1, \dots, w_N, w_0]^T \quad (2\text{-}3)$$

to calculate the output of the neuron y. The summation of inputs x_i multiplied by a weight is also called activation potential [28]. An individual weight w_i essentially contributes to the influence of an input towards the output. The higher the weight, the stronger the output is dependent on this specific input. The last element w_0 corresponds to the constant *bias* input of the neuron. It can be thought of as a unity input that is multiplied by the bias weight w_0 (see Figure 2.4). The inputs to the so-called *activation function*

$$y = \Phi\left(\sum_{i=1}^{N} w_i x_i + w_0\right) \quad (2\text{-}4)$$

are displayed in equation (2-4). The activation function is chosen dependent on the given task. A selection of activation functions for non-linear system modeling is discussed in section 4.4.3.

2.2.2 Neural Network Architectures for Time Series

Multiple nodes at the same hierarchical level that are parallel aligned are called *layers* [28]. Neurons in a layer are usually connected to all neurons in the adjacent layers but not in their own layer.

Feed forward and recurrent networks are the most used NN categories for non-linear signal processing and data fitting [4]. The major differences shall be briefly addressed:

- In a common *feed forward neural network*, neurons are connected to the previous and the next layer. The input signals are processed layer by layer, propagating them in a forward direction from the input layer until they reach the output layer [28]. The simplest

architecture of a neural network is a perceptron, consisting of only a single node with a threshold activation function. Connecting perceptrons forms multilayer perceptrons which can solve classification problems of any complexity [30].

- *Recurrent Neural Networks (RNNs)* use the same method of forwarding the signal through the network but feed the output back into the network together with delayed versions of the output. The signal can be fed into the hidden layer, into the output neurons themselves or introduced back into the model as input neurons [28] like in the NARMAX and NARX approaches [4] that are discussed in the next section and further in section 4.1. In a classical RNN architecture, the feedback connects back to the hidden neurons directly [31]. Recurrent delayed connections help introducing memory into the network [28, 32]. In the case of time series applications, RNN have shown to be superior to feed-forward networks, due to their ability to model time series including time-dependencies with lower errors in a more parsimonious way [33].

2.2.3 Recurrent Neural Networks

Problems such as pattern recognition and classification are static problems in machine learning. Dynamic problems include time series modeling and prediction, system identification and other problems where past values are required to determine the output of the current computation. Since there are no dynamics inside of a standard feedforward net, the dynamics must be introduced in some other fashion, for example by adding tapped delay lines for lagged input and output [4].

Recurrent neural networks are well suited to model non-linear dynamic systems in time series. Given enough neurons in a network, some RNNs can model those systems with arbitrary accuracy [34].

BOX ET AL. first introduced AR (*autoregressive*), MA (*moving average*) and ARMA (*autoregressive moving average*) models in the 1970s [35]. “Autoregressive” refers to the idea that the model estimates it output dependent on past sequences. “Moving average” models compute their output from the linear combination of random time series [26]. Developed without the idea of neural networks and its today’s applications in mind, these models cannot handle non-linear inputs but have been useful when applied to problems with unknown solution but with enough data available [26, 27]. Newer models which are adapted to non-linear inputs are called NARMA. In general, recurrent neural networks can be seen as a network representation of these models [28] and are more powerful since they can handle changing means and variances in the dataset and can model more complex time series. On the other hand, they are more likely to face challenges like overfitting, local minima and the requirement for larger datasets [26].

To reliably model complex dynamical systems, such as complex multibody mechanisms, a feedback is required. This feedback can be introduced by using a recurrent neural network architecture for modeling. Additionally, recurrent connections provide the network with the ability to embed a memory and thereby learn the underlying system dynamics. A typical recurrent network can be summarized as a function

$$y = F(y_{PAST}, u) \quad (2\text{-}5)$$

that is an approximation of the function y with the external input u and the previous approximations of this function y_{past} also given as input [28, 32].

As it is now clear that a recurrent neural network fits the given task, a closer look shall be given to few of the most used layouts:

Fully Connected Recurrent Networks describe networks, in which every neuron is connected to every other neuron. Recurrent connections are established only back to the same hidden neuron. These networks are universal approximators of dynamical systems, meaning they are able to approximate any underlying dynamical behavior (discrete- or continuous-time) with an arbitrary accuracy if the network size is large enough [29].

Long Short-Term Memory Networks (LSTM) are well-suited for learning long term dependencies in time series. For time series outputs, the complete network consists of an LSTM and a fully connected network operated in sequence [34]. Their architecture is more complex, adding loops whose weights are controlled by other hidden elements and gates that help to accumulate or forget certain parts of memory over a long duration. LSTM networks have found popularity especially as building blocks of deep learning architectures [36].

Non-linear autoregressive models with moving average and exogenous input (NARMAX) belong to the subclass of single-layer, time delay networks. “Exogenous” refers to the external input [29]. The network itself has a simple architecture, consisting of activation functions with one input-, one hidden- and one output layer. It uses lagged inputs – in addition to equation (2-5) – and lagged outputs that are fed back into the model [4]. The NARMAX approach, introduced by BILLINGS, has been successfully applied to a wide range of non-linear system problems including the identification of hysteresis effects in metal rubber damping devices, dynamic modeling of synthetic bio parts, forecasting high tides in Venice, to characterize the behavior of mobile robots [4], in field of biomechanical research [37], or to control thermal management in smart buildings [38].

2.2.4 Learning Algorithms

Learning rules are fundamental for the success of neural networks. A *learning algorithm* is basically an optimization procedure for finding the most suitable parameters that reduce a certain cost function [29]. During the *training phase*, a learning algorithm changes the weights

that determine the strength of the connection between the neurons. Learning how to emulate the given behavior in the data set, the weights converge towards a (local) optimal configuration [4]. Summarized, the problem of learning in terms of neural networks becomes optimizing weights and biases to compute the desired output [3].

The most common used cost function for neural networks

$$MSE = \frac{1}{K}\sum_{k=1}^{K} E(k)^2 = \frac{1}{K}\sum_{k=1}^{K} (T(k) - y(k))^2 \qquad (2\text{-}6)$$

is called *mean squared error* (MSE) [34]. The squared errors of all timesteps k are summed and then divided by the total number of timesteps K. An advantage of this measure of quality is the possibility to compare performances between different sizes of datasets of the same problem. It should be thought of like an average, rather than a sum [36].

The most basic learning algorithm for neural networks is standard *backpropagation* (BP). It was widely used in the early years of machine learning [39]. The BP algorithm is a way to compute the gradient with respect to the model's parameters. Gradient descent methods determine how to adjust the weights to approach the minimum of the cost function [36]. Modern applications of neural networks use the LEVENBERG-MARQUARDT Algorithm [40] as a variant of BP instead of standard BP. It is usually 10 to 100 times faster than the normal gradient descent BP method. It performs especially well on non-linear regression tasks than on pattern recognition problems [34].

The LEVENBERG-MARQUARDT Backpropagation Algorithm (LMBP) is described in detail by HAGAN ET AL. It is especially useful when the performance function consists of sums of squares, like in MSE. LMBP is the fastest algorithm for MSE minimization for up to few hundred weights. Beyond that size, other learning algorithms could be viable due to the high storage capacity in LMBP [34]. A short overview of the algorithm by HAGAN ET AL. [30]:

1. Run all inputs through the network to compute the outputs and the corresponding errors to each output as well as the squared errors as in equation (2-6).

2. Compute the JACOBIAN matrix, in which the partial derivatives of the errors with respect to the change in weights and biases are represented. Essentially, this matrix contains the transposed gradients. First, calculate the *Marquardt-Sensitivities* by backpropagating through the network, starting at the last layer of the network, using the derivatives of the activation functions. Update the JACOBIAN matrix with the product of MARQUARDT-Sensitivities and the input to the given node.

3. Update the weights by subtracting the product of sensitivity, learning rate and input vector and the biases by subtracting the product of sensitivity and learning rate. The

learning rate is first set to a small value, e. g. 0.01. It is increased when the cost function does not decrease at first and decreased when the cost function decreased at the last step. If the cost function does not increase with a small step size, it is an indicator that the algorithm could be stuck at a local minimum. In local minima, the cost function is lower than in all of its neighboring points, so the algorithm has to make a bigger step in the direction of the steepest descent to find a lower value of the cost function [36].

4. Compute the norm of the gradient. The learning is usually stopped when this value drops below a predefined value. Otherwise, repeat steps 1-4.

The LMPB can be implemented as described above for static neural networks like a feedforward architecture. For recurrent networks, a type of dynamic neural networks, the gradient has to be computed in a more complex way since a backpropagation through the network is not possible due to the recurrent connections [34]. When faced with the challenge of calculating the input to the first hidden layer (see step 2.), the input to the non-linear function is not directly available, since it requires the delayed feedback from the previous timesteps. Instead, the algorithm must compute the output of all the previous timesteps first in a recursive procedure, sometimes called unrolling [41]. When the first timestep is reached, the initial delay states are assumed to be zero. This is called backpropagation through time (BPTT) [39].

For NARX networks in an open configuration, the LMBP algorithm can be used as described above for the learning phase since the structure resembles a single layer feedforward network with additional input. When a NARX network is trained in the closed configuration, a dynamic back-propagation algorithm like BPTT is necessary to calculate gradients. This is more computationally complex than for static networks [27] and more likely to get stuck in local minima. Therefore, the network should be retrained multiple times with new, randomly initialized weights and biases [28].

2.2.5 Generalization: Underfitting and Overfitting

The reason for training machine learning system is usually the future application to new, unseen data that was not part of the training dataset. The performance on new data is referred to as *generalization.* The performance on the training set can easily be measured by the MSE (see above). However, quantizing the error on new data is based on prediction, which is based on the performance computed with an additional test dataset, separate from the training set [36].

Generalization plays a major role in the model selection. As DU AND SWAMY point out, the key is to find a model that fulfills the requirement of providing sufficient accuracy during the training but should be simple enough, so that good generalization performance can be achieved [4, 29].

A poor model or training that does not yield a good error performance is *underfitted* for the given training [29]. On the other hand, focusing on the reduction of the training error alone will lead to *overfitting* [36]. The model performs well on the training data, but might also have been trained on the noise, rather than the underlying function. A bad test performance is the result [29]. An example is given in Figure 2.5. Samples depicted by a red '+' are found in the training dataset. The output function of the network estimates the datapoints from the training set perfectly but fails to learn the smoother underlying relation.

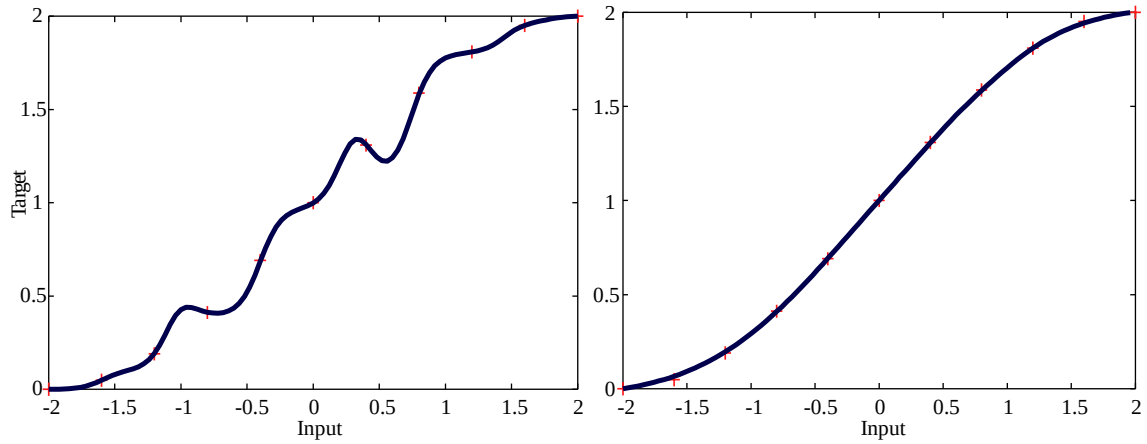


Figure 2.5 Example of overfitted data (left) and well-fitted data (right) in regression learning. Source: generated based on [30].

The cause for this might be found in the choice of too many parameters to train on (e. g. size or number of hidden layers, complexity of model, etc.), or the overtraining of the network. For the latter, early stopping methods can be applied (see section 4.3.1.2), to stop training when the model generalizes best [29]. One of the most common ways to ensure good generalization performances uses a constraint term added to the error function that penalizes high weights. A performance ratio then determines the balance between the two goals: error reduction and generalization. This method achieves generalization by so-called *regularization* [29]. To include the additional error term, the LEVENBERG-MARQUARDT Algorithm (see section 2.2.4) can be adjusted to optimize not only the MSE but also another term

$$MSW = \frac{1}{n}\sum_{i=1}^{n} w_j^2 \qquad (2\text{-}7)$$

called mean square weights (MSW). This penalizes the network for increasing the weights too much and thus overfitting to the training data. The new cost function

$$MSEreg = \xi MSE + (1-\xi)MSW \qquad (2\text{-}8)$$

includes both MSE and MSW. The balance between both terms is held by the performance ratio ξ (see equation (2-8)) [27], which is adjusted automatically [29].

3 Dataset Generation

In this chapter, the approach to obtaining datasets for the training and testing of the neural network is outlined.

A dataset in general consists of a multitude of samples. In the context of time series, samples are called timesteps. For the generation of a time series, certain attributes (also called features) are measured at every timestep to build up information contained in the dataset [36]. In this context, the attributes represent physical variables of a mechanical system like *forces* $\boldsymbol{F}$, *torques* $\boldsymbol{M}$, *accelerations* $\boldsymbol{a}$, *velocities* $\boldsymbol{v}$, *positions* $\boldsymbol{r}$ and *generalized coordinates* $\boldsymbol{x}$.

Because of the upper frequency bound of 30 Hz, the step size that would ensure functionality of the numeric approximators used in the object-oriented model must not exceed $h_{max} = 1/60\ \mathrm{s} = 0{,}017\ \mathrm{s}$. However, to simplify future integration into a full vehicle model that uses higher frequencies, the frequency of this model is set to $1000/\mathrm{s}$, implying that a sample is generated every 1 ms.

The signal flow of the neural network model takes the same variables as input and selected output from the object oriented model described by KRACHT [2, 5] (see section 3.2.2).

Consider a quarter vehicle. The tire serves as contact to the street but is not part of the suspension model. Forces and torques are applied to the wheel carrier by the tire which basically acts as an air spring [1]. These are the inputs to the neural network $\boldsymbol{F}_{wc}$, $\boldsymbol{M}_{wc}$. In section 4.1, the procedure of obtaining the input data from the full vehicle simulation in ADAMS CAR is explained.

The output $\boldsymbol{y}$ of the NN might consist of a wide variety of variables, depending on which the network has been trained on. For the supervised learning of the network, target (or teaching) data is required. This can be attained by training the network on a desired set of outputs. The data is generated by the object-oriented model of the suspension, implemented in SIMULINK. It is further discussed in section 3.2.2.

It is difficult to determine whether enough data has been collected for the given problem since it depends on the complexity of the underlying function that describes the system. For most NN applications, the underlying function is not known, the process must be iterative – if the accuracy of the neural network is sufficient, enough data was used [30]. In this case the input-output relation is known but cannot easily be described. However, data can be generation with the help of the vehicle simulation in arbitrary dataset sizes.

3.1 Input Data

The Data generated by the processes described in this section will serve as input data to the neural network model as shown in Figure 3.1. Respective of the purpose of the dataset (training or testing), different settings in ADAMS CAR are made to accommodate for proper NN training and testing. The forces and torques applied to the center of the wheel $\boldsymbol{F}_{wc}$ and $\boldsymbol{M}_{wc}$ function as interface between the two environments.

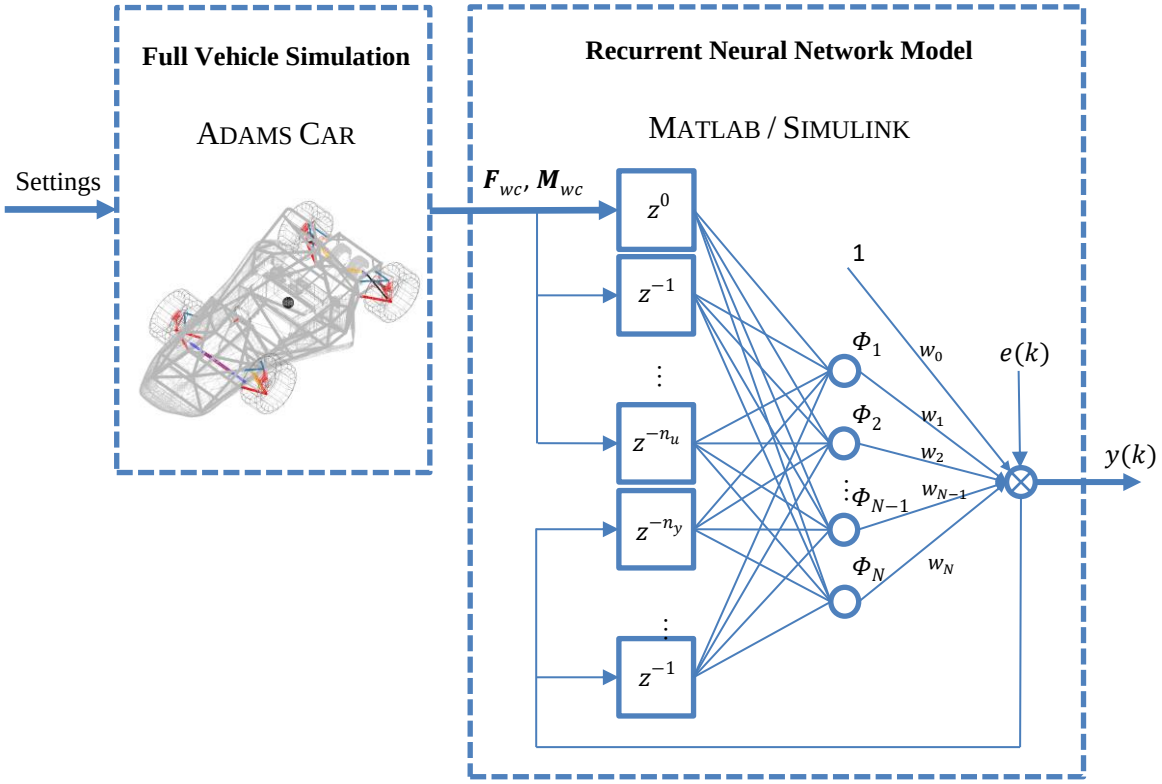


Figure 3.1 Simplified process for input data generation. Source: Network model representation based on [4].

In an individual suspension of the front left quarter vehicle, the forces and torques that are applied to the wheel carrier can be divided each into three vector components in the reference frame of the wheel carrier / tire as displayed in figure

- x $-$*axis:* When rolling, the wheel moves along the x $-$direction, also called longitudinal. The torque about this axis is called overturning moment M_x.

- y $-$*axis:* Points into the lateral direction. For the left suspension, the positive y-direction faces left. The torque about this axis is called rolling resistance M_y which is also influenced by braking. Throttling has no direct influence in this case because the front axis is not driven.

- z −*axis:* Normal forces point upwards. The torque about this axis is called aligning torque M_z, caused by lateral stress on the tire patch [1].

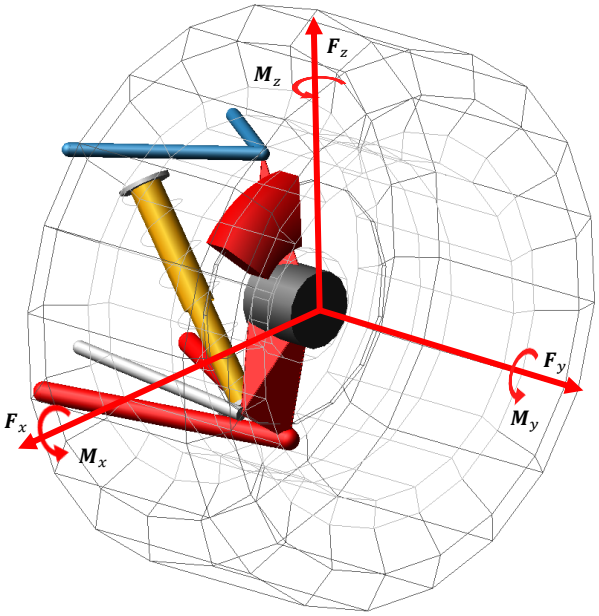


Figure 3.2 Reference frame of the wheel hub (front left)

To model the input-output relation of the underlying process, any machine learning algorithm requires data that is used for the learning process. The generation of the training dataset is described in section 4.1.1. In sections 4.1.2 and 4.1.3 some additions are made to the process of the training data generation, that add extra difficulty like road profiles to the training data but make it more realistic. In sections 4.1.3 to 4.1.7, tests are proposed that are utilized to examine the fully trained network's capability to handle new data. The test maneuvers are divided into acceleration, braking, turning and *ISO Double Lane Change* test.

The training data consists of a time series from a full racetrack simulation. The test time series are short maneuvers that each examine a different driving situation. These tests are partly oriented towards the situations found in the training dataset, but also explore different driving situations.

Next, the race car A40-02 of the FORMULA STUDENT E-TEAM DUISBURG-ESSEN E.V. is introduced. Its total weight is 300 kg and is powered by 2 electric motors (100 kW max.). The battery holds 6 kWh and a planetary gear is used for torque vectoring. The front track width measures 1.2 m and the rear track width 1.18 m. The wheelbase is 1.575 m long [5].

The front suspension of the A40-02 is carried out as a double wishbone suspension which consists of four rigid components plus a spring / damper element. An anti-roll bar is not built in. The length of the control arms can be altered (see chapter 2.1 for the structure and kinematics

of a double wishbone suspension). The rear uses a double wishbone suspension but in addition a pushrod and a rocker are located between the lower control arm and the spring / damper [5].

A rigid model of the formula student racecar has been initially created at the CHAIR OF MECHATRONICS at the UNIVERSITY DUISBURG-ESSEN [6, 7] and later validated by [8]. Parameter were tuned, so that differences between the CAD (Computer Aided Design) model and the real parts (e.g. manufacturing tolerances) are now included in the ADAMS CAR model [5].

3.1.1 Vehicle Simulations in Adams Car

To generate input data for the neural network model, an existing ADAMS CAR model of the A40-02 race car is used. ADAMS CAR is a multibody simulation software specifically designed for vehicle simulations. However, the source code of this commercial software and therefore the underlying methods of calculation are only partly accessible. Thus, some features and mechanism cannot be discussed in full detail.

In this ADAMS CAR model, the complete vehicle assembly consists of at least six subsystem assemblies: front suspension, rear suspension, steering, front tires, rear tires and chassis. Templates are used as the base for these subsystems, defining the geometry of each used part. To work on these templates, ADAMS CAR must be opened in expert mode. In standard mode, simulations can be run, evaluated and generated data can be exported. Furthermore, simulation results can be plotted in a post-processor and visualized by animation controls. The pre-defined simulations can be split up into two categories: subsystem simulations and full vehicle simulations [42]:

- *Subsystem simulations* are used to design or validate subsystems, e. g. to adjust front suspension parameters to optimize the course of the camber angle while simulating parallel wheel travel.

- *Full-vehicle simulations* require a full vehicle assembly, consisting of all subsystems. Their goal is to analyze the effect of changes to the vehicle dynamics.

For this research only full-vehicle simulations are used in ADAMS CAR, because the dynamic behavior of the suspension during realistic events is of interest. In the simulation menu, full-vehicle simulations are further categorized into straight-line, open-loop steering and closed-loop steering events [42]:

- *Straight-line events:* include braking, acceleration or constant throttle events in which no significant steering input is simulated. The steering can be chosen to be either locked, free or tuned to keep the vehicle on a straight line. A locked steering has proven to stabilize the vehicle best although it deviates a bit from staying on a perfectly straight line.

- *Open-loop steering events*: events in which the steering input is a function of time. The course of the event has no impact on the steering input.

- *Closed-loop steering events:* a controller is used to calculate the steering input. For example, when simulating a double lane change, a PID controller is used to keep the vehicle on the ISO-defined course (Course Event, described in detail in section 3.1.7) [43].

Moreover, ADAMS CAR requires the input of a gear position for most maneuvers. In case of the A40-02, only one drive gear exists and is therefore chosen in all the following simulations.

All the simulations used to generate data for the training and testing of the neural network model are explained in detail in the following sections 3.1.2 to 3.1.7.

3.1.2 Training Data - Road Smooth Track

To ensure that the fully trained model can simulate the suspension behavior during all driving situations, a complete track with multiple, different turns and differently sized straight sections is chosen. The vehicle in the simulation must steer around corners, brake before and accelerate after turning. For different radii of corners, different yaw angles can be observed that might have different impacts on the suspension. Additionally, for the different straight sections of the course, varying acceleration and braking behavior is expected and the dynamic suspension behavior can be observed during high velocity and low velocity sections. To tune the driving behavior, the function ADAMS CAR *Smart Driver* is used.

The pre-defined road smooth track in ADAMS CAR consists of 1561 road points. The attributes *X-coordinate, Y-coordinate, Z-coordinate, width, bank, friction left* and *friction right* are linked to every road point. The $x-$ and $y-$coordinates can be plotted and are displayed in Figure 3.3. The road points can be altered in the Road Builder, a function of ADAMS CAR.

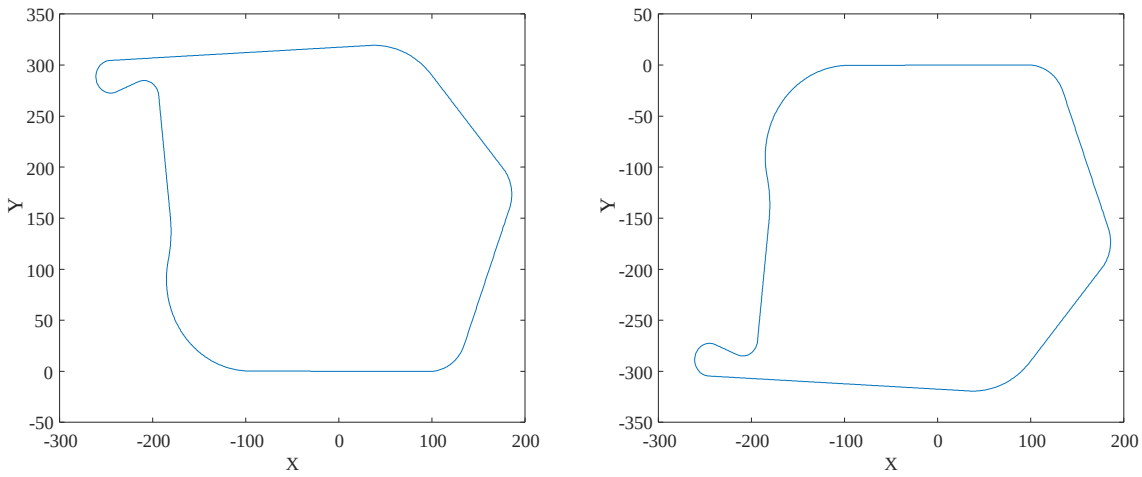


Figure 3.3 Map of „Road Smooth Track“ (left); Mirrored „Road Smooth Track“ (right)

The starting point for the vehicle is located at the coordinates $(0|0)$. Starting from the origin in a counterclockwise direction, the vehicle must steer around multiple corners with different radii. To allow the vehicle to stay on the track and not abort the simulation, the road width is extended to 12 m for all road points. All other attributes remain unchanged. The standard configuration has no elevation, therefore $z-$coordinate equals 0 for all road points. Furthermore, the road bank value is set to 0 and the friction parameters for left and right are set to 0.9 by default.

As can be seen in the map of the racetrack (see left side of Figure 3.3), the racetrack mostly consists of left turns. To prevent the neural network from training asymmetrically on turning left, the racetrack is mirrored and added to the training dataset, achieving a balance in left and right turns. To mirror the racetrack, all $y-$values of the original racetrack are multiplied with the factor -1 (see right side of Figure 3.3). Unaffected of the mirroring, the starting point for the vehicle is located at the coordinates $(0|0)$ but the moving direction changes to clockwise.

Initial trials to connect the original and the mirrored racetrack to a single loop failed because the generated results file seemed too big for ADAMS CAR to handle (> 10 GB).

The user has the option in ADAMS CAR to perform a full vehicle analysis by using the Smart Driver option. The vehicle will drive along a pre-defined course while the steering, throttle and braking input is controlled by the Smart Driver model. Unfortunately, no more information about the internal computing of the smart driver is available.

To perform a Smart Driver analysis, a road data file and a course data file must be provided as well as and end time and a starting velocity. For the Smart Driver Task there are two options, the user can either define driving, braking and cornering parameters or let ADAMS CAR adjust them according to the vehicle limits [42].

Using the Road Smooth Track, the Smart Driver Task *vehicle limits* caused the simulation to halt immediately after the start. Therefore, driving, braking and cornering parameters are adjusted manually. Since a minimal lap time would be desirable during a real race event, all parameters should be chosen as high as possible. Increasing every parameter until the vehicle could not stay on the track leads to the following values: *Driving Acceleration:* 10 %, *Braking Acceleration*: 50 %, *Cornering Left*: 10 % and *Cornering Right*: 10 %.

The observed values for steering wheel angle (see Figure 3.4), longitudinal chassis acceleration and velocity (see Figure 3.5) are serving as a guideline for configuring the tests. The tests will be discussed in detail in the subsequent sections.

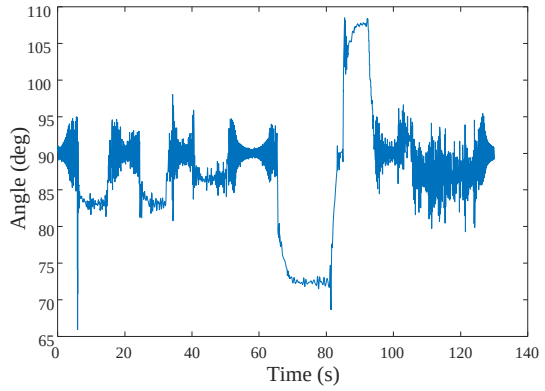


Figure 3.4 Steering wheel angle - "Road Smooth Track"

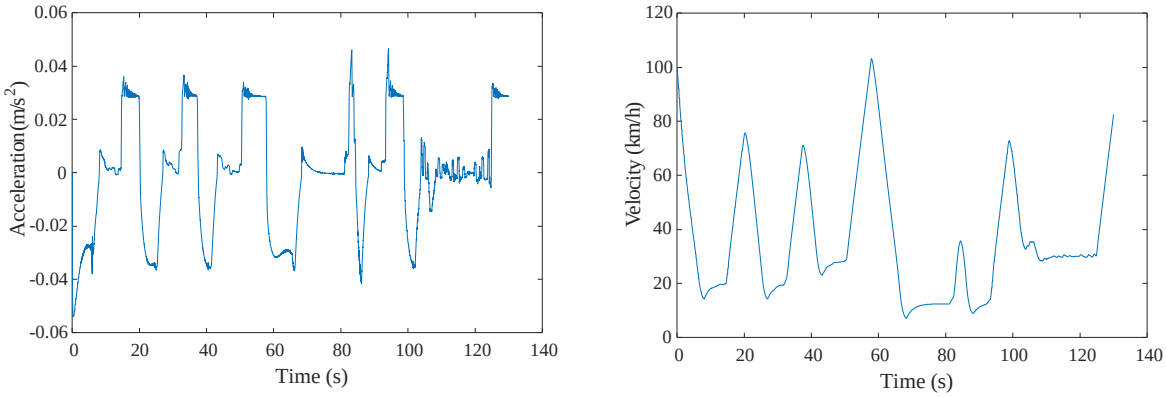


Figure 3.5 Longitudinal chassis acceleration (left); Longitudinal chassis velocity (right) – "Road Smooth Track"

This racetrack configuration is based on the road smooth track (see above). To add complexity to the racetrack and to train the network on the vehicle's behavior during ramps, one up-ramp and one down-ramp are added to the track on the long straight section before the narrow turns.

The Z −values were calculated by the help of the MATLAB expressions

```
round(-0.5*sin(pi/2:0.05:3*pi/2),3)'+0.5
```

```
round(0.5*sin(pi/2:0.05:3*pi/2),3)'+0.5
```

for the up-ramp and for the down-ramp. This puts out 2 x 63 values in the shape of a π-long sine curve section that are inserted in the road smooth track file. Considering the gradient is largest at the middle of this section and the standard distance between two road points is 1 m, the maximum road slope equals approximately 2.5 %.

3.1.3 Road Profile

When driving on a road, the unevenness of the road's surface influences the road-tire contact and the emerging forces. Statistical methods are used to measure the roughness of a given road

section. To reverse engineer the roughness into a model of an uneven road, power spectral density is used as the most important indicator among other parameters.

Statistical data about the roughness of the road makes it possible to categorize roads into classes A-H, with A having the smoothest and H having the roughest surface [44]. According to [45], only classes A and B can be seen as representative of road profiles from highways or similar functional road categories. Various country's highways show a road class of A. On another note, the recommended maximum speed on roads corresponding to classes below B falls under $100\ \mathrm{km/h}$. As the maximum speed during the racetrack simulation reaches more than $100\ \mathrm{km/h}$, road classes below B cannot be considered for this roughness simulation. Therefore, to simulate the roughness of roads on a racetrack, which should have the characteristics of a well-built highway, a road profile of class A will be assumed.

Table 3-1 Spatial frequency for road classes in ISO 8606:2016-11. Source: Excerpt from [44].

	G_d in 10^{-6} m^3		
Road Class	**Lower Limit**	**Geometric Mean**	**Upper Limit**
A	—	16	32
B	32	64	128
C	128	256	512

The ADAMS CAR RIDE Road Profile Generation Tool calculates the displacement PSD [46, 47]

$$G_d(v) = G_e + \frac{G_s}{(2\pi v)^2} + \frac{G_a}{(2\pi v)^4}, \qquad (3\text{-}1)$$

taking the inputs G_e (white-noise elevation), G_s (white-noise slope, integrated once with respect to time) and G_a (white-noise acceleration, integrated twice with respect to time), in which white noise is a sequence of random numbers with normal distribution.

[47, 48] argue that the value for G_s describes the range of roughness while G_a and G_e only correspond to very long and very short wavelengths respectively. The reason for non-zero G_e values could be local faults on the road while non-zero G_a values could indicate construction errors of the road. For this reason, only G_s is set to a non-zero value.

Assuming a common wavenumber above $0.2\ \mathrm{cycle/m}$ and below $1\ \mathrm{cycle/m}$, and a G_s value suggested by [47] for a moderately rough road of $20 * 10^{-6}\ \mathrm{m/cycle}$, G_d lies well within the road class A of the ISO [44] categorization (compare Table 3-1). With this input, ADAMS CAR generates a channel of road profile for each of both tracks [46].

The road profile is added to the racetrack simulation in order to train the neural network on the suspension's dynamic behavior while driving on uneven roads. It is used by the lateral controller and the tire subsystem to compute the contact forces between road and tire [46].

Road Profile Event

To test the neural network on predicting suspension variables on rough roads, a simple test is introduced. ADAMS CAR simulates the vehicle driving in a straight line (locked steering) on an uneven ground for 20 s. For this test, the same road profile configuration is used as in the racetrack simulation. A constant velocity of 70 km/h is maintained throughout the test.

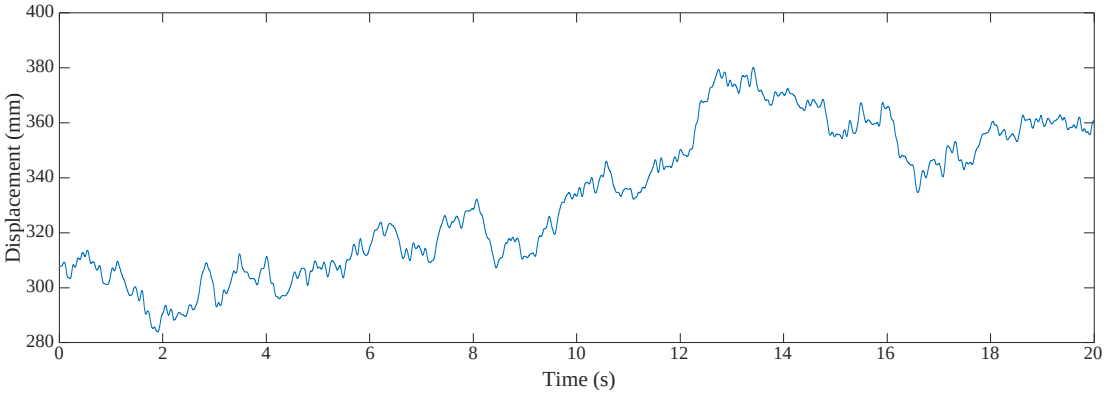


Figure 3.6 Vertical chassis displacement wrt to starting point - Road Profile Event

3.1.4 Acceleration Test

An acceleration maneuver was chosen to test the neural networks ability to perform predictions on the dynamic behavior of the suspension during acceleration of the vehicle without any steering input. All the following maneuvers, including the standard double lane change, do not contain any acceleration. The torque of the powertrain is applied to the rear wheels, so no input torque is to be expected at the front suspension that is examined in this research.

The acceleration maneuver is a pre-defined straight-line event in ADAMS CAR. At the start of the event, the car is moving at a starting velocity without acceleration. The acceleration step period within the event is defined by a start and a duration time. The acceleration is the result of the throttle applied to the powertrain. From the start of the acceleration period, the throttle ramps up linearly until the given percentage of the maximum throttle is applied at the end time [42].

Two separate acceleration events shall be examined with the goal of comparing the network's performance afterwards. The first event is designed with the goal of mimicking a situation from the racetrack data – a situation that the neural network is trained on. The second event is to test the network on an unknown acceleration situation.

Acceleration Event 1

The neural network will first be tested on acceleration maneuvers that it was trained on beforehand. To simulate the behavior of accelerations similar to those appearing during the

racetrack, the throttle value is chosen so that the vehicle reaches acceleration values similar to the values during the simulated racetrack (see left side of Figure 3.5). The maximum acceleration stays between 0.03 m/s^2 and 0.04 m/s^2. The starting velocity is set to 1 km/h to eradicate errors from ADAMS CAR because the vehicle is moving under the idle speed. Still, discontinuities are caused in the MBS due to the small velocity in the denominator of the tire slip calculation (refer to [1] for detailed definitions of tire slip). The duration of the acceleration step is set to 0.1 s. At the end of this step, maximum throttle is reached.

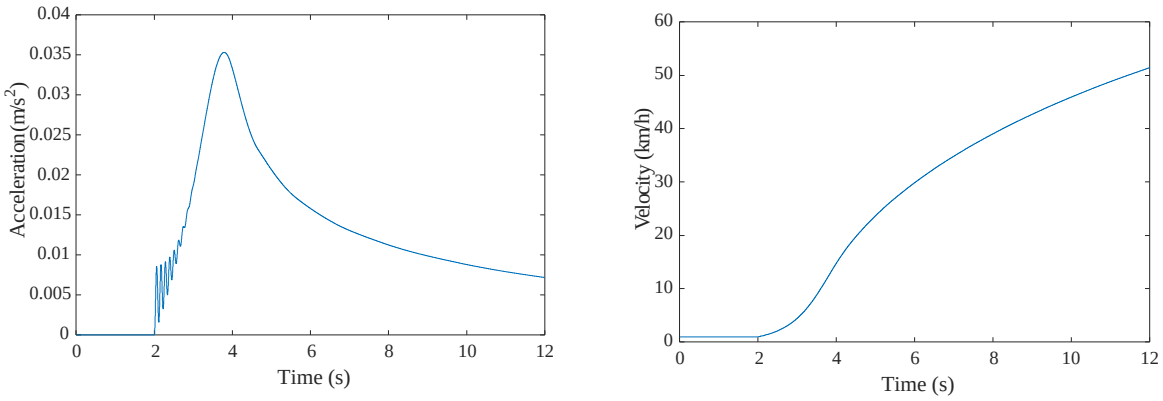


Figure 3.7 Longitudinal chassis acceleration (left); Longitudinal chassis velocity (right) - Acceleration Event 1

Acceleration Event 2

In this configuration, 100 % of throttle is chosen as the maximum throttle, since the car is designed for races, in which often the maximum acceleration is desirable. For example, at the start of a race, the car must accelerate from a stationary position (0 km/h starting velocity). Therefore, initial velocity for the event is set to 0.01 km/h, because it is not possible to choose 0 km/h as starting velocity in ADAMS CAR. The duration until maximum throttle is reached is set to 0.1 s like in the previous event. To accommodate for the fast acceleration and to prevent the locked steering to cause turning at high speeds, the length of this event is reduced from 12 to 10 s.

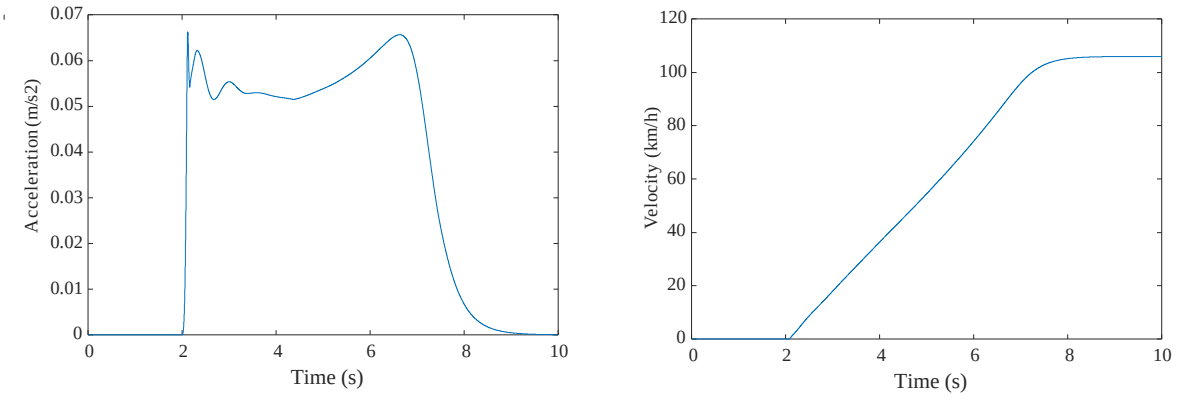


Figure 3.8 Longitudinal chassis acceleration (left); Longitudinal chassis velocity (right) - Acceleration Event 2

At the end of the event, the car has reached a velocity of approximately 105 km/h, which is close to the top velocity that is reached during real formula student races. Locked steering is chosen in order to keep the vehicle on a straight line.

3.1.5 Braking Test

To test the neural networks capability of predicting the dynamic behavior of the suspension during braking, a braking event in ADAMS CAR serves as a test. Since the A40-02 model has brakes on the front wheels, a braking torque is applied to the wheel hub in addition to the reaction forces and torques from the suspension and the tire. Furthermore, this is the only test setup in this research that contains braking.

Like the acceleration maneuver, the braking maneuver is a pre-installed straight-line event in ADAMS CAR. At the start of the event, the car is moving at a constant starting velocity. After the start time is reached, the brakes start to apply a torque to the wheel and deceleration begins. ADAMS CAR provides the options to either select open-loop braking, in which case a braking value has to be entered that is dependent on the way the brakes are designed, or to select closed-loop braking, in which case the software will try to maintain a constant deceleration. The simulation ends at the user defined end time or when the vehicle falls below a minimum speed (1.38889 m/s).

Two separate braking events shall be examined with the goal of comparing the network's performance afterwards. The first event is designed with the goal of mimicking a situation from the racetrack data – a situation that the neural network is trained on. The second event is to test the network on an unknown braking situation.

Braking Event 1

During the simulation of the racetrack in section 3.1.2 the maximum deceleration from 100 km/h before a turn was approximately 0.4 G (see Figure 3.5). Therefore, in this test configuration the constant deceleration is defined at 0.4 G and will be achieved and maintained by the closed-loop braking option.

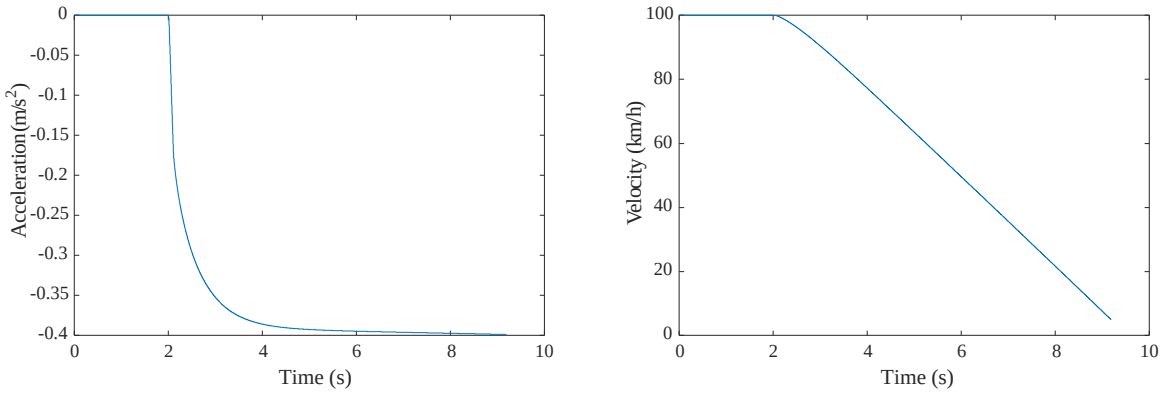


Figure 3.9 Longitudinal chassis acceleration (left); Longitudinal chassis velocity (right) – Braking Event 1

100 km/h is selected as the starting speed since this velocity is often reached during long straight section on a racetrack. Locked steering is chosen to keep the vehicle on a straight line. With this configuration, the simulation stopped after 9.185 s because the velocity fell below the minimum value of 5 km/h.

Braking Event 2

For the second braking event, a different configuration is chosen. Starting from 100 km/h, the open-loop brake begins to brake at $t = 2\ s$ with the maximum braking throttle. As can be seen in Figure 3.10, the deceleration happens faster and more abruptly than during braking event 1 (compare to Figure 3.9).

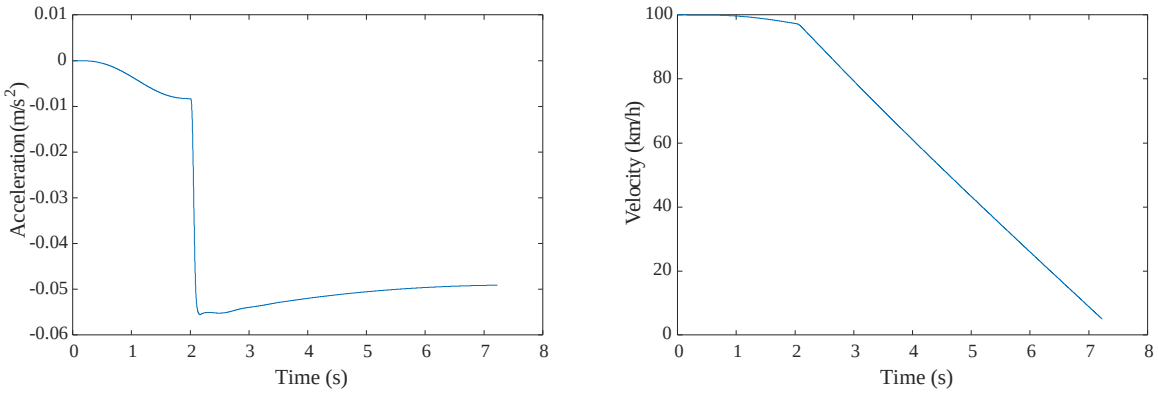


Figure 3.10 Longitudinal chassis acceleration (left); Longitudinal chassis velocity (right) – Braking Event 2

Apart from losing some velocity at the beginning due to the higher driving resistance at higher velocities, the velocity decreases similarly to braking event 1. The gradient (acceleration, compare Figure 3.10 to Figure 3.9) is less. Consequently, the vehicle reaches the minimum velocity faster, after 7.227 s.

3.1.6 Step Steer Test

One of the most important aspects of vehicle dynamics, is the dynamic behavior of the vehicle while taking turns. In the step steer maneuver, the vehicle reacts to an almost instant steering wheel angle change. To test the neural network's capability of handling this rapid change, this cornering maneuver is tested. After the steering step, the vehicle merges into a steady state skid pad test with a constant steering wheel angle which additionally shows how well the network can predict the suspension's behavior while steady state cornering. Furthermore, the step steer maneuver serves as a way to introduce a vibration into the vehicle as a complete system [22].

Other than the two previous discussed maneuvers, this event is categorized in ADAMS CAR as an open-loop steering event. The user must define a start time and an end time of the steering step, as well as the final steering value.

To examine the vehicle's response to the change, the step duration must not be too long. To approximate an instant change, the steering step duration is defined at 0.001 s, which is also the overall step size for the simulation.

Step Steer Event 1

The longitudinal velocity while cornering during the racetrack simulation is in the range of $10 - 30$ km/h and the steering wheel angle between 7 and 20°. Therefore 30 km/h is chosen as the constant velocity and 20° as the steering wheel value for the step steer test. The course

of the steering wheel angle is depicted in Figure 3.11. In the Adams Car simulation, a steering wheel angle of 90° corresponds to no steering.

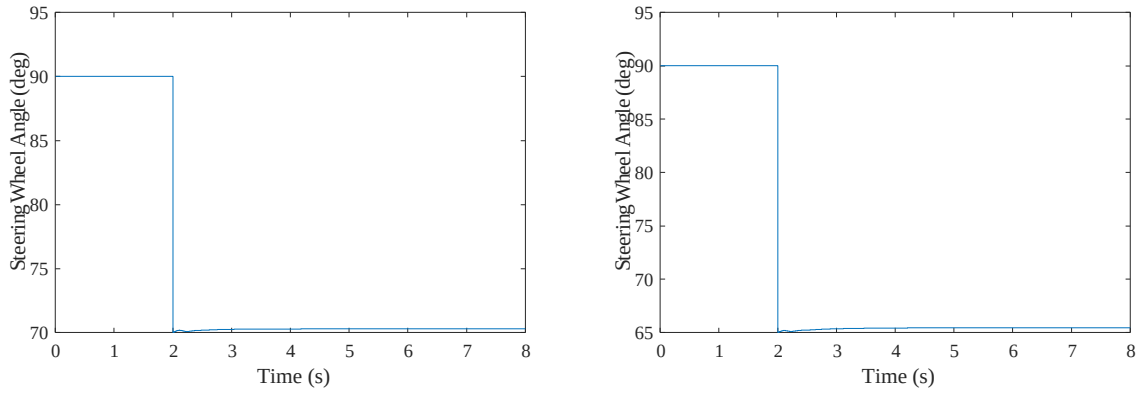


Figure 3.11 Steering wheel angle - Step Steer Event 1 (left); Step Steer Event 2 (right)

Step Steer Event 2

To test the neural network on another step steer situation, in which the steering value and the velocity are higher, and therefore suspension behavior is ultimately different, another test is introduced. The constant velocity is set to 35 km/h, while the steering wheel angle jumps to 25°. Simulations with the velocity or the steering wheel change being higher have shown that the tires lose grip to the surface and the vehicle starts skidding off the course.

3.1.7 Double Lane Change Test

The double lane change ISO 3888-1 [43] was developed to test lateral vehicle dynamics. During the test, an obstacle-avoidance maneuver is simulated by changing lanes twice while maintaining high speed e. g. on a highway. ISO 3888-1 [43] defines the dimensions for this test track. The vehicle moves between cones that are positioned relative to the vehicle's width. Afterwards it changes the lane during a 30 m long section and enters the new lane. The lanes have a 3.5 m offset. Next, the vehicle steers back into the original lane during a 25 m long section. The exact dimensions of the track are denoted in Figure 3.12.

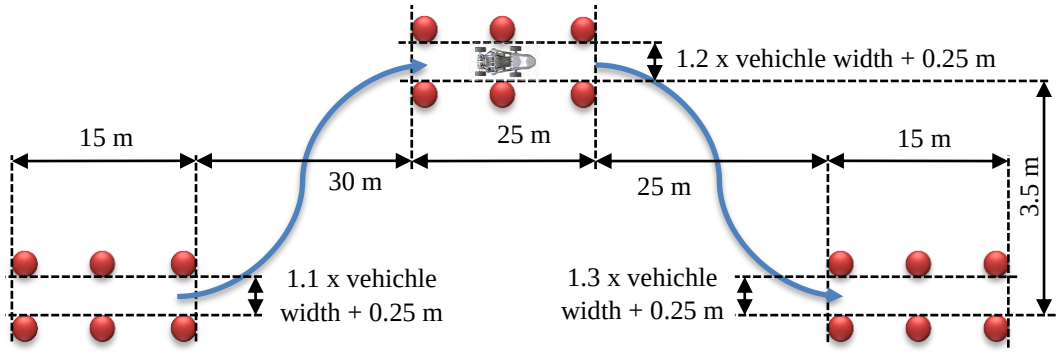


Figure 3.12 Test track for double lane change according to ISO 3888-1. Source: [43].

Ideally, the throttle position should not move during the maneuver. High speeds are recommended for this test. However, [43] also states, that any throttle position can be used during the test.

The double lane change is implemented in ADAMS CAR as a closed-loop steering event / course event. Instead of a driver, a PID controller is used to replace the human driver to control the steering and to keep the vehicle on the ISO-defined path. A longitudinal controller maintains a constant velocity [42]. In contrast to the ISO-3888-1 dimensions, ADAMS Car's standard ISO configuration contains an initial straight section that is 45 m longer. It should not affect the subsequent lane change maneuver and help the simulation to initialize before entering the test track.

Recommended by [43], the highest possible speed in which the vehicle stays within the tracks limitations should be chosen to perform the test. To find out, which speed is most suitable, four setups are chosen:

- Double Lange Change Event 1: 30 km/h constant speed
- Double Lange Change Event 2: 50 km/h constant speed
- Double Lange Change Event 3: 70 km/h constant speed
- Double Lange Change Event 4: 100 km/h constant speed

It should be noted that the vehicle never faces rapid steering changes at speeds over 30 km/h during the racetrack event. Thus, the NN is forced to predict suspension behavior in mostly untrained situations.

As can be seen in Figure 3.13, the vehicle stays within the track width in event 1 and 2 of the double lane change (constant speeds: 30 km/h, 50 km/h).

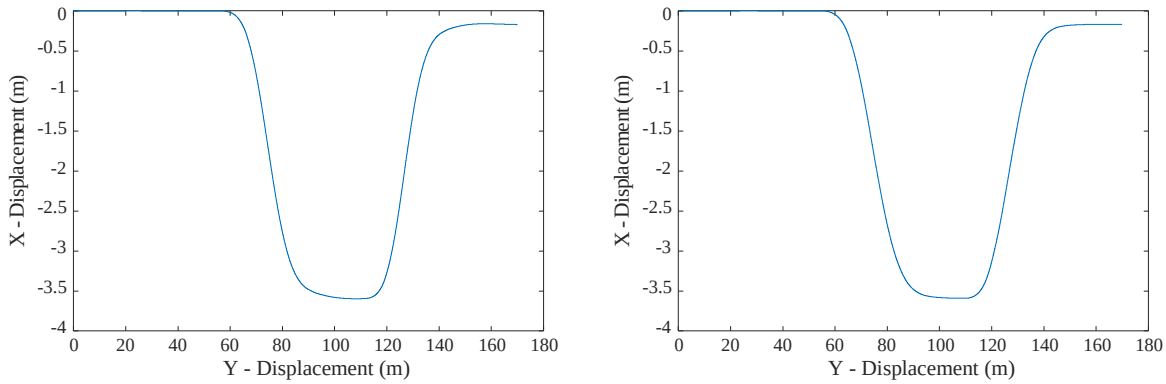


Figure 3.13 Chassis displacement wrt starting point - Double Lane Change Event 30 km/h (left); Double Lane Change Event 50 km/h (right)

In event 3 (see left side of Figure 3.14), has problems to merge the vehicle back into the original lane. It would need further investigation to decide, whether it stayed within the track width.

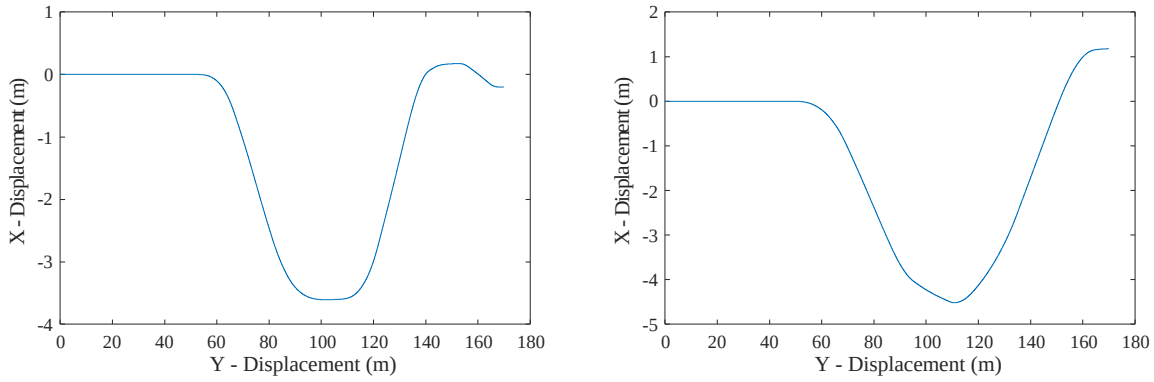


Figure 3.14 Chassis displacement wrt starting point - Double Lane Change Event 70 km/h (left); Double Lane Change Event 100 km/h (right)

During the fourth event (see right side of Figure 3.14), the car fails to merge into the lane and overshoots. When merging back, it misses the original lane by a large degree. During a real-world test, the vehicle would have left the allowed width of the track and touched at least one of the cones that are limiting the track. This maneuver shows driving in the critical region, the vehicle becomes unstable [1]. For the neural network model, this presents a new situation. The result is to be discussed in chapter 5.3.

3.2 Target Data Generation

The purpose of target data is to set a reference for the learning algorithms used later in the training of the neural network. Supervised learning requires the existence of true output to calculate errors that are minimized by the algorithm. Based on the input data, the generation of

target takes place in MATLAB / SIMULINK. An object-oriented model that was developed by KRACHT [5] is used. The steps taken to prepare the data are discussed in section 3.2.1 while the actual model is explained in detail in section 3.2.2. Its context is visualized below in Figure 3.15. For the training and testing of the neural networks, the focus stays on a few key output variables. Their selection and importance for further research and modeling activities is outlined in section 3.2.3.

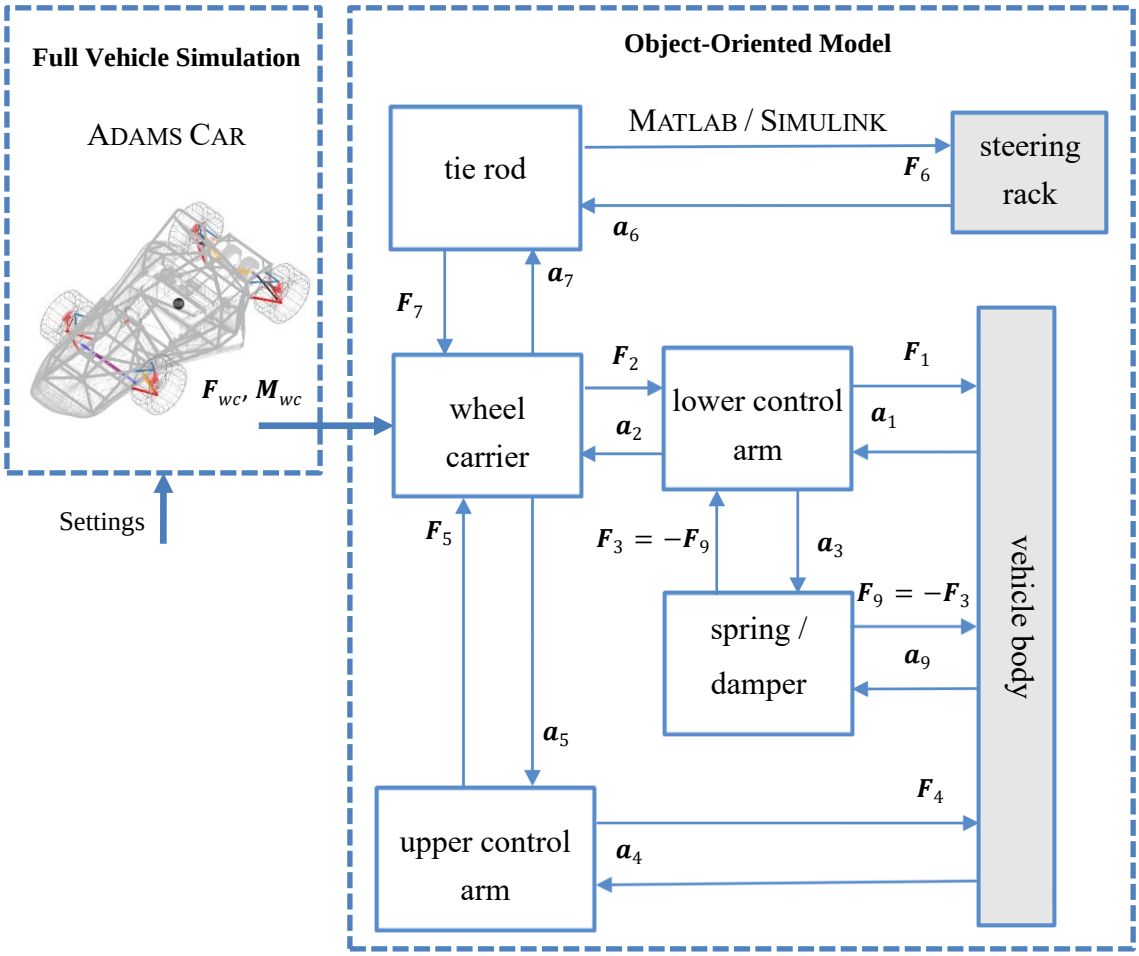


Figure 3.15 Simplified data flow for target data generation. Source: Object-oriented model representation based on [5].

3.2.1 Preparing the Dataset for the Object-Oriented Model

Starting a simulation in ADAMS CAR forces the car to be placed on the ground in the configuration it was designed. This also implies that springs and dampers might be in a different situation than during the design phase. To prevent training the network on unusual behavior caused by the start of the simulation, the first 0.1 seconds are cut off from the training set. To combine the training set from the original racetrack in the and the training set from the mirrored racetrack, the first seconds from the mirrored racetrack training set are cut off as well. This leaves the final training set with 239.865 s, equaling 239865 data samples. Every sample of

the input data consists of the attributes time, forces applied to the wheel hub in $x-$, $y-$ and $z-$direction (see Figure 3.16, top) as well as the torque applied to the wheel hub about the $x-$, $y-$, and $z-$axis (see Figure 3.16, bottom). Thus, the final training set results in a 239865×7 matrix.

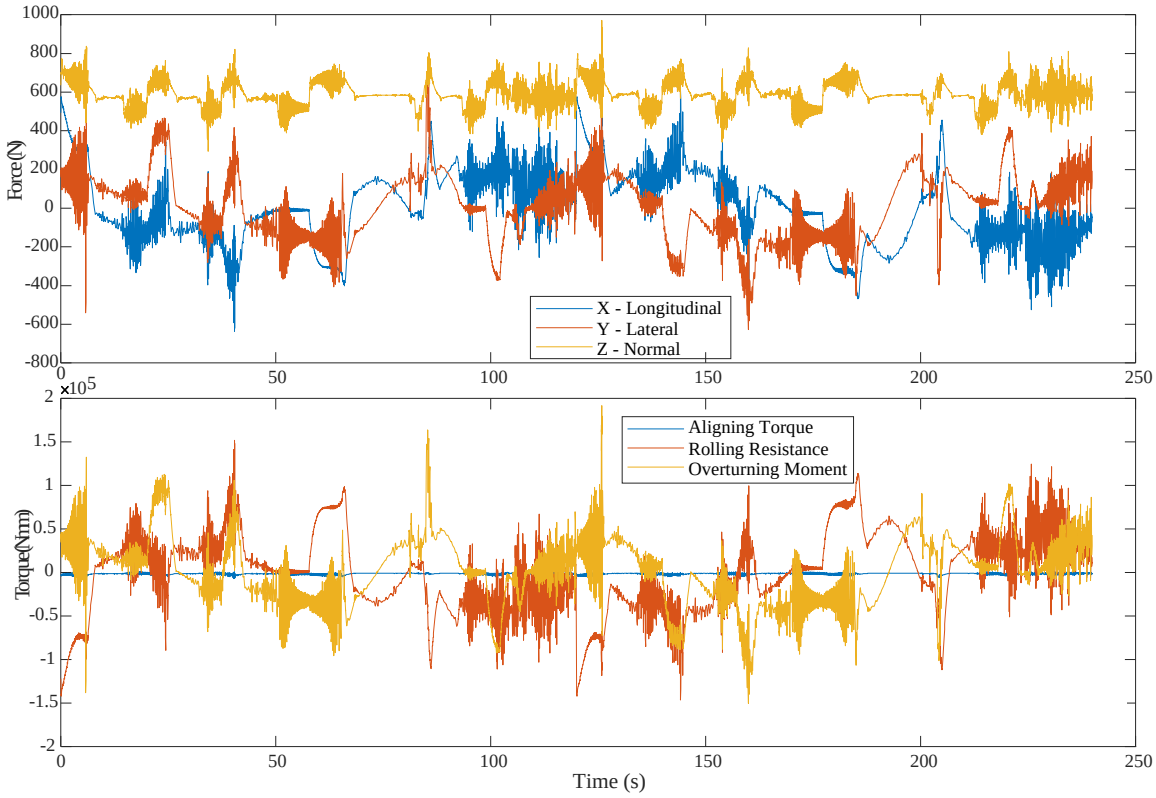


Figure 3.16 Wheel hub force (top) and torque (bottom) in the training input dataset

The applied forces and torques at the wheel hub are set to 0 for the first $0.5\ s$ to prevent the simulation to cause a discontinuity from the former idle state. Furthermore, the signal is passed through a low pass filter. High frequencies are found throughout the event due to the tire model that is used by ADAMS CAR. Therefore, frequencies that exceed the desired range are filtered out. This is done to ensure the numerical stability and accuracy of the simulation. Because the scope of the application is set to 0 to 30 Hz (maximum frequency from road roughness [22]), the low pass filter is applied as described in detail by KRACHT [5]. The application spectrum of the model should be 0 to 30 Hz. According to [5, 49], the time constant in the low pass filter needs to be $\tau < 0{,}005305$ s.

It should also be noted that the multibody simulation developed in the SIMULINK modeling environment uses a different orientation for the $x-$ and $y-$ elements of the force vector at the wheel hub. Thus, the signal for the longitudinal and lateral parts of the force vector and the

torques about those vectors are passed through a gain of -1 to accommodate for the rotation of the coordinates.

3.2.2 Object Oriented Model of the A40-02 Front Suspension

Wheel suspensions consists of only a few components. This makes component-wise modeling attractive. In object-oriented modeling, opposed to signal-based modeling, the physics of each component are stored in a single block, which makes this method highly modular. The blocks communicate via interfaces and connections that contain the state of a certain variable. The components are assembled to form the complete system in a common modeling language. Open and closed kinematic chains are possible [2, 5].

For this implementation of object-oriented modeling, NEWTON-EULER-equations with absolute coordinates are used to obtain generalized coordinates which avoid algebraic loops on component level. As a result, each object is described by kinematic differential equations that can be solved for the generalized coordinates which decreases the computational effort. Additional use of the force-coupling method (dampening by bushings) makes it viable for the use in real-time simulations [2, 5].

A component part of a wheel suspension can be described by their mass m_i, their center of mass CM, their moment of inertia $I_{CM,i}$ with respect to the CM_i and their J joints that link them other components. For simplification, all bodies are considered rigid. It is possible to model flexible bodies and use them for this model by discretization methods but are not discussed here. Joints to other bodies can be realized rigid or elastic (e. g. rubber bushings) [5].

In Figure 3.17, a body i with j (3) joints to other bodies is displayed. Forces and torques are shown in red. These contain the gravitational forces, reaction forces and torques (when linked to other bodies via rigid joints), and applied forces and torques that can result from interacting spring- / damper elements, rubber bushings or an anti-roll bar [5].

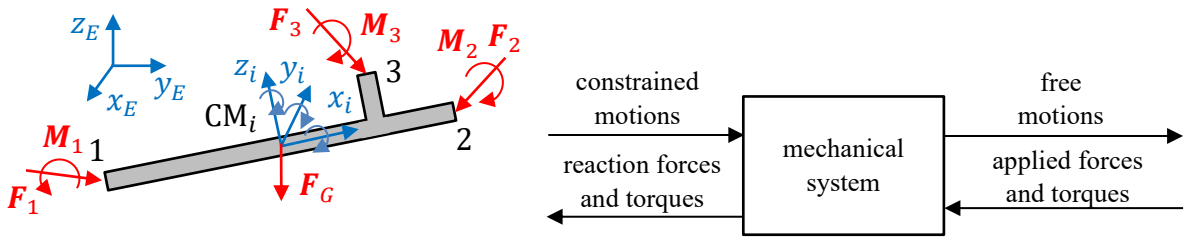


Figure 3.17 Rigid body with 3 joints (left), Signal flow of a single object (right). Source: [5].

In the first step of the modeling procedure proposed by KRACHT [5], the inputs (e. g. applied forces and torques) are transformed from the global coordinate frame to the body fixed coordinate frame by transformation matrices. Now, the NEWTON-EULER equations [1, 25, 50] can be applied to compute the angle acceleration $\boldsymbol{\alpha}_i$. With the help of the KARDAN equations

[25], the KARDAN accelerations $\ddot{\psi}_i$, $\ddot{\theta}_i$ and $\ddot{\varphi}_i$ are derived. Subsequently, the equations can be explicitly solved for the outputs ${}^i\boldsymbol{a}_j$, ${}^i\boldsymbol{v}_j$ and ${}^i\boldsymbol{F}_1$. These calculations vary, dependent on the type of joints. Algebraic software is used for computation. If the body is ideally constrained at multiple joints, the solution must be computed numerically due to an algebraic loop and is not usable for real-time simulations. Last, the outputs are transformed back into the global coordinate frame.

The spring / damper system is modeled using the forces F_S and F_D with known spring stiffness coefficient k_S, damper constant c_D, free spring length l_0. The time dependent spring length l, the rate of change of damper length and position vector $\boldsymbol{r}_{DC}$ determine the spring and damper force and their direction [5].

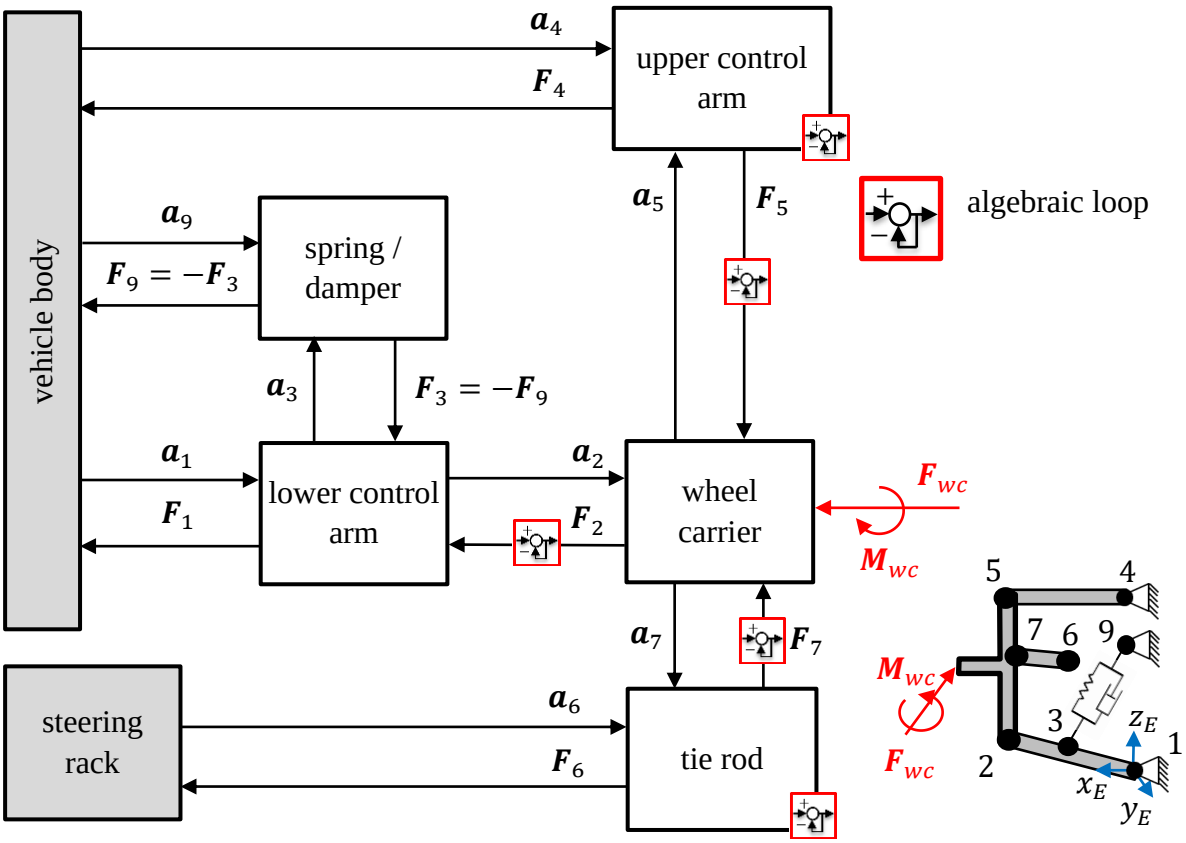


Figure 3.18 Object-oriented double wishbone suspension model. Source: [5]

All component blocks are assembled to the complete suspension system as in Figure 3.18. It was tested and validated with an ADAMS CAR reference model. This model contains algebraic loops. A solver has to be applied to computational intensively solve the algebraic loops. Therefore this model is not real-time capable. Variants of this model that allow the transformation of DAEs into ODEs are discussed in detail in [5] and in [2] in English language.

3.2.3 Key Output Variables

To succesful model dynamic applications for a suspension, the model must be integrated into a full vehicle model, for instance to create a vehicle simulator. As a foundation, a twintrack model is often used. For a model with kinematic wheel suspensions, the deflection of the wheel and their velocities as well as the force applied to the vehicle body by spring and damper need to be determined to serve as interfaces [1]. The object-oriented approach as explained above, employs forces and accelerations as variables between object blocks but also computes the position of the wheel carrier [5]. If acceleration or position can be modeled with enough accuracy, the velocity can be obtained by deriving the position vector time series or integrating the acceleration vector time series. For the latter procedure, the integration constant needs to be addressed. In a twin-track model, steering is considered as input to the wheels [1]. Therefore, the mechanics of the steering system are not included in the key output variables.

In conlusion, having the objective of implementing the suspension model into a complete vehicle model in mind, the focus is set on a few key output vectors: wheel carrier acceleration $\boldsymbol{a}_{wc}$, wheel carrier position $\boldsymbol{r}_{wc}$ and spring damper reaction force $\boldsymbol{F}_{lcas}$.

4 Neural Network Implementation

In this chapter, the implementation of a neural network in MATLAB is presented. First, the reasons that lead to the selection of the NARX model are outlined. In section 4.1, its specifics are explained in detail. Afterwards, in section 4.2, the procedure of pre-processing is addressed. This is done to prepare the data for the application of the NN model. Next, the training phase is discussed in section 4.3. To optimize the network for the given problem, a brief hyperparameter optimization is carried out in section 4.4. Finally, section 4.5 covers the post-processing.

Now that the structure of the problem has been explained and the dataset has been prepared, a neural network model selection must be made. Recall from chapter 2.2, that a recurrent neural network is well-suited for dynamic problems like this. A feedback connection introduces previous time samples into the model and embeds memory that is learned through BPTT algorithms.

NARMAX models have proven to be efficient neural network architectures for a wide variety of non-linear prediction and modeling tasks [28]. Within the class of NARMAX models, NARX models stand out as computationally powerful tools. They were originally designed to act as control systems or to identify the behavior of non-linear systems [4]. Due to the use of feedback delays and their function in BPTT algorithms, NARX networks have shown to be capable of learning long term dependencies better than usual RNNs (who are possibly more complex, having a higher number of hidden layers) while also converging faster [27, 51]. So that, in contrary to standard RNNs, NARX networks can replace fully connected RNNs without losing computational power [52]. In fact, NARX models with an order larger than one can simply be transformed into any RNN and vice versa if the activation functions are similar [31]. Not needing to scale up hidden layer sizes, terms of higher order are modeled efficiently by introducing lagged inputs and feedbacks. This makes NARX models computationally parsimonious: just a few hundred data samples are often enough as training datasets [4].

NARX models have proved good performance in tasks like non-linear system identification [4], prediction for chaotic time series [27], non-linear filtering or modeling non-linear dynamic systems [34]. For instance, SALEHINIA ET. AL. [21] successfully modeled the forwards kinematics of a 6 DoF parallel manipulator using the NARX approach. [16, 53] show the ability of modeling damper systems by NARX networks.

A downside of NARX networks is their explosion in training computation time when the number of delays, hidden nodes or dimension of in- and output are increased [54] (see section 5.4 for details). To keep the computational effort relatively low, the modeling of the three key output variables is split up into three networks, that are optimized separately. The training process implemented in MATLAB is outlined in the following procedure:

1. Load the input and target data into the workspace and perform pre-processing: normalize all datasets (section 4.2.14.2.2), split inputs and targets into training-, validation- and test data (section 4.2.2), and prepare initial layer states for training (section 4.2.3).

2. Create a NARX network architecture with hidden layer size N, input delays d_u, feedback delays d_y in open-loop configuration. Set the number of input and output nodes N_u and N_y according to the dimensions of input and target vectors and initialize biases and weights of the network (section 4.3.1.1).

3. Train the network in the open configuration see section (section 4.3.2) with the goal of minimizing the loss function `MSEreg` by applying the BAYESIAN Regularization algorithm. Use parallel computing if available. Stop training when minimum gradient, maximum Mu or maximum epochs are reached. These stopping criterions are explained in detail in section 4.3.1.2. Afterwards, close the feedback loop and test the network.

4. Train the neural network in the closed-loop configuration (section 4.3.3). Instead of delaying target values and feeding them into the network, estimated values from the network are delayed and fed back into the network. Then test the network again in the closed-loop configuration.

5. Remove or add a delay from / to the network and test again with $n_{y,new} = n_{y,old} \pm 1$ and $n_{u,new} = n_{u,old} \pm 1$. If the performance improves, repeat this step until the performance stops improving and update the network with the best-performing delays.

6. Keep the network from either step 3, 4 or 5 with the best performance. Repeat steps 3 to 6 until I iterations have been conducted.

4.1 The NARX Model

For the learning phase, NARX networks can be configured in two ways [27]:

Open-loop configuration (also called decoupled or series parallel architecture): The network takes the input and its delays, as well as delays of the target (true) output as input and feeds it into the hidden layer. The architecture becomes a feed-forward network. Learning in the open-loop configuration is executed by applying the static LMBP algorithm. The open-loop performance achieved after this training is only an indicator for the performance that the network will have in a simulation application because the target output will not be available.

Closed-loop configuration (also called coupled or parallel architecture): The output of the network is tapped and fed back into the hidden layer of the network, which turns the network into a RNN. This feedback and its delays replace the input of target output $T(k)$ in the open-loop configuration. The closing of the loop is presented in Figure 4.1. Learning in this

configuration happens with the LEVENBERG-MARQUARDT Algorithm that is adjusted to BPTT. New data can be applied to the model in this configuration.

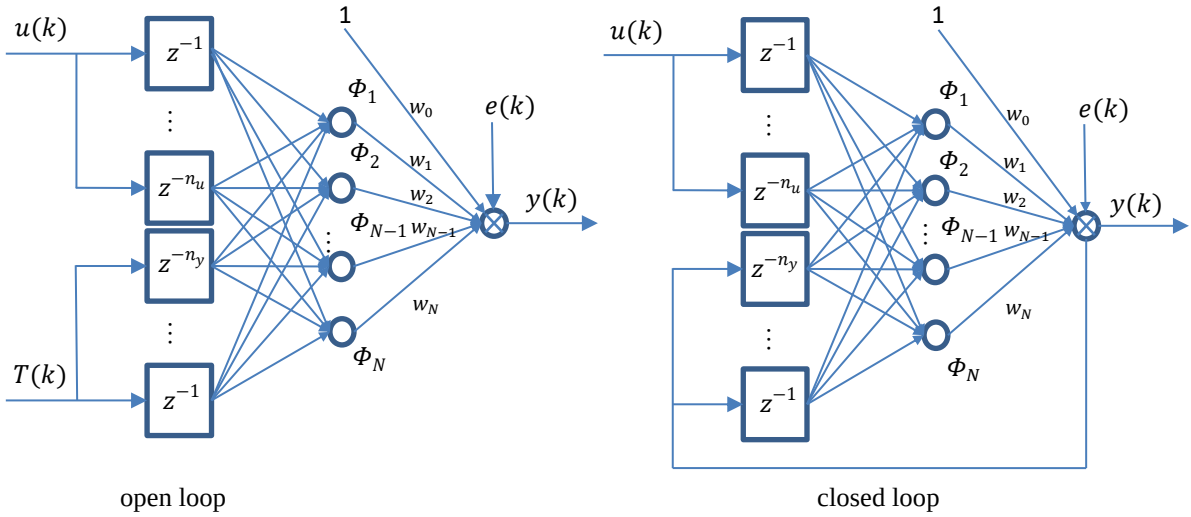


Figure 4.1 NARX network in both configurations with single input and single output. Source: Based on [4].

To describe the mathematics behind the model, one has to look at the relationship between the input u and the output y. NARX models feature input delay states, which are denoted by d_u for the n_u input delays and d_y for the n_y feedback delays, assuming that input is available in discrete timesteps [30]. Together, both delays describe the time window that the model's computation is based on [27]. In Figure 4.1, the delay is represented by delay blocks, specified in Figure 4.2. The block's output

$$a(k) = u(k-1) \qquad (4\text{-}1)$$

is the input from the previous timestep. At $t = 0$, the block must be initialized with a value $a(0)$. The relationship for the output of a 1-step delay is [30].

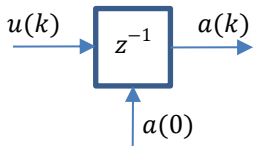


Figure 4.2 Delay block. Source: Based on [30].

As discussed previously in chapter 2.2, NARX and NARMAX networks belong to the class of single layer networks (SLNs). In this single hidden layer, m nodes are operated in a parallel structure of which a single node is described by the index i. Furthermore, a noise function e is lead into the output layer. This zero-mean sequence covers measurement and modeling errors,

as well as other disturbances and noise that can arise from data which is gathered in the real world. The independent error term $e(k)$ distinguishes NARX networks from NARMAX networks, which includes noise dependent error [4]. Containing a noise term is beneficial for the generalization performance [36].

To simplify, a NARX network with single input and single output (SISO) characteristics is considered. Like in chapter 3.1, the activation function that is used in all hidden nodes is called ϕ_m and the input $x(k)$ that enters a hidden node m and the corresponding weights w_m are defined like in chapter 3.1. The signal at the output node is described by the function F [4]

$$y(k) = F[x(k)] = w_0 + \sum_{m=1}^{N} w_m \phi_m\big(x_m(k)\big) + e(k) \qquad (4\text{-}2)$$

that acts as a transfer function of the system. The output node sums up the bias, the weighted signals from the hidden layer that are passed through the non-linear activation function $\phi_m\big(x(k)\big)$, and the independent error term. At the hidden layer, inputs

$$x_i(k) = \begin{cases} y(k-i), & 1 \le i \le n_y \\ u(k-i+n_y), & n_y + 1 \le i \le n_y + n_u \end{cases} \qquad (4\text{-}3)$$

are defined by the lagged inputs and feedbacks to the model. Note that in this original description by BILLINGS, the node's input is either the lagged input or the lagged fed back output but not the direct input. A configuration like this is required when prediction tasks are performed, since there is no current input vector available. When necessary, a connection from the input directly into the hidden layer can be added (see section 4.4.2). Another way of formulating the the way in which NARX models work is an implicit representation [4] of the output $y(k)$.

$$\begin{aligned} y(k) = F\big[&y(k-1), y(k-2), \dots, y\big(k-n_y\big), \\ &u(k-d), u(k-d-1), \dots, u(k-d-n_u)\big] + e(k) \end{aligned} \qquad (4\text{-}4)$$

NARX models can also be described by polynomials. To derive the polynomial representation of the network, the network has to be analyzed to find the predominant parts of the network [4]. The method of identifying and sorting out irrelevant parts to reduce the network's complexity is called pruning. For an overview of those methods, see [29]. Some algorithms like the BAYESIAN Regularization already determine the effective parameters used for the computation [34].

Explained above for the SISO case, all equations can be extended to fit the multiple input, multiple output (MIMO) case. Inputs, lagged inputs, output nodes and lagged feedbacks need to be multiplied according to the new input and output. Still, all of the aforementioned elements need to be connected to the hidden layer by indivual weights [4].

4.2 Pre-Processing

Transforming raw data into signals that can be fed into the neural network is called pre-processing. It is used to remove redundant or irrelevant information and help the network to increase the quality of output, ultimately resulting in a less complex network that can converge faster and generalizes better. It should be noted that if the input data is pre-processed before training, the input data used for testing needs to be pre-processed the same way (see Figure 4.3). Following this train of thought, output data that was generated by the network, trained on pre-processed target data, needs to be post-processed in a reverse manner to obtain data that is comparable to the raw dataset [29].

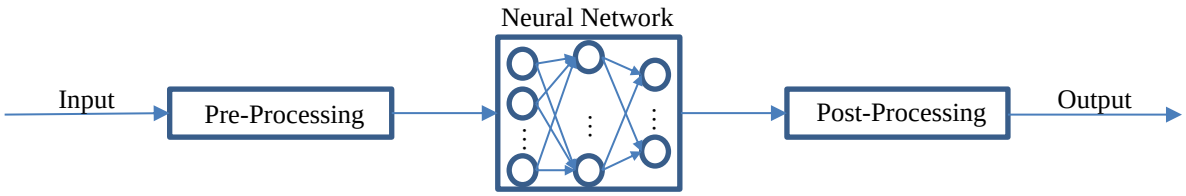


Figure 4.3 Pre- and post-processing in terms of neural networks. Source: Based on [29].

Feature reduction or feature selection is often used in deep learning for high-dimensional raw datasets [36]. For this application scenario, all features are needed to be fed into the network, since every attribute (vector component of force or torque) contains necessary information that could not possibly be carried by other attributes and ultimately has an effect on the suspension behavior [29]. Other usually conducted pre-processing methods include the removal of systematics such as trends (constant increase or decrease of the average value) or seasonality (periodic patterns due to periodic influence) because they would deter the network from learning the problem rather than learning the systematics [26, 27]. Both trend and seasonality are not expected to occur during the vehicle simulation because of its tie to the underlying mechanical system which does not produce trends or seasonality.

Pre-processing implies the need for post-processing. After leaving the neural network, the signal needs to be re-transform into its raw state so that it can be compared to the raw target data [29].

4.2.1 Normalization

Activation functions like the logistic or the hyperbolic tangent function possess a certain range. To increase the efficiency of the network, the range of the input data should match the range of the output [28].

The function `mapminmax`, included in MATLAB, maps the input of the network to the range of `ymin` to `ymax`. It follows the expression

```
y = (ymax-ymin)*(x-xmin)/(xmax-xmin) + ymin;
```

to calculate the output `y`. In the standard configuration, `ymin` is set to -1 and `ymax` is set to 1. Other approaches to normalization require to divide the inputs by the 2-norm of the input vector [28].

A similar procedure is applied to map the mean value of each individual time series to 0 and the standard deviation to 1 by implementing

```
y = (x-xmean)*(ystd/xstd) + ymean;
```

into the code or simply using the MATLAB-function `mapstd`.

The functions described above also allow to save the normalization parameters and reverse-normalize the output after training or applying the network in the post-processing. SIMULINK blocks `mapminmax` and `mapstd` are available [30, 55].

4.2.2 Data Division

The two key working paradigms to which machine learning can be applied are called induction and transduction. The objective for *induction learning* is to gain knowledge about the input and output relationship, so it can be applied to new data. In contrary, the reasoning for *transduction learning* is obtained from observed cases and applied to new test cases directly. This is particularly useful if test- and training data are not distributed the same way [29].

Before training can commence, the total dataset is to be divided into three subsets:

- *Training Data:* This set of data will be used during the learning phase to alter the weights and biases and to determine the gradient of the learning process.
- *Validation Data:* The second objective besides minimizing the error function is minimizing the generalization error. This is measured on a different set of data, called validation set. Its performance is determined during the training process. This value typically rises when the data begins to overfit [29].
- *Test Data:* Not being used for the learning phase, the test data is evaluated after each iteration and is an important tool for the user. The performance on the test set shows how well a network can apply its learned knowledge on a new test set, that it has not been trained with. Therefore, it allows to compare different models.

For smaller datasets, a train / test / validation split of equal sizes (33 / 33 / 33) is problematic, as it can cause the trained network to become sub-optimal due to the lack of sufficient training data. On the other hand, a split of 80 % or more training data should also be avoided because validation and test can become unreliable. However, the bigger the dataset is, the less important

is the split of the data. In general, a share of 50 % to 70 % of training data is advisable [56]. Given the large dataset available, MATLAB's default configuration [34] of 70 / 15 / 15 is chosen for the training.

Splitting the data can be done randomly for each data sample. Other approaches are splitting the data block-wise or in more complex ways [34]. There are multiple advantages of choosing the block-wise division mode for this dataset and network model: First, NARX networks use delayed inputs from previous time samples. To train, validate or test the ability to estimate an output, previous estimated samples need to be available. Second, performing the testing of the network on the last block of the dataset, the network will be trained on a different situation from the racetrack that it is tested on. This is desirable because further tests and possible real-world application will also be performed on new data blocks, not individual samples. In this case, the performance is an indicator for the generalization ability.

4.2.3 Initial Delay States

Considering the network uses previous time samples for its delayed inputs and feedback delays, these layer states need to be initialized before training can start. When training takes place in open configuration, the feedback delay states are initialized with target values from the n_y-first timesteps and the input delay states are initialized with the inputs of the n_u-first timesteps. There are no estimated outputs available when starting training in closed-loop configuration, the delay states must be initialized the same way. However, in a real-world scenario, target values are not available. This problem is addressed in section 5.2.

It should be noted that the time samples in the effective training dataset are reduced by the shifted amount of time samples which can have a big impact when smaller datasets are used. To prepare the time series for training, a MATLAB command `preparets` can be executed:

```
[X,Xi,Ai,T,EW,shift]=preparets(net,inputs,targets,feedback,EW)
```

Using information about the input- and feedback delays saved in the `net`-object, this function returns the resulting `shift` in time samples, the effective network input `X`, the initial states for the input delay `Xi` and feedback delay `Ai`, and the training targets `T`. In case individual error weights are used for training, they can also be prepared by this function [34].

4.3 Training

After initializing the weights, biases and defining the stopping criteria, the training is executed in open-loop configuration first. Afterwards, the same network is undergoing training in the closed-loop configuration to improve the test (and therefore generalization-) performance.

4.3.1 Training Parameters

4.3.1.1 Weights and Bias Initialization

The goal of initializing weights is to optimize the convergence of the network. If the weights are chosen poorly, the network might get stuck in a local optimum or slow down the convergence. Instead of initializing the weights with zeros in every case, randomized values with a mean of 0 and a standard deviation of 1 are used to steer the direction of the gradient descent into a specific direction [34].

4.3.1.2 Stopping Criteria

To maximize the generalization potential, the training of a neural network should be stopped as soon as the quality of its estimation allows it to. This is the most common procedure to ensure a good generalization result. Avoiding overtraining, the training is stopped when it could still improve error function. When using a gradient-descent method in training the neural net, the range of values that appears most often in the output dataset are learned at first. This also implies that higher frequency components are potentially left out when early stopping is applied. Based on repeatedly rising generalization errors, the algorithm will stop the training process and leave the network with the parameters from the iteration with the minimum generalization error [29]. Next, some stopping criteria for the learning phase are explained:

- *Maximum epochs:* In epoch wise learning (also called offline or batch learning) weights are changed according to the algorithm only after every sample of the training dataset has been passed through the model [28]. A maximum amount of epochs can be defined before the training phase to keep the computational effort low [34].

- *Minimum gradient:* When optimizing a cost function E, the gradient refers to the generalized derivative of the function with respect to a vector. [36]. Gradient descent algorithms generally choose the direction in which the performance function of the network decreases the most, this is where the gradient is the lowest [34]. See section 2.2.4 for a description of the LMBP, one variant of a backpropagation gradient-descent algorithm.

- *Maximum Mu:* When training a neural network by LMBP, the learning rate is determined by the control parameter Mu. A big Mu value indicates learning with a small step size. After a reduction in the cost function, Mu is decreased and increased, when only a bigger step would allow the cost function to reduce further [55].

- *Maximum time:* The user can also set a maximum time for the training of the network. After the time is reached, the training phase is canceled [55].

- *Repeated increase in validation performance:* If training is executed by the LMPB algorithm and a share of data is allocated to validate the generalization of the network, the learning algorithm will stop after the validation performance has increased more than

predefined times. Afterwards, the network state with the best validation performance is returned [34, 55].

When using the LMBP algorithm, the cost function could converge abruptly, therefore it is important to choose the right parameters. Nevertheless, the amount of data samples available for the training is vastly larger than the model parameters. So it remains questionable if measures against overfitting are necessary [34].

4.3.1.3 Training Iterations

The network is initialized with different weights and biases every time it is created. To find the best network for the problem, the network has to be trained multiple times [34]. This is especially true if the network is trained with the LEVENBERG-MARQUARDT Algorithm in the closed-loop configuration. In this case the algorithm is more likely to get stuck in a local optimum [28].

4.3.2 Open-Loop Training

BEALE ET AL. [34] suggest training the NARX network in open loop configuration first, before closing the loop for testing (for a prediction use case). In this configuration, the (lagged) input enters the network along with the (lagged) target data (see Figure 4.4).

Advantages of this first step include the availability of parallel computing and faster BP algorithms that do not need to backpropagate through time (see chapters 2.2.4 and 4.1). This method is also referred to as *teacher forcing* [36] and has shown good results for the modeling of dynamical systems [16, 21, 53]. However, the model cannot be applied to new data in this configuration since target data is not available during simulation. The feedback loop needs to be closed first.

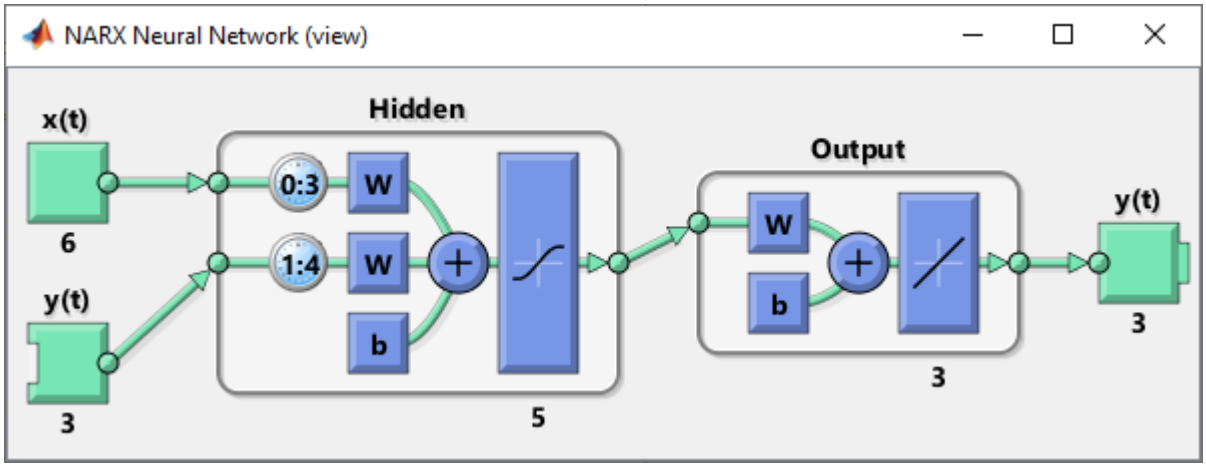


Figure 4.4 MATLAB representation of the NARX network in open-loop configuration

The network trained in open-loop configuration produces consistent results for the training and testing performance. However, test performance in this configuration must be viewed critically

since it still uses target data as input. Closing the loop afterwards and then testing the network with the test dataset produces high error measures deviating from the well-performing training errors, revealing that open-loop training alone is insufficient for the given problem.

4.3.3 Closed-Loop Training

After closing the loop, a feedback connection from the output to the input of the feedback delays is established. Now, the network is fed with its own estimations from the previous timesteps (see Figure 4.5).

Training is also possible in the closed-loop configuration. However, the learning algorithm needs to be adjusted to fit BPTT which eliminates the possibility of parallel computing and usually increases computation time. According to [34], retraining the neural network in this configuration can increase performance.

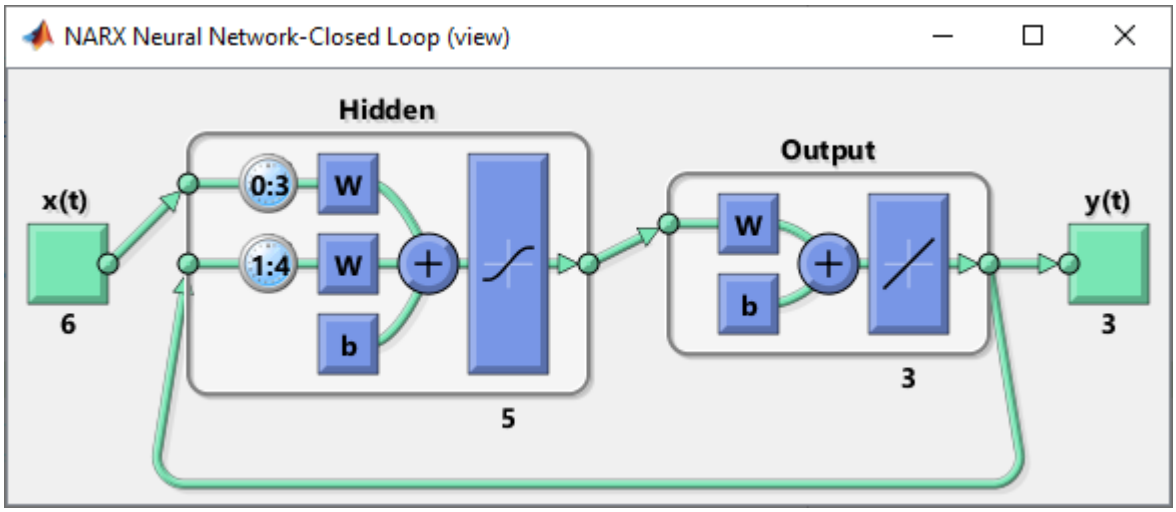


Figure 4.5 MATLAB representation of the NARX network in closed-loop configuration

After completing the training phase in closed-loop configuration, the network is tested again, now without targets as input. Instead, its own outputs are fed back. MSE and nMAE measures account to the true ability of the network to model new inputs, since the same configuration is also applied for simulation tasks. Below, the courses after 100 epochs of training procedures in both open- and closed-loop configuration are depicted.

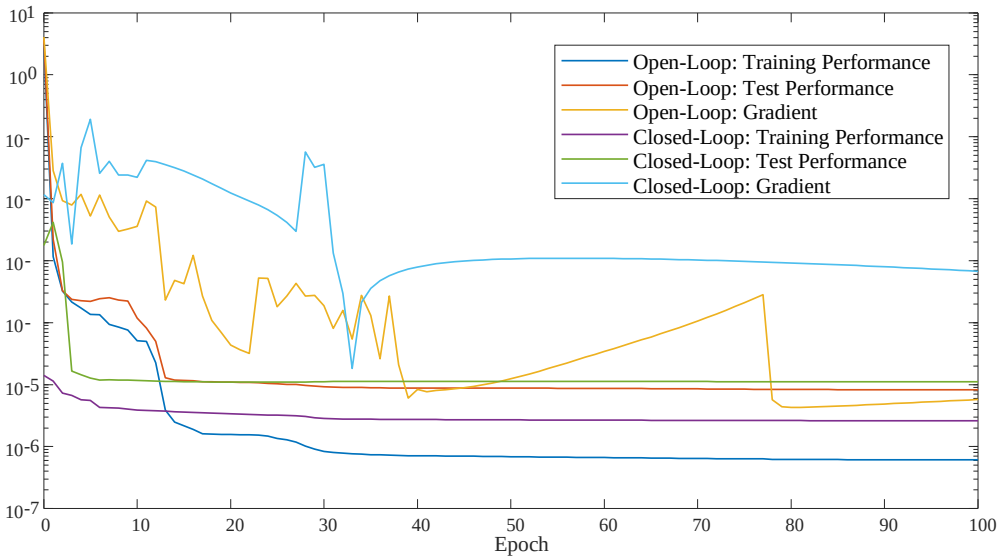


Figure 4.6 Training example

The performance accounts to the cost function, in this case the `MSEreg`, since `trainbr` was chosen as learning algorithm. In both instances, the test-performance stays above the training performance, which is expected: neural networks usually perform better on the exact problems they were trained on than on new ones. The gradients are both descending but have not reached their minimum value of 10^{-7} yet. Nonetheless, all performances have not improved significantly during the last 40 epochs. Therefore, a big improvement after 100 epochs is not expected anymore.

4.4 Hyperparameter Optimization

Hyperparameters (HP) are part of every machine learning system. In a neural network, parameters such as weights and biases are adjusted during the training phase and do not count as HPs. HPs are not considered by the learning algorithm and must be defined beforehand. For most machine learning methods, there is no universally optimal configuration of HPs, since they are highly dependent on the problems and the dataset at use [57].

Hyperparameter optimization (HPO) is computationally expensive. Especially neural networks are determined by many HP related to their architecture, learning algorithms or generalization abilities. In general, the objective is to improve performance by modifying the network according to the problem and dataset at hand. As a result, better reproducibility in scientific experiments is achieved. Often, HPO goes beyond the default setting of ML-libraries and toolboxes [57].

The computational effort for the HPO increases with the size of the dataset and the complexity of the configuration space, which is almost always high dimensional. To simplify, only some

HPs can be selected for optimization, in which case the tasks become selecting them and determining the optimization range [57], often by trial and error [29].

In theory, the global optimum of HP configurations can only be found by an exhaustive search of all possible sub-sets [29] which still does not address the problem, that error minimization *and* generalization are to be optimized [57].

A basic tool for searching for HP is grid search. The user sets the range of HPs to include in the grid search. Then, all combinations within the grid are evaluated. If HP are added to the grid, it causes exponential growth of required resources due to another dimension being. Grid search is recommended when the configuration space or range can be limited due to user experience [58].

A comprehensive grid search for all HP or the application of other HPO-algorithms like random search or BAYESIAN optimization, which are often used for deep learning purposes [36, 57], would presume the availability of more computational power. Due to the limited scope of this research, the focus is set to a smaller number of hyperparameters which are briefly analyzed to make an estimation for an optimal hyperparameter configuration.

MSE is the most popular measure of error [29] and also the optimal criterion for many NN applications [59], it will be further used for the error minimization during the training phase. However, *mean absolute error (MAE)* is an important measure of quality as well, especially if comparisons to other fields of research, outside of ML, are made. It is defined as [29, 34]:

$$MAE = \frac{1}{K}\sum_{k=1}^{K}|y(k) - T(k)| \qquad (4\text{-}5)$$

To make this measure independent of the scale of data, it is useful to normalize. It also allows the quality of different estimated output signals to be compared. This can be done by dividing it by the mean value of the time series. This has a significant drawback: when a time series is oscillating around 0, the denominator becomes very small, resulting in an overly big error measure. A similar problem occurs when error measures are chosen in which a division by a single target or output value is performed since the time series includes zeros. Therefore, in this thesis

$$nMAE = \frac{MAE}{|y_{max} - y_{min}|} \qquad (4\text{-}6)$$

is defined by dividing the mean absolute error by the range of estimated values. Finally, the correlation between test output and test target is a valuable measure of quality for the generalization performance of the network as well. It is quantified by the coefficient of

determination R^2. This is used to evaluate the effectiveness of the trained network on the test share.

As far as the architectural hyperparameters are concerned, the number input and output nodes are defined by the input and output attributes of the dataset.

At the start of the HPO, an assumption of the initial HP must be made. First, the hidden layer size is determined. Afterwards, the input and feedback delays are optimized with the previously identified optimal hidden layer size. Next, the activation function of the neurons in the hidden layer size is examined. Last, the training algorithm is inspected.

The procedure outlined at the start of chapter 4 is followed with the starting configuration:

- Input and feedback delays: 2 (default setting used in NARX time series example [55])
- Minimum gradient $= 1 * 10^{-7}$ or maximum $Mu = 1 * 10^{10}$ (default [55]). Maximum epochs $= 20$ (to shorten computation time, networks are trained longer for the final training of networks)
- Learning algorithm: BAYESIAN Regularization (chosen for generalization purpose)

Although the HP configuration is determined by the following quick tests, one must conclude that given sufficient computational resources, an exhaustive HPO could probably produce better HP configurations.

4.4.1 Hidden Layer Size

Smaller networks can emulate high-order non-linear systems. In fact, considering that a neural network with a single layer of hidden neurons can theoretically model dynamical systems with an arbitrary accuracy, it is necessary to ensure that the size of the single hidden layer in a NARX network is not chosen too small [28], since it is less flexibility and might cause underfitting. On the other hand, it has been shown that large networks improve the chance of overfitting [29]. While it is true that larger networks are generally more robust, having the increase in computational effort in mind, the network should not be designed larger than necessary to ensure better generalization abilities [28, 29].

The following computational expensive test to determine the hidden layer size is performed on the target vector *wheel hub acceleration* $\mathbf{a}_{\mathrm{wc}}$. This vector proved to be difficult to estimate for the network during initial tests and is also an important variable for the overall dynamics of the vehicle suspension. Training on a single vector requires less time than training on the complete set of variables and is therefore better suited for a time-consuming *trial and error* HPO. It is assumed that results from this test, performed on a single output vector, are translating well to the other output vectors.

To find the most adequate hidden layer size for the problem, different amounts of elements need to be tested with a simple starting configuration of the NARX network.

Tests are performed on layer sizes $1, 2, 5, 10, 20, 50$. The test for each layer size is repeated ten times. Acknowledging that the number of iterations is not enough to find sound scientific evidence for the best network configuration, it can give a rough estimation which layer size is efficient.

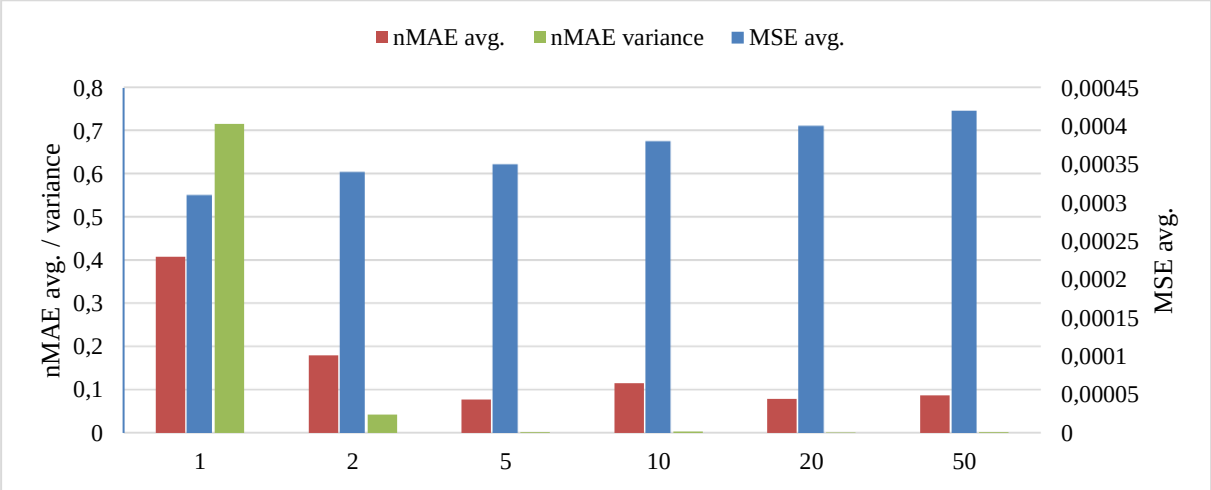


Figure 4.7 Results from hidden layer size optimization - single output signal

Displayed in Figure 4.7, the average nMAE and its variance decreases when the number of hidden nodes is increased from 1 to 2 and from 2 to 5 elements. After raising the number of elements past 10, these measures do not change significantly anymore. Having a low variance at 5, 10, 20 and 50 nodes shows that trained nets can produce consistent results after a certain threshold of nodes is surpassed. A slightly reverse trend can be seen at the MSE average which might indicate that the number of nodes should not be chosen too low. Another factor to consider here is computational efficiency. Training a network with a single hidden node in 20 epochs with the presented dataset takes a few seconds while training a network with more than 50 nodes takes ca. one hour (Intel Core i7-85550U / 8 GB RAM / Windows 10). The best results were achieved by a network that was trained with 2 nodes in the hidden layer (nMAE $\approx$ 2.92 %).

The same test as above is repeated with the goal of estimating not one component of the wheel hub acceleration but all three vector components. This test is performed to study the influence of multiple output signals on the best trained hidden layer configuration. Training on multiple outputs is requiring multiples of the computational effort of training on one output. Therefore, a test with a hidden layer size of 50 nodes is omitted.

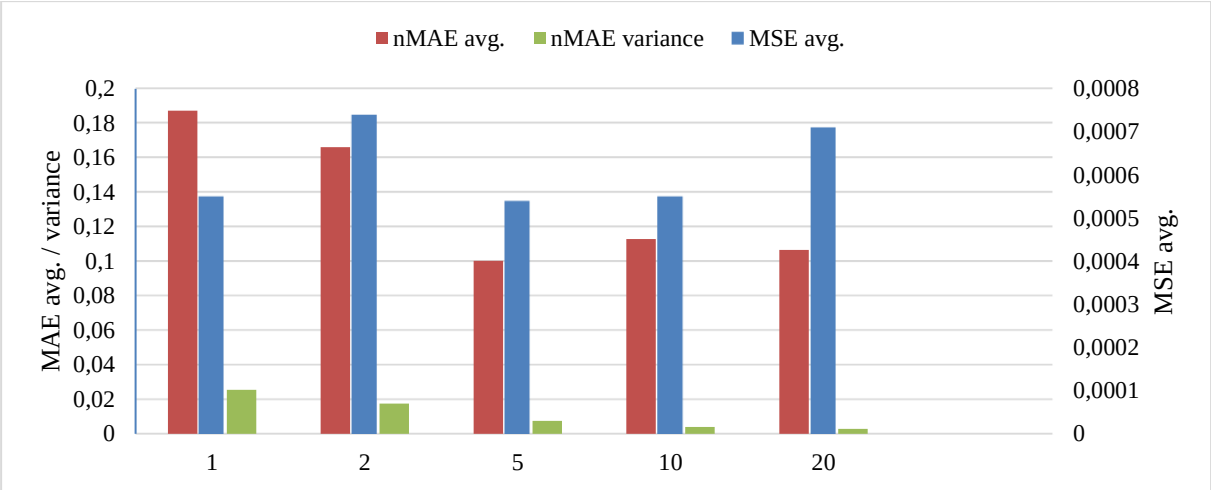


Figure 4.8 Results from hidden layer size optimization - vector output

One can find that the nMAE measure is decreasing if the hidden layer size is increased for a small number of nodes. After the number of nodes has surpassed 5, this is not the case anymore. Although a configuration of 10-20 nodes produces more consistent results, the nMAE and the MSE measure do not improve, confirming the result from the first optimization run. The best results were achieved by a network that was trained with 5 nodes in the hidden layer (overall nMAE $\approx$ 3.84 %).

Due to the findings in both of the previous test runs, the following tests are executed with NARX networks, containing a hidden layer of 5 nodes.

4.4.2 Input- and Feedback Delays

While it is possible to pick any combination of input and output delays, only continuous delay sizes will be considered, starting at 0-step delay for the input and 1-step delay for the feedback until the maximum input- and feedback delay n_u and n_y are reached. 0-step delay for the input simply means that the input is directly fed into the hidden layer. For the feedback delay, no 0-step delay is possible since the output is only available after the network has made its first estimation of the output. A well-trained network with BAYESIAN Regularization should only leave behind relevant parameters, so that an irrelevant delay is ignored due to its weight configuration when the network is used for simulation purposes.

An indicator of which lags lead to improved results, might be found in cross-correlation. Consider a time series, that has an output directly linked to the previous input signal. In that case, a peak in cross-correlation is located at lag 1. Cross-correlation shows the similarity between a time series x and a shifted time series y [55].

The network receives input (and target) data in normalized scales as explained in section 4.2.1. To not distort the cross-correlation, normalized inputs will be analyzed here as well. All inputs and outputs in scope of this thesis are multivariate, meaning they consist of multiple signal

streams. To find cross-correlation between each input and output signal, $6 * 3 = 18$ combination would need to be evaluated. Still, it is hard to predict the relationship between cross-correlation of certain input-output signal combinations and its effect on the neural network performance. Whether there exists a general dependence of the output on lagged inputs is analyzed by taking the mean of the six input signals and looking at the cross-correlation with respect to the output signals. In the Figure 4.9, the cross-correlation with respect to the key output vectors is shown. One can note that in all three cases, the cross-correlation is close to 1 across all examined lags in the range of -100 to 100. However, the peak is located at 0 lag, and no other local maxima can be found. Therefore, no assumption on the neural network performance with certain delay states can be made.

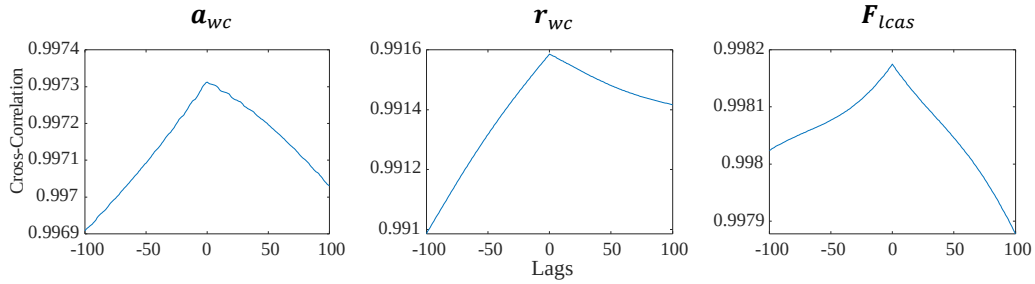


Figure 4.9 Cross-correlation of normalized input and output vectors

To find a possibly sub-optimal, but sufficiently well performing configuration for the number of input and feedback delays, a two-dimensional grid search is performed. First, the search is carried out by training the network against $\boldsymbol{a}_{wc}$, the wheel carrier acceleration. In Table 4-1, the results are listed for the corresponding maximum delays n_u and n_y, training the network with a hidden layer size of $N = 5$. Every configuration is tested in 10 iterations. The median values for these 10 iterations show how well networks with these configurations are usually performing in terms of MAE and how stable the outcome is. The overall best performing network has the configuration $n_u = 3$, $n_y = 4$. Since multiple training iterations are possible for the final training, the ability of a delay configuration to produce a well-performing network is key. Therefore, the delay configuration of $n_u = 3$, $n_y = 4$ is set for the next HPO tests and the rest of the research when wheel carrier acceleration modeling is the objective.

Table 4-1 Median nMAE results for delay size grid search

$N = 5$	$n_y = 1$	$n_y = 2$	$n_y = 3$	$n_y = 4$	$n_y = 5$	$n_y = 6$
$n_u = 0$	0.3096333	0.2662333	0.2785000	0.2653833	0.2752000	0.2536167
$n_u = 1$	0.0228500	0.0252000	0.0229333	0.0227667	0.0377000	0.0411833
$n_u = 2$	0.0230833	0.0218167	0.0234000	0.0209833	0.0203667	0.0390167
$n_u = 3$	0.0212333	0.0247333	0.0243833	*0.0301167**	0.0282167	0.0317167
$n_u = 4$	0.0270500	0.0249500	0.0464167	0.0211667	**0.0192167**	0.0205833

A drawback from increasing n_u or n_y is the more extensive initialization of delay states when the network is used for simulation. This could cause unreliable simulation results for the first few timesteps. Nevertheless, in a simulation with a duration of multiple seconds equaling multiple thousand timesteps, less reliable results for the first n timesteps is negligible.

Another disadvantage from having to train a bigger network is the increasing demand in memory space and exponentially growing computing effort [30]. One could suggest that the optimal configuration exceeds values of $n_y = 6$ and $n_u = 4$ but when tests are conducted with a configuration of $n_u = 5$, MATLAB runs out of memory.

For the other two output vectors, the same delay optimization procedure is followed. Different lags might be effective in the modeling of other mechanical variables. The configuration that produces the bet closed-loop test performance is chosen, resulting in optimized configurations of $n_u = 4$, $n_y = 4$ for $\boldsymbol{r}_{wc}$ and $n_u = 2$, $n_y = 5$ for $\boldsymbol{F}_{lcas}$.

4.4.3 Activation Function

The activation function (also called transfer function) used in the nodes is key to the performance of any neural network. For each layer, different activation functions can be used. However, at least in one of the layers, a non-linear activation function is needed to introduce non-linearity into the model [28]. On the other hand, non-linear activation functions cause the error surface to have many local minima [34]. Additionally, the activation function should be monotonic and differentiable to make gradient-based learning possible. Furthermore, the maximum slope should be located at the origin. S-shaped activations are the most common functions used, although there is no reason why they should be superior to others [28]. Overviews of different activation functions for various neural network applications can be found in [28, 34, 55].

For the output layer in modeling problems, a linear activation function is recommended which simply returns the value that serves as input to the function [34]:

$$y = \text{purelin}(x) = x \qquad (4\text{-}7)$$

In the implementation in MATLAB, the default activation function for the hidden layer of NARX networks is `tansig`, the hyperbolic tangent function which has a s-shape *(sigmoid)* is calculated by [34]

$$y = \text{tansig}(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad (4\text{-}8)$$

Another activation function to consider is the logistic sigmoid function, implemented in MATLAB as `logsig` [30]. Note that the function outputs values in the range of 0 to 1, therefore the normalization parameters need to be adapted as well (see 4.2.1). The logistic sigmoid function is defined as

$$y = \text{logsig}(x) = \frac{1}{1 + e^{-x}} \qquad (4\text{-}9)$$

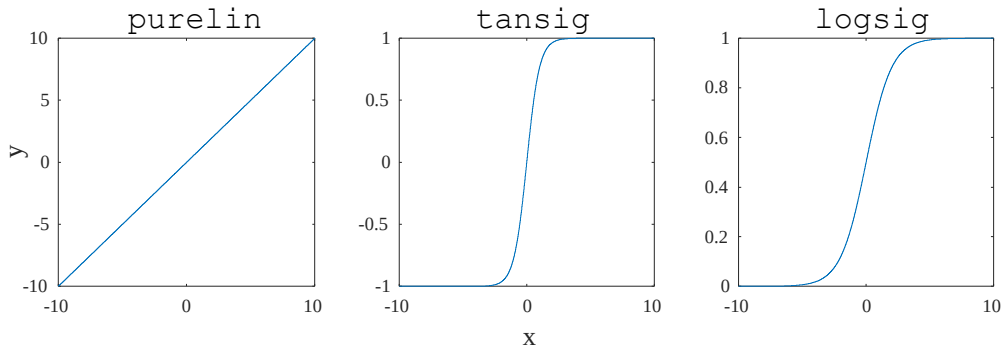


Figure 4.10 Selection of activation functions for the use in NARX networks

After training the network with each activation function 100 times, a difference in results can be detected. Networks that contain the `tansig` activation function achieve a mean nMAE of ≈ 8.59 %, a median nMAE of ≈ 2.45 % and a minimum nMAE score of ≈ 0.67 %, while networks that feature the `logsig` activation function achieved a mean nMAE of ≈ 3.67 %, a median nMAE of ≈ 2.11 % and a minimum nMAE score of ≈ 0.66 %. Since networks with the `logsig` activation function delivered the better results on all three scores, this activation function is set for the rest of the research.

4.4.4 Training Function

A detailed explanation of the function of MLBP and its adjustment to train recurrent networks by BPTT can be found in section 2.2.4. In MATLAB, the algorithm is referred to as `trainlm`. To improve generalization performance, a regularization term is often added to penalize parameter increase [34]. The learning algorithm is adapted to accommodate for the new cost

function. Known as BAYESIAN Regularization, the training function of the network in MATLAB can be set to `trainbr` [55].

To set the learning algorithm in MATLAB associated to the network object, the user has to adjust the attribute `net.trainFcn` accordingly. An overview of the usable learning algorithms can be found in [34]. Other training functions are not recommended since they are slower or intended for use on other problems [34].

Both the LEVENBERG-MARQUARDT and BAYESIAN Regularization algorithms were tested in 100 iterations each to compute the mean, median and minimum nMAE score. Networks trained with the BAYESIAN Regularization learning algorithm show superior results in all three categories compared to the LEVENBERG-MARQUARDT Algorithm: Mean nMAE (≈ 4.45 % vs. ≈ 4.92 %), median nMAE ($\approx$ 1.98 % vs. $\approx$ 2.24 %) and minimum nMAE ($\approx$ 0.65 % vs. $\approx$ 0.76 %). In conclusion, BAYESIAN Regularization will be applied as the learning algorithm.

4.5 Post-Processing

If the input and target data has been processed to improve the training as outlined in section 4.2, another process after the training is necessary to convert the output data back to fit the original target data. Additionally, some changes to the network can be made that would allow better test result. This is called *post-processing*.

After training the network and before applying new data is to the network, the user can choose to remove or add delays to the network. This is especially helpful if the optimized input and output delays were not determined beforehand, for instance during HPO. Since the ideal network delays were set in 4.4.2, no improvement for the test performance by adding or removing delays is noticed. Therefore, the addition and removal of delays is not pursued further.

To compare output and target data in their original scale, one must re-normalize the output data. As explained in section 4.2.1, the input and target data are normalized before the training and the normalization parameters are saved. Applying these parameters on the output data generated by the network, an output is derived that is comparable to the original target data [55].

5 Results

In this chapter, training results are shown and explained first. Section 5.2 covers the embedding of the model in SIMULINK. Next, the model is applied to the test maneuvers (described in detail in sections 3.1.3– 3.1.7) and the results are evaluated in section 5.3. Finally, the computational effort and the possibility of parallel computing are topic of section 5.4.

5.1 Training Results

In section 4.4, HP for the training phase were optimized. In summary, the optimized NARX network possesses 3 input delays and 4 feedback delays. The hidden layer consists of 5 nodes, that sum up their inputs and pass them through the logistic sigmoid activation function. The number of input and output nodes is defined by the dimensions of input and output vectors. Weights and biases are changed during the training phase by the BAYESIAN Regularization algorithm, a variant of MLBP that includes a term in the cost function that keeps the weights low. During the open-loop training, normal backpropagation can be applied. After closing the loop, training takes place using BPTT. The three networks are each trained on a single output vector. For each output vector, weights and biases are initialized and the network is trained during 50 iterations. The networks with the best test-performance after the closed-loop training are kept for the implementation to SIMULINK (section 5.2) and further testing (section 5.3).

The best training results for the modeling of wheel carrier acceleration, wheel carrier position and spring / damper force are listed in Table 5-1.

Table 5-1 Training results for key output variables

Output Variable	$\boldsymbol{a}_{wc}$	$\boldsymbol{F}_{lcas}$	$\boldsymbol{r}_{wc}$
Input Delays	0, 1, 2, 3	0, 1, 2	0, 1, 2, 3, 4
Feedback Delays	1, 2, 3, 4	1, 2, 3, 4, 5	1, 2, 3, 4
MSE Test OL	1.0525e-05	3.9596e-06	3.1168e-05
MSE Test CL	0.02812	0.0123	0.47704
OL Training Time (s)	190	204	227
CL Training Time (s)	1136	1081	1290
R^2	0.99069	0.99684	0.96575
nMAE x	0.0136	0.0161	0.0918
nMAE y	0.0038	0.0158	0.1216
nMAE z	0.0020	0.0156	0.0316
Overall nMAE	0.0064	0.0158	0.0817
Total Parameters	203	188	233
Effective Parameters	170	138	162

For the objective of modeling the wheel carrier acceleration $\boldsymbol{a}_{wc}$, the best network achieved an overall nMAE below 1 %. Nonetheless it is noticeable, that the acceleration in x-direction shows the largest nMAE. However, the scale of values in the time series is significantly lower than in the other two directions, so the impact on the overall MAE would be rather low. The coefficient of determination R^2 of over 0.99 is another indicator for the good quality of the model.

The reaction force of the spring damper element $\boldsymbol{F}_{lcas}$ is modeled with high accuracy as well. All the target time series for the individual vector components stay below the nMAE score of 2 % in addition to the very high R^2 value of over 0.996. Furthermore, this trained network seems to be the most parsimonious. It only uses 133 out of 188 available parameters. This might be attributed to the underlying force-to-force-relationship.

In contrast, the position of the wheel carrier can only be modeled with a significantly worse accuracy than the two aforementioned vectors. The overall nMAE score of ca. 8 % displays that the network might have reduced usability, especially when faced with new datasets.

5.2 Embedding in Simulink

It can be useful to implement the trained model in SIMULINK, e. g. to integrate the model into a full vehicle simulator. Two ways are introduced that can produce the same results.

1. Generate a *NARX simulation function* that maps inputs, lagged inputs and lagged feedbacks to outputs. This function can be implemented into a SIMULINK model as a MATLAB `Fcn`-

block. The outputs have to be delayed by unit-delays and fed back into the block. The inputs must be delayed and fed into the model as well. Weights and biases are saved in the script as constant values. The expression for the generation of the script is given by `genFunction(net,pathname)`. The function takes the network object and the pathname, for saving the file, as inputs and processes one timestep for every time it is called. As an option, the user can choose to generate a function that computes with matrices only instead of cell arrays [55].

2. Generate a NARX simulation block in SIMULINK. The model comes with an input and output block that can be replaced by simulation input and output signal connections (see Figure 5.1). The NARX block itself has a mask that allows the user to configure the initial delay states. Weights and biases are set to constants. In theory, manual changes to the network are possible, a learning algorithm however cannot be applied anymore.

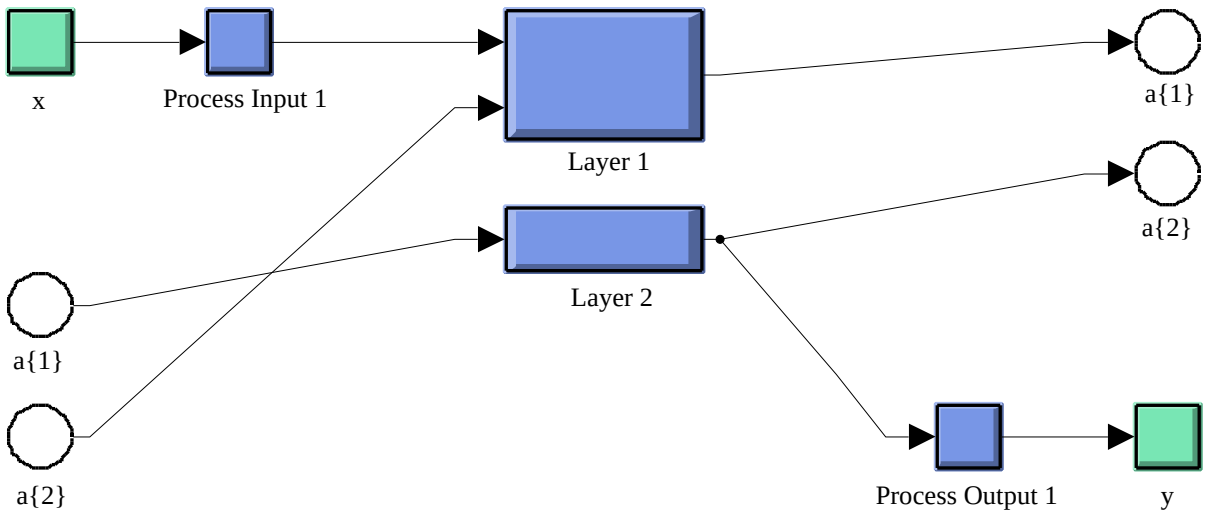


Figure 5.1 Generated SIMULINK model of closed-loop NARX network

Regardless of which method is chosen, the input and output data must be normalized for the use in NARX simulations and for the analysis afterwards. When using the SIMULINK approach, normalization blocks can be pasted in. To ensure full performance, the same normalization parameters that have been established before the training are used. For instance, these parameters can be saved to the network object in a struct-field and referred to by the normalization block's mask.

In most applications, the initial delay states are left blank (setting them to zero) for the BPTT algorithm and as application to new data [39]. In some cases, a different default value besides 0 exists. For example, the default position of the wheel carrier center or the force applied to the wheel carrier due to the weight of the vehicle may be chosen. These values must be normalized

according to the expressions listed in section 4.2.1. Nevertheless, the best-performing initial delay states are usually not known. Therefore, the error during the first few timesteps is typically increased. An assumption can be made that initial layer states that were deemed optimal for the training should also be applied when testing is performed on new data.

The velocity of the wheel carrier could be an interesting variable for a complete vehicle model. To be included in the following analysis, the velocity can be generated as differentiation of the position with regards to time or the integration of the acceleration. The generalization performance of the acceleration-trained network is significantly better than the one of the position-trained networks. Therefore, an integration block is added for the acceleration signals to obtain the velocities.

5.3 Simulation of Test Maneuvers

The simulation model that is obtained as explained in the previous section can now be loaded with new input datasets. These datasets are generated as explained in full detail in sections 3.1.2 to 3.1.7. The initial layer and delay states are initialized with the same values that have been used in the training phase. However, a significant error is noticeable during the first few timesteps. To ensure the comparability of the results, the first second in each maneuver is left out of consideration. This is possible because all maneuvers were designed to spare at least one second at the beginning of the test for the vehicle to initialize in the simulation. The nMAE score is determined as explained in section 4.4 and listed in Table 5-2.

Table 5-2 Overview of nMAE scores for all applied test maneuvers

Maneuver	$\boldsymbol{a_{wc}}$	$\boldsymbol{F_{lcas}}$	$\boldsymbol{r_{wc}}$	$\boldsymbol{v_{wc}}$
Racetrack (Training)	*0.0064*	*0.0158*	*0.0816*	-
Road Profile	0.08565089	0.01810654	0.21810155	0.48395931
Acceleration 1	0.05655148	0.01672318	1.33731197	0.41486562
Acceleration 2	0.0159581	0.00587355	0.61435123	0.45229979
Braking 1	0.04366704	0.00459517	1.29384389	0.34261304
Braking 2	0.01709268	0.00362209	0.80210613	0.38383981
Step Steer 1	0.51917437	0.00356954	0.66391799	0.38804199
Step Steer 2	0.33964987	0.00552478	0.54658001	0.38446873
Double Lane Change 30 km/h	0.00250651	0.00492791	0.53687921	0.54240962
Double Lane Change 50 km/h	0.00486352	0.00390315	0.40258607	0.25520148
Double Lane Change 70 km/h	0.08254193	0.00386442	0.36784361	0.52232828
Double Lane Change 100 km/h	0.13270338	0.00676815	0.28761674	0.88747467

Before comparing the results, it must be kept in mind that the maneuvers are not completely comparable. The different maneuvers impose different tasks on the wheel suspension. While some timeseries only feature a single peak and the diagram only shows a flat curve for the rest of the event, in other events the wheel suspension must handle constant change of forces and

torques applied to the suspension. Therefore, the complexity of the timeseries must be considered as well.

The overall results show that the acceleration of the wheel carrier can be modeled with good accuracy. In some maneuvers like the first two double lane change events, the nMAE score even stays below 1 %. Nevertheless, in some cases the network fails to deliver acceptable accuracy, for instance during the first step steer event, in which the produced output misses the target output on average by more than half of the output's range.

Displayed in the next column in Table 5-2, the nMAE score of the reaction force at the spring / damper element is displayed. In most scenarios, the value does not reach more than 1 % and even supersedes the training performance. Hence, it can be assumed that the network generalized very well for the given modeling task.

As displayed in the table above, the position of the wheel carrier and its velocity (integrated from the acceleration timeseries) could not be modeled with reasonable accuracy by the networks at hand. Therefore, these plots will not be analyzed further. However, it should be noted that generalized coordinates could be used instead of the position vector to identify the position state of the component. In a few initial tests, the generalized coordinates could be modeled with good accuracy. In theory, a new analytical model could derive positions, velocities and accelerations from generalized coordinates.

Despite the simple computation process of a single node, it is difficult for the user to reconstruct the complete neural network computation of a certain output sample. Therefore, it is practically impossible in most cases to obtain the factors of why a certain output variable was well- or badly performing against the target. Nevertheless, it is expected that neural networks handle samples better if these samples are similar to those that they were trained on.

Road Profile Event

The setup for this test-simulation is explained in detail in section 3.1.3. To summarize, the vehicle drives on an uneven road for $20\ s$. The roughness of the road resembles the roughness on the racetrack that was used for training. In other words, the same roughness parameters are used. The performance of the networks in this test could indicate how well networks are performing in test scenarios with uneven roads in general.

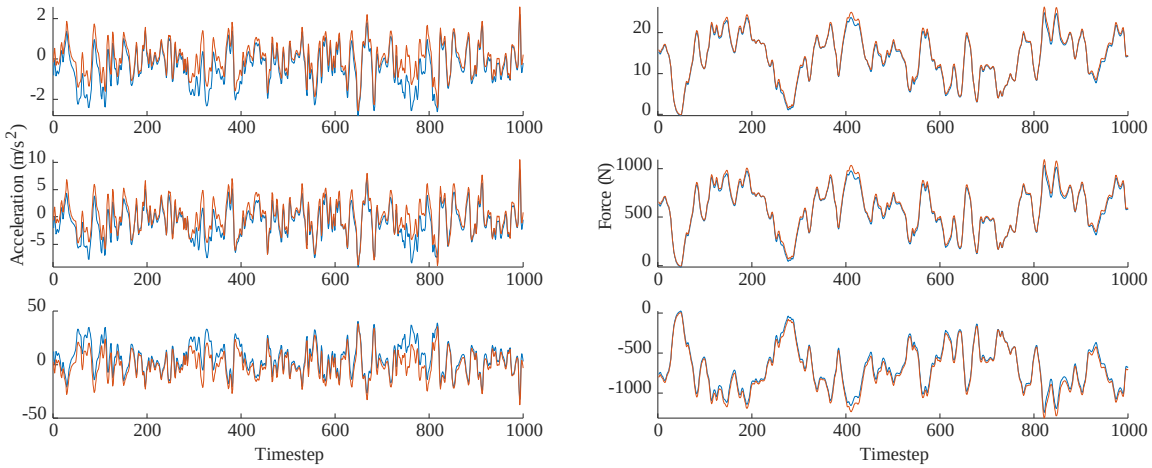


Figure 5.2 NARX results (blue) of $\boldsymbol{a}_{wc}$ (left) and $\boldsymbol{F}_{lcas}$ (right) vs. target values (orange) – Road Profile Event

Above, the timeseries of acceleration of the wheel carrier and the reaction force at the spring / damper element are depicted. For visualization purposes, only 1000 timesteps from the simulation are shown. The acceleration is estimated with acceptable accuracy (nMAE $<$ 10 %). The course of the output timeseries follows the course of the target timeseries closely, with exception of some peaks. The force of the spring / damper element is reproduced with very high accuracy (nMAE $<$ 2 %) throughout the duration of the test. The position of wheel carrier can only be modeled with high deviations (nMAE $>$ 20 %) and its velocity, integrated from the acceleration time series, failed to match the target time series as well (nMAE $>$ 40 %).

Acceleration Events

The acceleration test simulations are introduced in section 3.1.4. Two separate events are described. The first acceleration event tries to replicate an acceleration situation from the racetrack simulation.

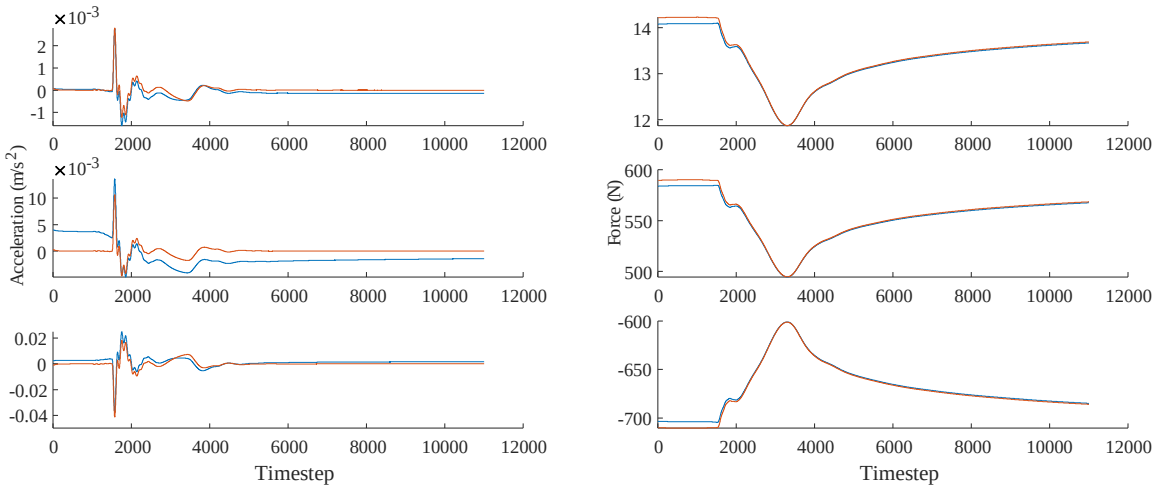


Figure 5.3 NARX results (blue) of $\boldsymbol{a}_{wc}$ (left) and $\boldsymbol{F}_{lcas}$ (right) vs. target values (orange) – Acceleration Event 1

As displayed in Figure 5.3, the acceleration of the wheel carrier and the force at the spring / damper element are modeled with good accuracy. The acceleration timeseries, especially the $y-$component, show a small offset for most of the duration of the first acceleration maneuver. This is also reflected in the nMAE value which is ca. 5.6 % for this maneuver. The nMAE for the reaction force at spring / damper is kept below 2 %, despite its initial offset until the acceleration commences.

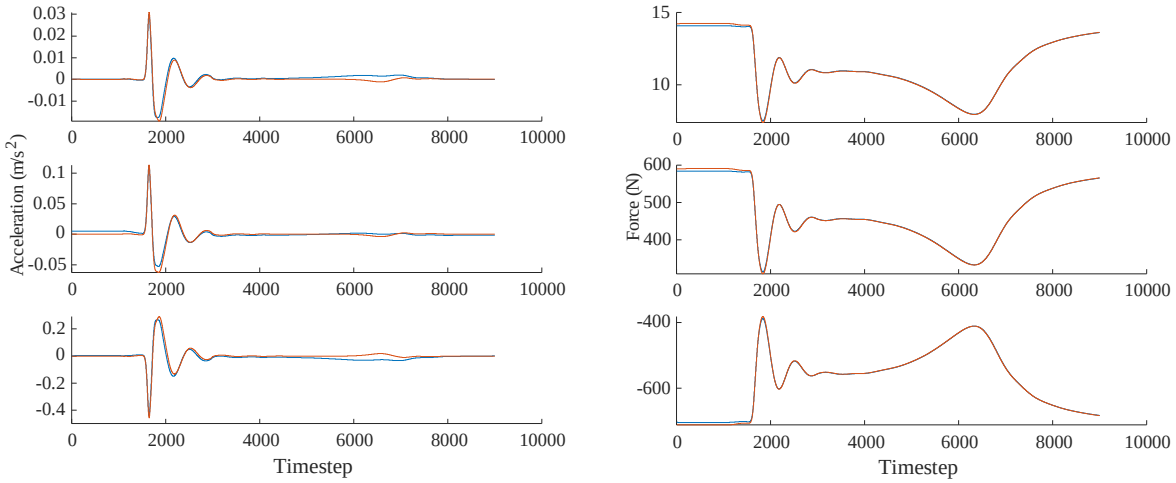


Figure 5.4 NARX results (blue) of $\boldsymbol{a}_{wc}$ (left) and $\boldsymbol{F}_{lcas}$ (right) vs. target values (orange) – Acceleration Event 2

The second acceleration maneuver consists of the vehicle accelerating with full throttle. Good estimations for the target values of $\boldsymbol{a}_{wc}$ (nMAE $\approx$ 1.6 %) and $\boldsymbol{F}_{lcas}$ (nMAE $\approx$ 0.6 %) are made by the networks. Surprisingly, the timeseries from this event are modeled with better accuracy than the timeseries from acceleration event 1. The networks are expected to perform

better on the first event since it resembled similar data from the training dataset. In addition, the vehicle is brought to its physical limits during the second event, implying that vehicle dynamics could deviate from those in standard scenarios.

Braking Events

The two braking events are described in section 3.1.5. The first braking event tries to mimic a braking situation that can be found in the training dataset. The simulation stops, as soon as the vehicle's velocity drops below a pre-defined threshold. In contrast to the acceleration maneuver, the information about the braking action is contained in the input dataset directly since the rolling resistance torque is affected by the activation of the brake.

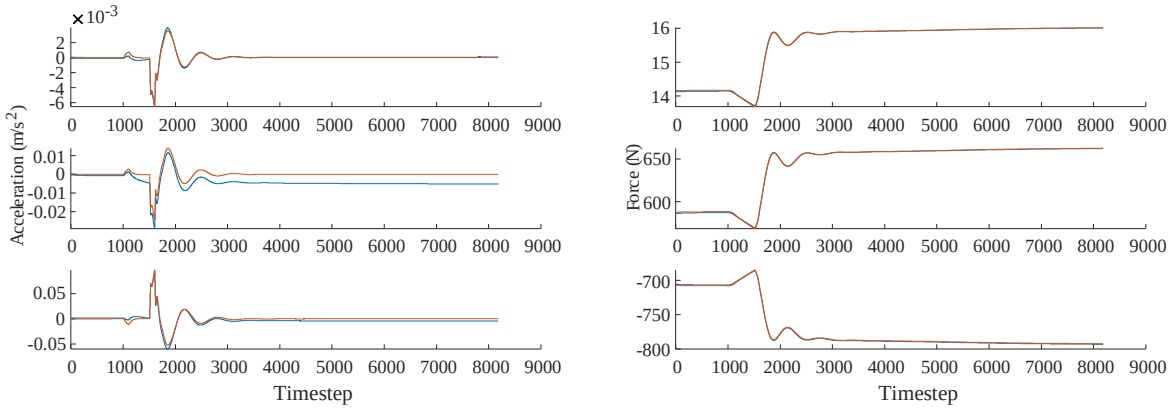


Figure 5.5 NARX results (blue) of $\boldsymbol{a}_{wc}$ (left) and $\boldsymbol{F}_{lcas}$ (right) vs. target values (orange) – Braking Event 1

The NARX networks used for the modeling of acceleration of the wheel carrier and the reaction force at the spring damper element can handle the input data from the first braking maneuver well. Only the y-component of the acceleration vector is offset after the braking commences. The modeling of the force is achieved with almost full accuracy.

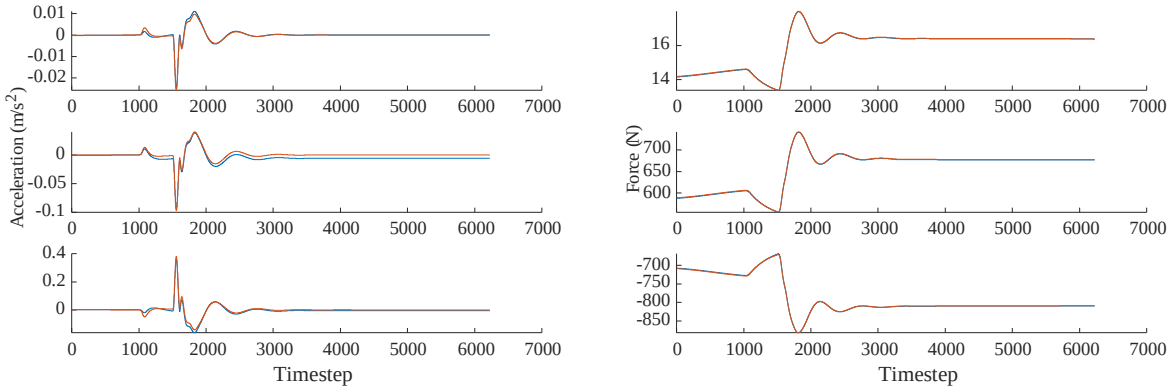


Figure 5.6 NARX results (blue) of $\boldsymbol{a}_{wc}$ (left) and $\boldsymbol{F}_{lcas}$ (right) vs. target values (orange) – Braking Event 2

A similar trend to the acceleration maneuvers is noted. The second test that uses the higher acceleration (in this case: deceleration) values can be modeled with improved accuracy. This is true for both $\boldsymbol{a}_{wc}$ and $\boldsymbol{F}_{lcas}$.

Step Steer Event

In the simulation of a step steering event, the vehicle must handle an abrupt steering change. After this change, the vehicle keeps steering, therefore taking a turn of a constant radius. The most sudden change of the steering wheel angle that can be simulated takes just a single timestep, 0.001 s.

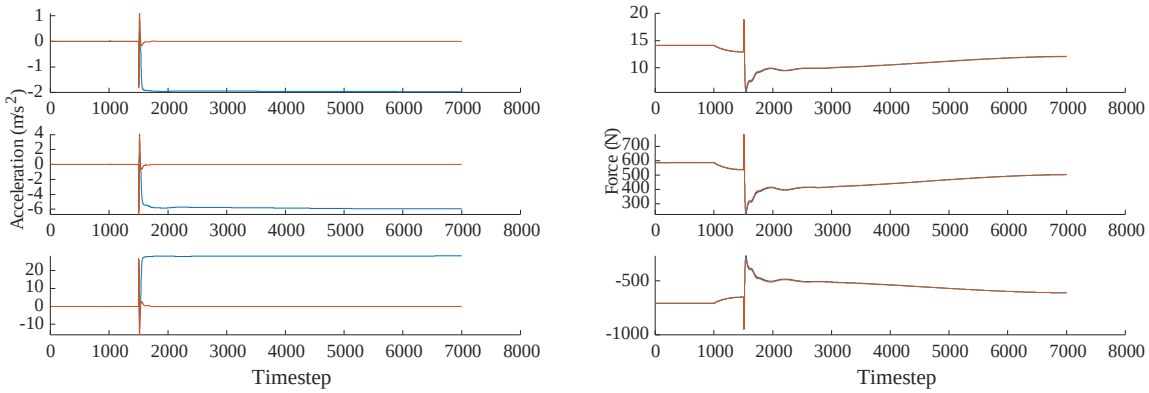


Figure 5.7 NARX results (blue) of $\boldsymbol{a}_{wc}$ (left) and $\boldsymbol{F}_{lcas}$ (right) vs. target values (orange) – Step Steer Event 1

The first step steering event emulates a turn from the training dataset, configured at 20° of steering wheel angle and 30 km/h constant speed. From the course of the output vs. the target data above in Figure 5.7, it is noticeable that the reaction force at the spring / damper element is modeled with high accuracy. The network trained on acceleration however produces a large

offset from the target data after the steering step. As expected, this can be monitored in the nMAE score, which displays a value of more than 50 %.

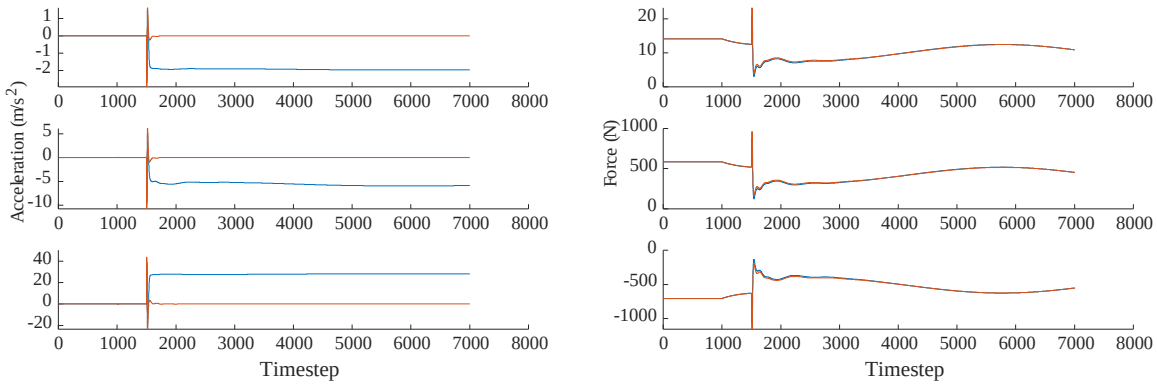


Figure 5.8 NARX results (blue) of $\boldsymbol{a}_{wc}$ (left) and $\boldsymbol{F}_{lcas}$ (right) vs. target values (orange) – Step Steer Event 2

For the second step steering event, the steering wheel angle and the constant speed were increased to create a new maneuver with no precedence in the training dataset. Similar to the previous plots, the reaction force is modeled almost flawlessly while the acceleration outputs produce large offsets after the steering change. The acceleration nMAE score measures less, but still over 30 %, which can be accounted to smaller offset after the steering angle change.

Double Lane Change

The double lane change is a standard ISO test maneuver [43] used to analyze the lateral vehicle dynamics. The vehicle has to merge into a parallel lane and merge back into the original lane again. A more detailed explanation is given in section 3.1.7. Since the lateral vehicle dynamics are managed by controller in ADAMS CAR, the constant velocity is the only parameter that can be altered to change the suspension behavior. The network's performances are analyzed in regards of 4 different velocities: $30, 50, 70$ and 100 km/h. For the latter two velocities, the vehicle failed to stay within the course. The two most extreme cases, at 30 and 100 km/h shall be visualized here.

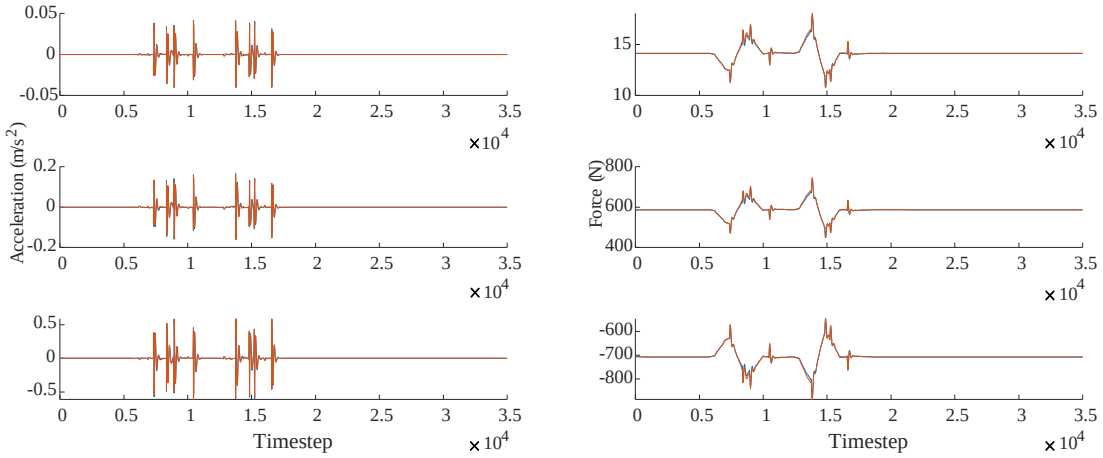


Figure 5.9 NARX results (blue) of $\boldsymbol{a}_{wc}$ (left) and $\boldsymbol{F}_{lcas}$ (right) vs. target values (orange) – Double Lange Change 30 km/h

The simulation results of the object-oriented and the neural network model are depicted above. Clearly, the acceleration and the spring / damper reaction force is modeled with very good accuracy, resulting in nMAE scores of 0.25 % and 0.49 %. Similarly, the data network's output from the input data of the second event follows the course of the target timeseries very closely, producing nMAE scores of 0.48 % and 0.39 % for acceleration and the spring / damper reaction force.

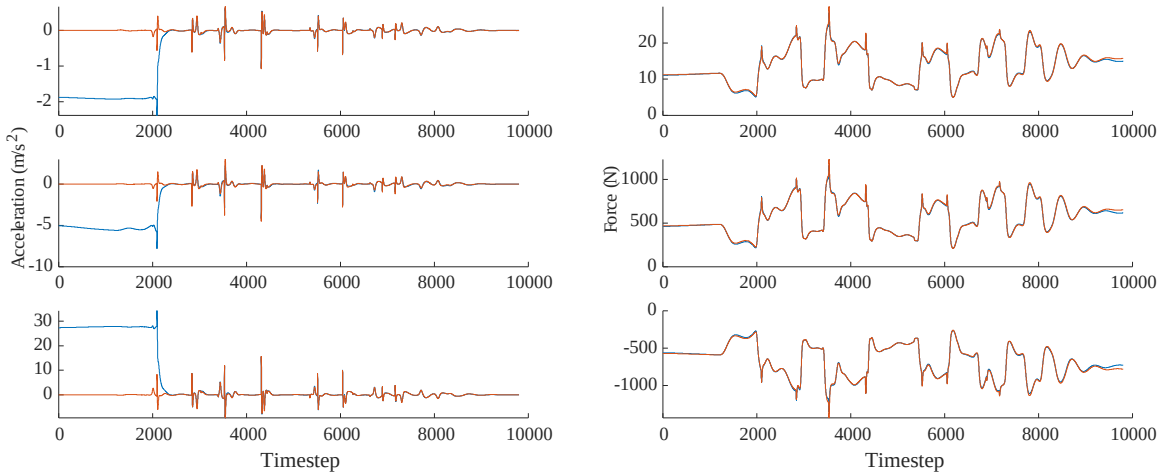


Figure 5.10 NARX results (blue) of $\boldsymbol{a}_{wc}$ (left) and $\boldsymbol{F}_{lcas}$ (right) vs. target values (orange) - Double Lane Change 100 km/h

Recall that during the third and fourth event, the vehicle failed to stay on course and the tires even lost grip to the road. As can be seen above in Figure 5.10 for the double lane change at 100 km/h constant speed, the output acceleration starts shifted from the target timeseries, but merges back with it after the vehicle starts steering. The spring damper force is modeled with

negligible errors. Likewise, the outputs of the double lane change at 70 km/h are modeled, differentiated only by a smaller offset at the start of the acceleration timeseries.

5.4 Computational Effort and Parallel Computing

One must be careful about the increase of network sizes when computational resources for the training is limited. When performing LEVENBERG-MARQUARDT Backpropagation, the computation time rises exponentially when parameters are increased [34]. In a NARX network, increasing the hidden nodes N, the input or feedback delays n_u, n_y or the dimension of input N_u or output N_y accounts to multiple extra parameters

$$p_{NARX} = \underbrace{\left(N_u * (n_u + 1) + N_y * n_y\right) * N}_{w \text{ input to hidden layer}} + \underbrace{N_y * N}_{w \text{ hidden to output layer}} + \underbrace{N_y + N}_{\text{bias}} \quad (5\text{-}1)$$

that the network must optimize [27, 55]. The combinatorial growth of parameters when input or output dimensions are increased leads to an explosion in computational effort. This is considered a major drawback for the use of NARX networks [54]. For the application at hand, most parameters are made up of the weights between the input- and hidden layer. As a result of using only 5 hidden nodes, the number of biases is kept low. Refer to Table 5-1 for an overview of the total number of parameters per use case.

The parallel structure of neural networks suits them for the use of parallel computing [30], however, this can only be exercised in feed-forward networks (or open-loop NARX networks) since BPTT algorithms cannot be computed in parallel [55]. However, the results from section 5.1 show that closed-loop training is essential for the generalization capabilities. When the procedure outlined in chapter 4 is followed and the computational effort is analyzed by the MATLAB profiler, it shows that 83.9 % of the computation time was spent on training the closed-loop network, 15.9 % on training the open-loop network and the remaining time on pre-processing, testing and configurations.

Often used in training deep neural networks, training on GPUs can accelerate the training process. However, GPUs cannot support JACOBIAN-based training. Also, training with `tansig` or `logsig` activations function might be problematic since the exponential function is often not implemented with the hardware [34, 55].

6 Discussion

In the previous sections and chapters, the suitability of NN for the use of modeling entire suspension systems was explored. The training results are not completely satisfactory but show that the reaction force could be modeled with the highest accuracy out of the 4 examined timeseries, followed by the acceleration of the wheel carrier which performed well in most scenarios.

After implementing the networks into the SIMULINK environment, the test maneuvers were simulated. The findings from the training results could mostly be confirmed. The neural networks that were trained on acceleration of the wheel carrier and the spring / damper reaction force delivered results close to the target timeseries. However, during the step steering event and the double lane changes at 70 and 100 km/h constant speed, the resulting output timeseries was shifted from the target timeseries for a section of the event, producing a tremendously larger error than found in other events. Therefore, the modeling of the reaction force at the spring / damper and wheel carrier can be deemed successful, though the acceleration-trained network must be used with caution. Assuming shifted output sections are the only problem of the acceleration network, a simple post-processing algorithm could detect those faults by checking whether the acceleration timeseries stays at a constant value other than 0 for a section of the event.

An explanation for this phenomenon cannot easily be investigated due to the black-box representation of the neural network. It can only be assumed that the network had problems handling the sudden change of steering and the critical driving situations, in which abrupt and unseen forces and torques are applied to the suspension system.

Another network tried to model the position of the wheel carrier. Additionally, the velocity was integrated from the acceleration of the target and output timeseries and compared. Both output vectors could not be modeled with acceptable accuracies. Again, the black-box model does not allow to simply identify the problem of why the modeling of the position failed. One could argue that the relationship of applied forces and torques to position is vastly more complex than the mentioned modeling tasks regarding acceleration and reaction force. Modeling the velocity by integrating the acceleration only becomes a valid option when the acceleration is modeled at good accuracy. Even then, the problem of the integration constant is left unaddressed.

The problem of poor position modeling might be fixable by training networks on generalized coordinates. If the isolated DoF is left out of consideration, only two generalized coordinates (minimum coordinates) need to be modeled to fully determine the positions of the suspension components. However, another model would have to be developed to determine these positions. Alternatively, the generalized coordinates of all components can be modeled independently.

Still, a physical model would need to be developed that translates generalized coordinates into cartesian coordinates.

Many analytic solutions to multibody systems utilize D'ALEMBERT's principle [24], in which reaction forces are allowed to be omitted from the method of computation. The reaction forces are otherwise difficult to obtain. With the presented neural network, it is possible to calculate the reaction force at the spring / damper element efficiently without a major loss of accuracy.

Although the modeling of other reaction forces and accelerations were not explicitly examined in this thesis, initial tests have shown that well-performing reaction force- and acceleration modeling is likely possible for other suspension components. Based on this assumption, it is possible to model all the signals used for interfaces between components in the object oriented model (compare figures in Section 3.2.2). Furthermore, the forces and accelerations used as interfaces to the full-vehicle model would be covered. This allows further applications like the monitoring or prediction of abrasion inside of the joints.

In the previous section, the computational effort for training recurrent NARX networks was explained. However, once neural networks are trained, the computation of output does not require complex computations since the output is basically calculated as multiple summations of simple linear and non-linear functions. For instance, the calculation of all three key output vectors in a simulation with a duration of 10 seconds is completed after less than one second. In contrast, the DAE-based object-oriented model needs to run numerical approximation method for every timestep.

7 Summary and Outlook

In this chapter, approaches and results from the previous chapters are summarized and suggestions for future work are made.

7.1 Summary

This thesis examined the feasibility of training neural networks that can resolve the issue of computationally intensive DAE-based simulations of wheel suspensions.

In chapter 2, the basics of the double wishbone suspension was presented along with its kinematics. In addition, the simulations and their numeric results - timeseries - were explained in the context of model-based vehicle dynamics. Afterwards, an introduction to neural networks was made. A focus was set on recurrent neural networks and their learning abilities in terms of dynamical timeseries modeling problems. It was shown how a single node computes outputs and how they are incorporated into a neural network, covering the input to output relationship and how learning algorithms like the standard BP in the shape of the LMBP or its adaption to RNNs, the BPTT are applied to network. Furthermore, the concept of generalization and ways to achieve it were explored.

At the example of the double wishbone suspension of the FORMULA STUDENT racecar, multiple ADAMS CAR full vehicle simulations were carried out in Chapter 3 to generate datasets that served as training and testing inputs for the neural network. Herein, the thought of training the network model on a wide variety of driving situations and testing it on specific maneuvers was followed. The target values that are essential to supervised neural network learning were then determined by the object-oriented analytical model of the double wishbone wheel suspension. Finally, 3 output vectors wheel carrier acceleration $\boldsymbol{a}_{wc}$, wheel carrier position $\boldsymbol{r}_{wc}$ and spring damper reaction force $\boldsymbol{F}_{lcas}$ were chosen, originating from the object-oriented model. These vectors are of interest regarding the future application of the neural network, e. g. as a component of a full-vehicle model.

In the beginning of Chapter 4, the reasoning behind the choice of the NARX model was outlined. It had proven to be a well-performing recurrent neural network when used on dynamic timeseries modeling and features lagged inputs and lagged outputs that are fed back into the network. The NN implementation in MATLAB was outlined, introducing the NARX model and its two configurations. Before training and target data were ready to be presented to the network, pre-processing had to be applied. This included the division of the datasets into training, validation and testing shares. Before the final training of the network commenced, a brief hyperparameter optimization regarding the hidden layer size, delay size, activation function and learning algorithm was executed. The best performing configuration found contained a NARX network with 5 nodes using the logistic sigmoid activation function in the hidden layer that is

trained by the BAYESIAN Regularization Algorithm. The optimized configuration of the input and feedback delays largely depends on the target vector and resulted in the combinations $n_u = 3, n_y = 4$ for the acceleration-trained network, $n_u = 2, n_y = 5$ for the reaction-force-trained network and $n_u = 4, n_y = 4$ for the position-trained network.

In chapter 5, the results are presented. The network that produced the best test performance was kept after 50 iterations, each trained for 100 epochs. The coefficients of determination R^2 reached 0.99069 for the wheel carrier acceleration, 0.99684 for the spring / damper reaction force and 0.96575 for the wheel carrier position which show that all of the modeled outputs can be approximated, some even very accurately. The normalized mean absolute errors of acceleration and reaction force are promising as well with values of ca. 0.64 %, 1.58 %. In contrast, the position vector was offset from the target timeseries by a mean of 8.16 %.

Since a recurrent neural network utilizes past timesteps to make estimations of the current output, the delay states must be initialized in every simulation. Contrary to the training, target values are not available. The initial delay states therefore must be guessed. In this thesis, the same values from the training set were used. Nevertheless, the first few timesteps show major deviations from the target series. If this is accounted to in a future use of the network, e. g. by allowing the simulation to initialize first before starting the maneuver, it would yield better performances.

Testing the network in various driving situations proved that the reaction force at the spring / damper element can be modeled by the neural network in all examined driving circumstances. Likewise, the acceleration output was in general close to the target timeseries. However, during some maneuvers, shifted values were produced by the network for a section of the event. This was especially peculiar after the steering angle change in the step steering event and at the start of the higher-velocity double lane change maneuvers. Due to the NN being a black-box model, it becomes intricate to find an explanation for these phenomena and was thus not pursued further. In conclusion, the modeling of both vectors was successful, nevertheless the acceleration-trained network must be used cautionary for simulation purposes – especially if the event contains critical or unusual driving situations.

In contrast, the network model compiled to simulate the wheel carrier position is unfit for most applications since the produced errors were too high. The velocity timeseries created from the integration of the acceleration fits target curve only in some cases. It often drifts away from the target timeseries and is therefore neither recommended for use.

The scope of the research was limited to a single suspension archetype. Similarly, only a single neural network structure was trialed in detail after literature-backed initial tests with other archetypes produced worse results. Acknowledging the wide variety of suspension types and neural network structures existing in science and industry, a broader analysis would go beyond

the scope of the limited research. Nevertheless, with the acquired training and testing method, other suspensions or mechanical systems can be tested, if the underlying object-oriented model and NN hyperparameters are changed accordingly.

The spring / damper reaction force generated by the neural network is suited for an application as part of a full vehicle model due to its high accuracy. The modeling of wheel carrier acceleration, also showing high accuracy, requires postprocessing or at least a plausibility check. Position and velocity outputs as modeled in this thesis are not suited, due to their high errors. The computationally intensive training of the network is completed before the simulation commences. Outputs from the neural network are generated by simple computation that does not require approximating differential equations or other time-consuming tasks, ultimately providing real-time capable outputs with acceptable errors.

7.2 Outlook

With the developed procedure, it is possible to train NARX on other underlying systems. Hereby, an overall assessment on the suitability of NARX networks for suspension systems could be made.

As this thesis only trialed NARX networks after showing promising results early on, other network architectures should be investigated further as well. In conjunction with NARX networks, radial basis function networks or time delay network – the generalization of NARX networks - can be explored. Recurrent networks can also be implemented in deep learning structures, though requiring more computational resources. Additionally, other types of networks or machine learning algorithms might be viable when the focus is set on other output variables.

More effort could have been spent on an exhaustive grid search or other HPO algorithms that would optimize the trained networks. However, this requires time and effort dedicated to another field of research, that might only improve the performances by marginal percentage. Likewise, it is possible to train networks on a single vector component, possibly creating more focused networks. Similarly, pre-processing can be expanded. The success of the networks is utterly dependent on the data it was presented with. When handling uneven roads, focus could be set on the frequency analysis of the input data. In addition, more a priori relationships can possibly be detected and used in pre-processing methods.

Finally, as indicated in section 4.1, NARX models can be translated into polynomial systems with the help of pruning methods [4] which can be considered to reduce computational effort and to make the model more transparent [60].

8 References

[1] D. Schramm, M. Hiller, and R. Bardini, *Vehicle Dynamics: Modeling and Simulation*. Berlin, Heidelberg, s.l.: Springer Berlin Heidelberg, 2014. [Online]. Available: http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=815343

[2] F. Kracht, S. Michael, and D. Schramm, *Real-Time Calculation of Reaction Forces and Elasticities in Vehicle Wheel Suspensions*, 2018.

[3] D. E. Rumelhart, B. Widrow, and M. A. Lehr, "The basic ideas in neural networks," *Commun. ACM*, vol. 37, no. 3, pp. 87–92, 1994, doi: 10.1145/175247.175256.

[4] S. A. Billings, *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. Chichester, West Sussex: John Wiley & Sons, 2013. [Online]. Available: http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=609311

[5] F. E. Kracht, *Modellbildung und Simulation der Dynamik und Elastokinematik von Radaufhängungen für Echtzeitanwendungen*. Duisburg, Essen: Universität Duisburg-Essen, 2020.

[6] M. Türpe, *Modellierung des Rennwagens des E-Teams der Universität Duisburg- Essen als Mehrkörpersystem für Fahrdynamik Simulationen*. Final Thesis, University Duisburg-Essen, Duisburg, Germany, 2015.

[7] A. Aziz, *Modellierung des Rennwagens des Formula Student Electric Teams der Universität Duisburg-Essen als Mehrkörpersystem in ADAMS/Car als Referenz für Fahrdynamiksimulationen*. Final Thesis, University Duisburg-Essen, Duisburg, Germany, 2016.

[8] S. Dreier, *Validation of the Multibody Simulation Model of the Formula Student Racecar A40-02 of the E-Team of the University of Duisburg-Essen by Use of Measured Data of Standardized Suspension Tests*. Final Thesis, University Duisburg-Essen, Duisburg, Germany, 2015.

[9] J. Kacprzyk and D. Prokhorov, *Computational Intelligence in Automotive Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[10] Y. U. Yim and S.-Y. Oh, "Modeling of Vehicle Dynamics From Real Vehicle Measurements Using a Neural Network With Two-Stage Hybrid Learning for Accurate Long-Term Prediction," *IEEE Trans. Veh. Technol.*, vol. 53, no. 4, pp. 1076–1084, 2004, doi: 10.1109/TVT.2004.830145.

[11] O. Alexa, C. O. Ilie, R. Vilău, M. Marinescu, and M. Truta, "Using Neural Networks to Modeling Vehicle Dynamics," *AMM*, vol. 659, pp. 133–138, 2014, doi: 10.4028/www.scientific.net/AMM.659.133.

[12] J. Dye and H. Lankarani, “Hybrid Simulation of a Dynamic Multibody Vehicle Suspension System Using Neural Network Modeling Fit of Tire Data,” in *ASME 2016 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference (IDETC/CIE2016): Volume 6: 12th International Conference on Multibody Systems, Nonlinear Dynamics, and Control*, Charlotte, North Carolina, USA, 2016.

[13] Z. Ding, F. Zhao, Y. Qin, and C. Tan, “Adaptive neural network control for semi-active vehicle suspensions,” *J VIBROENG*, vol. 19, no. 4, pp. 2654–2669, 2017, doi: 10.21595/jve.2017.18045.

[14] J. J. Zhang, B. an Han, and R. Z. Gao, “Multi-Body Model Identification of Vehicle Semi-Active Suspension Based on Genetic Neural Network,” *AMM*, 121-126, pp. 4069–4073, 2011, doi: 10.4028/www.scientific.net/AMM.121-126.4069.

[15] A. Gustafsson and A. Sjörgren, *Neural network controller for semi-active suspension systems with road preview: Master’s thesis.* [Online]. Available: https://odr.chalmers.se/bitstream/20.500.12380/257298/1/257298.pdf (accessed: Jun. 19 2020).

[16] M. N. Alghafir and J. F. Dunne, “A NARX damper model for virtual tuning of automotive suspension systems with high-frequency loading,” *Vehicle System Dynamics*, vol. 50, no. 2, pp. 167–197, 2012, doi: 10.1080/00423114.2011.575946.

[17] A. M. Bash, “Predict Dynamic Response of Suspension Arm Based on Artificial Neural Network Technique,” *J. of Applied Sciences*, vol. 11, no. 6, pp. 988–995, 2011, doi: 10.3923/jas.2011.988.995.

[18] D. Ali and S. Frimpong, “Machine Learning Models for Suspension System Performance Prediction in Large Dump Trucks,” in 2019.

[19] C. Morgan and S. Yao, “Neural Network Model of a Vehicle Active Suspension System,” *IFAC Proceedings Volumes*, vol. 31, no. 14, pp. 129–134, 1998, doi: 10.1016/S1474-6670(17)44884-1.

[20] P. Guarneri, G. Rocca, and M. Gobbi, “A neural-network-based model for the dynamic simulation of the tire/suspension system while traversing road irregularities,” *IEEE transactions on neural networks*, vol. 19, no. 9, pp. 1549–1563, 2008, doi: 10.1109/TNN.2008.2000806.

[21] Y. Salehinia, S. Salehinia, F. Najafi, S. H. Sadati, and M. Shiee, “Solving Forward Kinematics Problem of Stewart Robot Using Soft Computing,” 2013.

[22] B. Heißing, M. Ersoy, and S. Gies, *Fahrwerkhandbuch*. Wiesbaden: Vieweg+Teubner, 2011.

[23] S. Chepkasov, G. Markin, and A. Akulova, “Suspension Kinematics Study of the “Formula SAE” Sports Car,” *Procedia Engineering*, vol. 150, pp. 1280–1286, 2016, doi: 10.1016/j.proeng.2016.07.288.

[24] W. Schiehlen and P. Eberhard, *Technische Dynamik*. Wiesbaden: Springer Fachmedien Wiesbaden, 2017.

[25] D. Schramm, M. Hiller, and R. Bardini, *Modellbildung und Simulation der Dynamik von Kraftfahrzeugen*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2018.

[26] G. Dorffner, “Neural Networks for Time Series Processing,” 1996.

[27] E. Diaconescu, “The use of NARX neural networks to predict chaotic time series,” *Wseas Transactions on computer research*, vol. 3, no. 3, pp. 182–191, 2008.

[28] D. P. Mandic, J. A. Chambers, and S. Haykin, *Recurrent Neural Networks for Prediction*. Chichester, UK: John Wiley & Sons, Ltd, 2001.

[29] K.-L. Du and M. N. S. Swamy, *Neural Networks and Statistical Learning*. London: Springer London, 2019.

[30] M. T. Hagan, H. B. Demuth, and M. H. Beale, *Neural network design*, 1st ed. Boston: PWS Publ, 1996.

[31] J.P.F. Sum, W.-K. Kan, and G. H. Young, “A Note on the Equivalence of NARX and RNN,” *NCA*, vol. 8, no. 1, pp. 33–39, 1999, doi: 10.1007/s005210050005.

[32] J. Elman, “Finding structure in time,” *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990, doi: 10.1016/0364-0213(90)90002-E.

[33] J. T. Connor, R. D. Martin, and L. E. Atlas, “Recurrent neural networks and robust time series prediction,” *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 240–254, 1994, doi: 10.1109/72.279188.

[34] M. Hudson Beale, M. T. Hagan, and H. B. Demuth, *Deep Learning Toolbox™ User's Guide*, 2020.

[35] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis*: Wiley, 2008.

[36] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. Cambridge, Massachusetts, London, England: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org/

[37] S. L. Kukreja, H. L. Galiana, and R. E. Kearney, “NARMAX representation and identification of ankle dynamics,” *IEEE transactions on bio-medical engineering*, vol. 50, no. 1, pp. 70–81, 2003, doi: 10.1109/TBME.2002.803507.

[38] R. Eini and S. Abdelwahed, “Learning-based Model Predictive Control for Smart Building Thermal Management,” in *IEEE HONET-ICT 2019: IEEE 16th International*

Conference on Smart Cities: Improving Quality of Life using ICT, IoT and AI : UNC Charlotte, NC, October 06-09, 2019, Charlotte, NC, USA, 2019, pp. 38–42.

[39] P. J. Werbos, “Backpropagation through time: what it does and how to do it,” *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990, doi: 10.1109/5.58337.

[40] M. T. Hagan and M. B. Menhaj, “Training feedforward networks with the Marquardt algorithm,” *IEEE transactions on neural networks*, vol. 5, no. 6, pp. 989–993, 1994, doi: 10.1109/72.329697.

[41] R. J. Williams and D. Zipser, “Gradient-based learning algorithms for recurrent networks and their computational complexity,” in *Backpropagation: theory, architectures, and applications*, 1995, pp. 433–486.

[42] MSC Software, *Using Adams Car*. [Online]. Available: https://simcompanion.mscsoftware.com/resources/sites/MSC/content/ (accessed: Apr. 16 2020).

[43] ISO, *ISO 3888-1 Passenger cars — Test track for a severe lane-change manoeuvre — Part 1: Double lange-change: International Organization for Standardization*, 2018.

[44] ISO, *ISO 8608:2016-11 Mechanical Vibration - Road Surface Profiles - Reporting of Measured Data.: International Organization for Standardization*, 2016.

[45] P. Múčka, “Simulated Road Profiles According to ISO 8608 in Vibration Analysis,” *J. Test. Eval.*, vol. 46, no. 1, p. 20160265, 2018, doi: 10.1520/JTE20160265.

[46] MSC Software, “Using Adams Car Ride,” 2015. [Online]. Available: https://www.simcompanion.mscsoftware.com/infocenter/

[47] M. W. Sayers, *DYNAMIC TERRAIN INPUTS TO PREDICT STRUCTURAL INTEGRITY OF GROUND VEHICLES*, 1988. [Online]. Available: https://books.google.de/books?id=6LiBtyjcoxgC

[48] T. Gillespie, “Effects of heavy-vehicle characteristics on pavement response and performance,” 2006.

[49] J. Detlefsen and U. Siart, *Grundlagen der Hochfrequenztechnik*. München: Oldenbourg Wissenschaftsverlag Verlag, 2012.

[50] R. Featherstone, *Rigid Body Dynamics Algorithms*. Boston, MA: Springer US, 2008.

[51] T. Lin, B. G. Horne, P. Tino, and C. L. Giles, “Learning long-term dependencies in NARX recurrent neural networks,” *IEEE transactions on neural networks*, vol. 7, no. 6, pp. 1329–1338, 1996, doi: 10.1109/72.548162.

[52] H. T. Siegelmann, B. G. Horne, and C. L. Giles, “Computational capabilities of recurrent NARX neural networks,” *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, vol. 27, no. 2, pp. 208–215, 1997, doi: 10.1109/3477.558801.

[53] A. Patel and J. F. Dunne, “NARX Neural Network Modelling of Hydraulic Suspension Dampers for Steady-state and Variable Temperature Operation,” *Vehicle System Dynamics*, vol. 40, no. 5, pp. 285–328, 2003, doi: 10.1076/vesd.40.5.285.17911.

[54] J. Schoukens and L. Ljung, “Nonlinear System Identification: A User-Oriented Road Map,” *IEEE Control Systems Magazine*, vol. 39, no. 6, pp. 28–99, 2019.

[55] Mathworks, *MATLAB Documentation.* [Online]. Available: https://www.mathworks.com /help/matlab/ (accessed: May 21 2020).

[56] R. Khosla, R. J. Howlett, and L. C. Jain, Eds., *Knowledge-based intelligent information and engineering systems: 9th International Conference, KES 2005 : Melbourne, Australia, September 14-16, 2005 : proceedings*. Berlin: Springer, op. 2005.

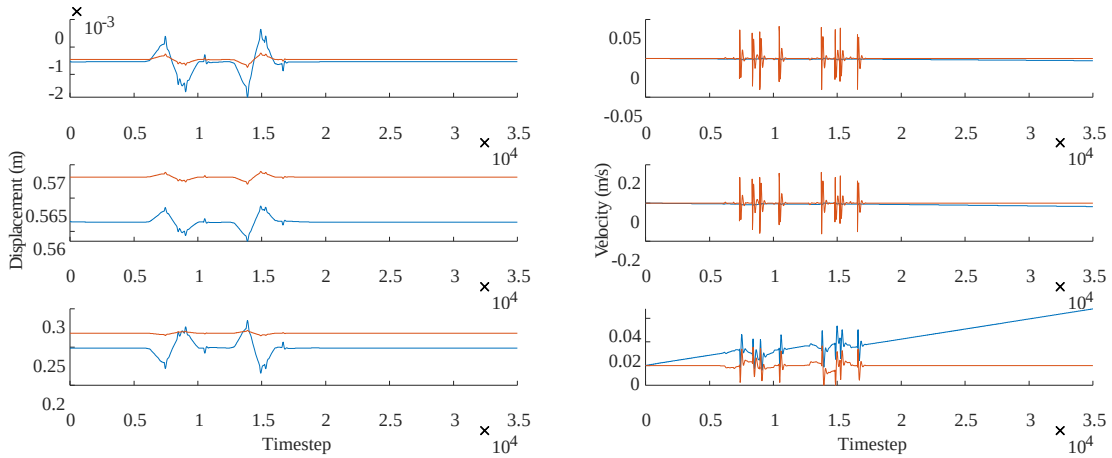
[57] M. Feurer and F. Hutter, “Hyperparameter Optimization,” in *The Springer Series on Challenges in Machine Learning, Automated machine learning: Methods, systems, challenges*, F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., Cham: Springer, 2019, pp. 3–33.

[58] T. Yu and H. Zhu, “Hyper-Parameter Optimization: A Review of Algorithms and Applications,” 2020. [Online]. Available: https://arxiv.org/pdf/2003.05689

[59] D.-G. Chen, Y. Lio, H. K. T. Ng, and T.-R. Tsai, *Statistical Modeling for Degradation Data*. Singapore: Springer Singapore, 2017. [Online]. Available: https:// pdfs.semanticscholar.org/abc7/f2ab42c7366ae3aa5f0b5e42aa924d531213.pdf

[60] D. T. Westwick, G. Hollander, K. Karami, and J. Schoukens, “Using Decoupling Methods to Reduce Polynomial NARX Models,” *IFAC-PapersOnLine*, vol. 51, no. 15, pp. 796–801, 2018, doi: 10.1016/j.ifacol.2018.09.133.

Appendix



Appendix 1 NARX Results (orange) of $\boldsymbol{r}_{wc}$ (left) and $\boldsymbol{v}_{wc}$ (right) vs target values (blue) – Double Lange Change 30 km/h

Eidesstattliche Versicherung

Name: *Seeger* Vorname: *Jan*

Matrikel-Nr.: Studiengang: *M.Sc. Wirtschaftsingenieurwesen*

Hiermit versichere ich, Jan Seeger, an Eides statt, dass ich die vorliegende Masterarbeit mit dem Titel *„Erstellung eines Radaufhängungsmodells mithilfe von neuronalen Netzen“* selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der eidesstattlichen Versicherung und prüfungsrechtlichen Folgen sowie die strafrechtlichen Folgen (siehe unten) einer unrichtigen oder unvollständigen eidesstattlichen Versicherung zur Kenntnis genommen.

Auszug aus dem Strafgesetzbuch (StGB)

§ 156 StGB

Falsche Versicherung an Eides Statt

Wer von einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

_____ _____

Ort, Datum Unterschrift

DOI: 10.17185/duepublico/72670
URN: urn:nbn:de:hbz:464-20200820-125511-4