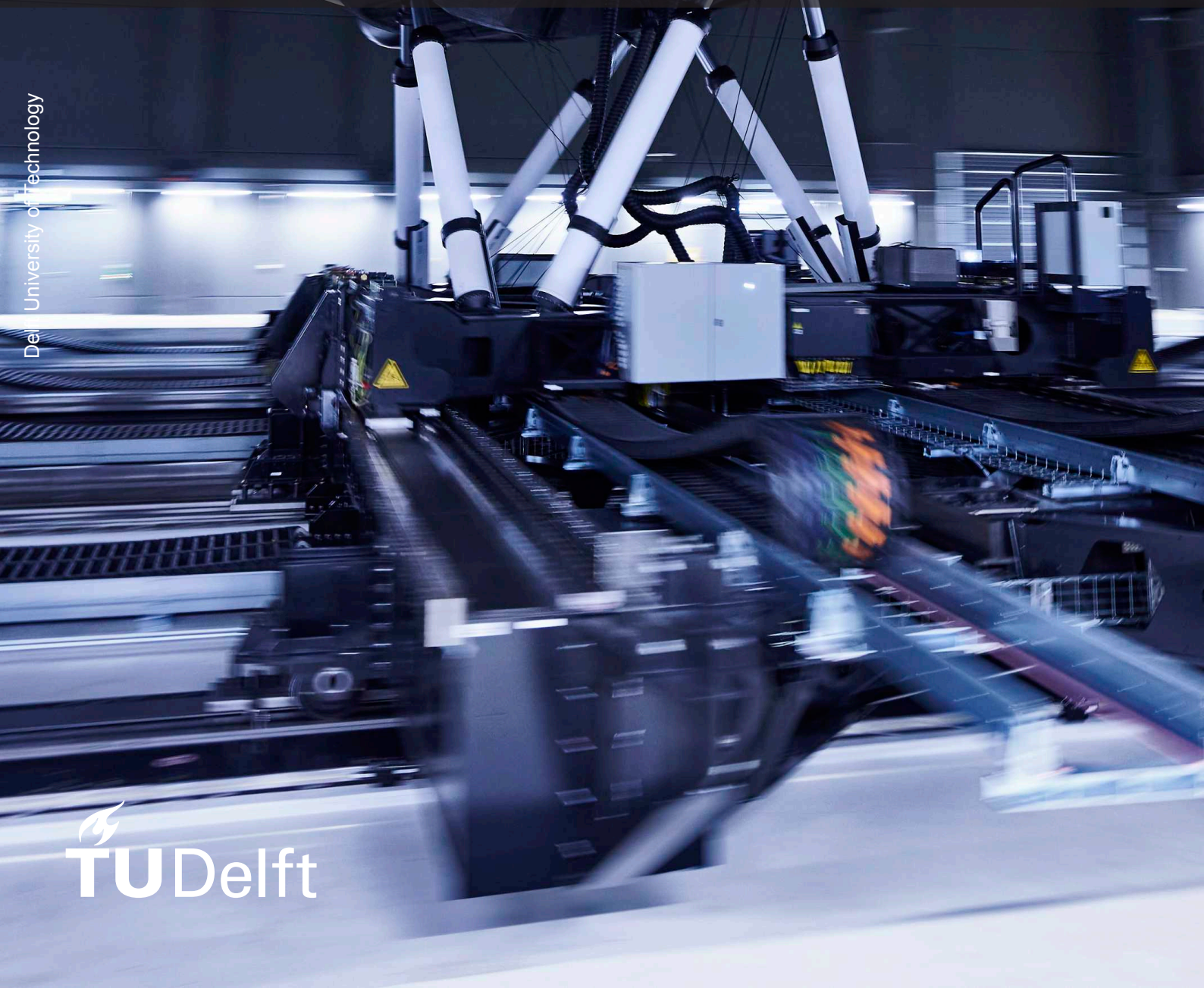
Motion Cueing in Driving Simulation

Effect of Neural Network Driver Predictions on the Expected Quality of a Model Predictive Control Algorithm for Driving Simulators

Master Thesis Report

Simon Beets



Delft University of Technology

TU Delft

Motion Cueing in Driving Simulation

Effect of Neural Network Driver Predictions on the Expected Quality of a Model Predictive Control Algorithm for Driving Simulators

by

Simon Beets

to obtain the degree of Master of Science
at the Delft University of Technology.
Publicly defended on the 18th of January 2023 at 13:00.

Student number:	4353870	
Project duration:	January 13, 2021 – January 18th, 2023	
Thesis committee:	Prof. dr. ir. M. Mulder,	TU Delft, Chair
	Dr. ir. D. Pool,	TU Delft, supervisor
	Dr. ir. J. Venrooij,	TU Delft/BMW, supervisor
	Ir. M. Kolff,	BMW, PhD candidate

TU Delft

Contents

List of Figures

List of Tables

List of Acronyms used in Preliminary Report

ACRONYM	FULL NAME
AS	Active Set Optimization
ADAM	Adaptive Moment Estimation
ADAS	Advanced Driver Assistance System
BMW	Bayerische Motoren Werke AG
BPTT	Backpropagation Through Time
CR	Continuous Rating
CRP	Control Reference Point
CWA	Classical Washout Algorithm
DIL	Driver-in-the-loop
DoF	Degree of Freedom
DSL	Driver Skill Level
EOM	Equation of Motion
HF	High Fidelity Simulator
IP	Interior Point
LA	Look-Ahead
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MCA	Motion Cueing Algorithm
MIR	Motion Incongruence Rating
MLP	Multi-Layer Perceptron
MPC	Model Predictive Control
MRP	Motion Reference Point
MSE	Mean Squared Error
NLA	Non-Look-Ahead
NS	North-South
OEM	Original Equipment Manufacturer
OTH	Otolith Organ
PR	Post-Hoc Rating
PMI	Perceived Motion Incongruence
QP	Quadratic Programming
RHC	Receding Horizon Control
RNN	Recurrent Neural Network
RMSE	Root Mean Square Error
SCC	Semi-Circular Canal

Introduction

Background Information

Ever since the mid 1970's dynamic driver simulators have been used as a tool in research and vehicle development in the automotive industry [86]. The main use-cases for dynamic driving simulators are in the field of research, and software and vehicle development [11, 79]. In case of vehicle, hardware development and human-centric research, a driver-in-the-loop (DIL) is required to give feedback on vehicle motion. In order to be able to give relevant feedback, it is desired for the motion of the simulator to be as close to reality as possible. This requires advanced control algorithms that are able to transform the real-time driver inputs to simulator motion commands, called motion cueing algorithms (MCA).

Today, the benchmark MCA that is used for both flight as well as driving simulators is a filter-based approach for which the foundation was laid by Schmidt and Conrad [85] in 1970. Although the algorithm has been further developed and expanded throughout the past decades, the basic working remains the same [77, 78, 80]. The algorithm utilizes combinations of low- and high-pass filters to transform the vehicle translational accelerations and rotational rates to usable simulator setpoints. Because the motion workspace of even the largest simulators is orders of magnitude smaller than that of an actual vehicle [11, 97], an exploit in the human vestibular system is used. Without extra external visual motion cues, a human is not able to distinguish between a specific force generated by rotation or acceleration given that the rotational rate stays below a certain threshold [26]. Using this exploit reduces the amount of required workspace to simulate vehicle motion. Although, the algorithm has been used for the past 50 years, researchers and the industry believe newer algorithms are able to solve some of the problems present in the filter-based approach.

In 2004, Dagdelen et al. [22] proposed to use a model predictive control MCA (MPC MCA). An MPC-based MCA tries finding an optimal control sequence over a predefined horizon that reduces the error between a discrete, future prediction and the modelled vehicle motion states. One major advantage compared to the industry standard filter-based control methods, is the explicit inclusion of system constraints. Research has shown that an MPC-based MCA has the ability to provide significantly improved motion cueing quality compared to the filter-based approaches, while also explicitly adhering to physical system limitations [24, 27, 60].

One of the main challenges of using MPC in driving simulation is providing a real-time prediction of future vehicle states, as these are dependent on driver inputs and therefore, inherently unknown. As was shown in [14, 26, 60], improving the quality of the reference given to MPC can significantly reduce motion stimuli mismatches, and therefore increase perceived motion cueing quality. In open-loop driving simulation, i.e., a scenario where all future vehicle specific forces and rotational rates are available, a perfect (oracle) reference based on the known future vehicle states can be used for tracking by the MPC. In experiments it was shown that using an oracle reference increases the perceived motion cueing fidelity significantly [26].

Since future vehicle states are influenced by future driver inputs, and thus inherently unknown, an oracle prediction is not available in real-time closed-loop simulations. Therefore, the current state-of-the-art is to use an extrapolated constant prediction, based on the current vehicle states. This prediction strategy is easy to implement and always possible independent of the driving scenario, but it limits the predictive potential of the MPC controller [26, 60].
In [24, 27] it was suggested that lengthening and improving the quality of the reference would contribute positively to MPC motion cueing quality, this resulted in the purpose of the presented research. The goal of this paper is to compare different data-driven machine learning frameworks, trained on data gathered during an experiment in a rural scenario [11]. This enables the different networks to predict the vehicle's future specific forces and rotational rates. The different frameworks are compared on a prediction quality level, as well as measured against the always available constant prediction. The best performing neural network will be used to generate the reference required by MPC, for which the resulting motion cueing will be thoroughly analyzed.

Report Outline

The report constitutes of three distinct parts:

Part I: Scientific Paper

Part I includes the scientific paper, as required to obtain the degree of Master of Science. In this paper, the models developed in the preliminary thesis are extended. The working principles of the 4-DoF MPC-based framework have been changed and extra features were added to the MPC model. The final result is a 6-DoF MPC-based MCA that can be used for longer prediction horizons. Next to this, the neural network models have been extended as well, introducing a different set of input features, using a different activation function for stability, and the introduction of a postprocessing cosinebell window function. The resulting predictions are analyzed and used in the MPC-based MCA framework. Instead of a DIL experiment, open-loop simulation results are used to give an estimation on motion cueing quality using both objective as well as subjective motion cueing quality metrics. The paper concludes with a discussion on the results, presenting some recommendations for future work.

Part II: Preliminary Thesis

At the moment of writing the preliminary thesis, i.e., Part II, as attached to this document is already graded. In this part of the report, a literature study has been performed on different important topics that relate to the field of motion cueing. This includes an investigation on the human biology of motion perception, as well as the modelling procedure and subsequent result analysis of a 6-DoF washout MCA and a 4-DoF MPC-based MCA. It also includes an extensive research towards different neural network modelling approaches. Four of these models are set-up, trained and preliminary results are reported. Finally, an initial DIL experiment proposal is given.

Part III: Appendices

The appendix consists of two parts, the first part contains the appendices to the preliminary thesis, these are referenced to in Part II, accordingly. The second part contains four appendices that mainly support ideas discussed in the scientific paper. Appendix B.1 gives more information about the relevant kinematic vehicle states predicted by the neural networks. Appendix B.2 provides information on the nonlinear relation between the available simulator motion space and its current state. In Appendix B.3, the three remaining network model structures are illustrated. Finally, in Appendix B.4, a comparison between an MPC using two set of weights is made.

Scientific Paper

Effect of Neural Network Driver Predictions on the Expected Quality of a Model Predictive Control Algorithm for Driving Simulators

S.W.M. Beets
Delft University of Technology
s.w.m.beets@student.tudelft.nl
Supervisors: M.J.C. Kolff, D.M. Pool, J. Venrooij & M. Mulder

Abstract*—One of the main challenges in using a closed-loop model predictive control (MPC) based motion cueing algorithm (MCA) is providing an accurate reference tracking signal consisting of future, driver influenced, vehicle states. In this paper four different neural network structures, i.e., a three-layer vanilla, a one-layer long short-term memory (LSTM) recurrent, a three-layer Deep LSTM recurrent, and an encoder-decoder neural network are introduced and trained to predict a discrete sequence of future vehicle states. The trained neural networks are compared on prediction level with the current benchmark prediction strategy, where future vehicle states are assumed to remain constant and equal to the current vehicle states. Through this analysis, a Deep LSTM recurrent neural network prediction was found to yield the strongest prediction error decrease in all Degree-of-Freedoms (DoF). A 6-DoF, open-loop MPC-based MCA was used to compare the influence of both the reference strategy, i.e., constant, Deep LSTM and perfect prediction capabilities (oracle), as well as the influence of imposing more or less neutral push on subjective motion cueing quality and motion cueing error type estimations. Compared to the constant MPC, it is estimated that the Deep LSTM MPC increases the subjective motion cueing quality by 13-22%, which is 5% less than an MPC using oracle prediction. It is also estimated, that significantly less missing, false and false direction cues are present in the resulting lateral cues provided by the Deep LSTM MPC compared to the constant MPC. It was also found that imposing less neutral push has a stronger positive effect on the MPC using a Deep LSTM and oracle prediction, showing the increased potential of providing higher accurate references for larger motion range systems. This paper successfully demonstrates that neural networks are able to accurately predict future, unknown, vehicle states and, when used as reference in MPC MCA, could result in significant improvements in both subjective motion cueing quality and a reduction in cueing type errors.

NOMENCLATURE

α	=	selu weighting factor exponential term
β_k	=	column vector of simulator angles
λ	=	regularization weight factor or selu weight factor
ω_c	=	cut-off frequency
$\omega_{x,y,z}$	=	rotational rate around x,y, and z-direction
ϕ	=	roll angle
ψ	=	yaw angle
ρ_h	=	activation function
σ	=	sigmoid activation function or standard deviation
θ	=	pitch angle
A	=	state matrix
$ADAM$	=	adaptive momentum estimation optimization
B	=	input matrix
$blkdiag$	=	block diagonal matrix
$BPTT$	=	backpropagation through time
C	=	output matrix
c_t	=	cell state LSTM cell
D	=	feedthrough matrix
DIL	=	driver-in-the-loop
DoF	=	degree-of-freedom
dt	=	sample time
E	=	cost term neural network optimization
f_T	=	forget gate LSTM cell
$f_{x,y,z}$	=	specific force in x,y, and z-direction
H	=	transfer function
i_t	=	input gate LSTM cell
ISO	=	international standardization organisation
J	=	cost function
j	=	specific time step in prediction horizon
$J_{a,b}$	=	Jacobian matrix transforming rotational rates from reference frame b to a
K	=	move window blocking matrix
k	=	current time instance
L_i	=	loss term neural network optimization
$LSTM$	=	long short-term memory cell
MAE	=	mean absolute error
MCA	=	motion cueing algorithm
MIR	=	motion incongruence rating
MLP	=	multi-layer perceptron model
MPC	=	model predictive control
MSE	=	mean squared error
MWB	=	move window blocking
N	=	number of input-output mappings
N_c	=	control horizon
N_p	=	prediction horizon
NN	=	neural network
NS	=	North-to-South direction

o_t = output gate LSTM cell
$p_{x,y,z}$ = positional coordinate in x,y, and z-direction
PID = proportional-integral-derivative controller
q = actuator deflection
Q_N = terminal state weighting
Q_u = input weighting matrix
Q_x = state weighting matrix
Q_y = reference tracking weighting matrix
Q_{du} = change in input weighting matrix
QP = quadratic problem
R = regularization term
r = reference trajectory
r_b = body reference frame
r_I = Inertial reference frame
RNN = recurrent neural network
SN = South-to-North direction
T_p = prediction time
$T_{a,b}$ = Euler transformation matrix from reference frame b to a
$tanh$ = hyperbolic tangent
u = applied input
W = neural network parameter value matrix
W_{f_x} = relative weight contribution specific force in x
x = state vector
x_N = final system state in N_p
x_s = simulator neutral state deviation
y = predicted system output, true reference or output vector

I. Introduction

In driving simulation, the goal is to perform driving experiments in a repeatable, simulated environment. Such experiments often require human participants or expert drivers who are able to provide high quality feedback. Depending on the requirements of an experiment, the accurate reproduction of motion cues can be important [1].

One of the major aspects that affects the perceived motion cueing quality is the ability of the driving simulator to significantly reduce vestibular motion stimuli mismatches [2–5]. A motion cueing control algorithm (MCA) is used to reduce the error between the actual and the perceived motion cues. An MCA uses outputs of the vehicle in the simulated environment to compute the required motion platform inputs. A high potential method, i.e., a real-time capable model predictive control (MPC) MCA, has been extensively researched for the past 20 years [4–6].

An MPC-based MCA tries finding an optimal control sequence over a predefined horizon that reduces the error between a discrete, future prediction and the modelled vehicle motion states. One major advantage compared to the industry standard filter-based control methods, is the explicit inclusion of system constraints. Research has shown that an MPC-based MCA has the ability to provide significantly improved motion cueing quality compared to the filter-based approaches, while also explicitly adhering to physical system limitations [7–9].

One of the main challenges of using MPC in driving simulation is providing a real-time prediction of future vehicle states, as these are dependent on driver inputs and therefore, inherently unknown. As was shown in [8, 10, 11], improving the quality of the reference given to MPC can significantly reduce motion stimuli mismatches, and therefore increase perceived motion cueing quality. In open-loop driving simulation, i.e., a scenario where all future vehicle specific forces and rotational rates are available, a perfect (oracle) reference based on the known future vehicle states can be used for tracking by the MPC. In experiments it was shown that using an oracle reference increases the perceived motion cueing fidelity significantly [11].

Since future vehicle states are influenced by future driver inputs, and thus inherently unknown, an oracle prediction is not available in real-time closed-loop simulations. Therefore, the current state-of-the-art is to use an extrapolated constant prediction, based on the current vehicle states. This prediction strategy is easy to implement and always possible independent of the driving scenario, but it limits the predictive potential of the MPC controller [8, 11].

[10] found that using improved references, based on prerecorded racetrack data, in a closed-loop experiment resulted in an increased use of motion workspace in longitudinal direction. In [9] it was found that using a PID regulated kinematic vehicle model, used to estimate future vehicle states, resulted in considerable improvements in motion cueing quality in closed-loop simulations. [12] and [8] propose to use a simple multi-layer perceptron (MLP) neural network as prediction method. In both cases, past information is fed to the network in real-time to provide a prediction on future vehicle states. The authors in [8] distinguish themselves from [12] by also including look-ahead information from a fixed driving direction to make real-time predictions. [8] showed that, when evaluating the absolute relative error, a reduction of around 60-80% was found when using the neural network predictions compared to a constant reference.

Although literature shows that using simple models that predict future vehicle states can provide a significant decrease in vestibular motion stimuli mismatches, the effect on the different types of motion cueing errors as well as the effect on subjective high temporal cueing evaluations is not fully understood. More advanced methods, able to provide high temporal estimations on the subjective motion cueing fidelity based on cueing errors, exist [13, 14].

These models estimate a continuous motion incongruence rating (MIR) based on longitudinal and lateral cues, i.e., how much does the presented motion by the simulator differ from the expected motion, rated on a scale from 0-10. A '0' implying no perceivable motion mismatches are felt, a '10' implying the most extreme perceived motion mismatches are experienced [7, 13, 14]. Using the MIR estimation model, a high-temporal resolution correlation between the prediction strategy and the subjective cueing quality can be established.

Next to an estimated subjective metric, [14] also proposes a method which measures several types of motion cueing errors by analyzing the type of motion cueing mismatch. This analysis provides an objective metric for motion cueing quality.

More advanced prediction neural networks, than the vanilla

multi-layered MLP network reported in [8] and [12], exist. Some of these are specifically designed to perform temporal sequence modelling. These models have often shown to outperform the vanilla MLP neural network in such tasks [15], and could improve the prediction quality even further when applied for vehicle state prediction.

Next to this, using the prediction of the improved neural network models in an MPC MCA could result in further improved motion cueing quality. To investigate the effect on the objective and subjective motion cueing metrics, the aforementioned motion cueing quality estimation models can be used in this analysis [13, 14].

The goal of this paper is to compare different data-driven machine learning frameworks, trained on data gathered during an experiment in a rural scenario [16]. This enables the different networks to predict the vehicle's future specific forces and rotational rates. The different frameworks are compared on a prediction quality level, as well as measured against the always available constant prediction. The best performing neural network prediction model is selected for open-loop MPC MCA simulations. MPC motion cueing quality, using the neural network reference, is compared to

1) An MPC using a constant prediction, and
2) An MPC with perfect prediction capabilities, i.e., oracle.

Two MPC weighting settings, one featuring a stronger neutral push effect, are used for these simulations in order to understand if and by how much the neutral push affects estimated cueing performance. These settings are compared using the MIR rating models and the motion cueing type error analysis presented earlier [13, 14]. Using both methods provides a full, complementary estimation on the influences on motion cueing quality on both an objective and subjective scale.

The structure of the paper is as follows, Section II introduces the 6-DoF MPC algorithm used for the presented analysis. Section III presents the full process of training and implementing the different neural network structures. Section IV shows the apparatus for which the analyses are made, provides the methods used to train the neural networks, and introduces the evaluation models used to analyze the MPC simulation results. Section V presents the simulation analysis results. Section VI provides the reader with a discussion on the results and a discussion on possible future work. The conclusions of the research are explained in Section VII.

II. Model Predictive Control

This section presents the MPC MCA setup used to perform the motion cueing analysis. The presented form of the MPC MCA follows the structure as defined in [11]. This paper considers an open-loop analysis only, i.e., real-time computational complexity is not an explicit consideration. However, [11] has shown that the presented framework is closed-loop capable.

A. MPC Formulation

A simplified schematic of the MPC control structure is given in Figure 1. In this figure the general flow of how MPC operates is shown. MPC solves a linear constrained optimization problem every discrete time instant to find an optimal sequence of simulator inputs, i.e., ΔU, of length N_c, called the control horizon. MPC does so by minimizing a cost function reducing a reference tracking error of length N_p, called the prediction horizon, for which the following condition holds $N_c \leq N_p$ [17]. However, in this work, the prediction horizon is assumed to be equal to the control horizon, i.e., $N_p = N_c$. In order to perform the numerical optimization, a kinematic simulator model is required to predict future system states for all time samples in N_p.

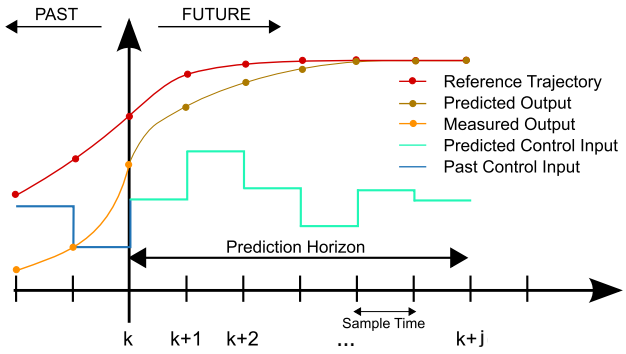


Fig. 1: Schematic of MPC horizonss. Source: [18].

In Figure 1, k denotes the current time step and j the specific time step in the prediction horizon for which $j \in [k+1, k+N_p]$. The full, linear MPC optimal control problem can be formulated as presented in Equation (1)-(2). J denotes the cost as a function of several specific cost terms. The first term in J denotes the reference tracking error. This term minimizes the error between the reference trajectory, $\vec{r}$, and the predicted system output, $\vec{y}$. The second term in J is defined as the cost on simulator neutral state deviations, $\vec{x}_s$, which effectively tries to push the system state to its neutral position. This term helps for optimization feasibility by keeping the excursions away from the constraint boundaries by providing washout. The third term depicts the cost on total applied input, $\vec{u}$, which is effectively an energy efficiency term. The fourth term is the cost on applied change in input, $\Delta\vec{u}$, reducing jerky inputs, which has the benefit of limiting strenuous forces on the actuators as well as removing chattering behavior in the perceived motion cues. The final term shows the terminal state cost term, which ensures asymptotic stability [17].

Each term features their own weight term matrix Q. Computation of the minimum cost is subject to the kinematic model defined in state-space form, as well as constraints on actuator position q, total input $\vec{u}$ and change in input $\Delta\vec{u}$, see Equation (1).

$$J = \min \sum_{j=1}^{N_p} \|\vec{y}(k+j) - \vec{r}(k+j)\|^2_{Q_y} + \sum_{j=1}^{N_p} \|\vec{x}_s(k+j)\|^2_{Qx} + \sum_{j=0}^{N_c-1} \|\vec{u}(k+j)\|^2_{Qu} + \sum_{j=0}^{N_c-1} \|\Delta\vec{u}(k+j)\|^2_{Qdu} + \vec{x}^2_{N_{Q_N}} \tag{1}$$

$$s.t.: \begin{cases} \vec{x}_{k+1} = A \cdot \vec{x}_{s_k} + B \cdot \Delta\vec{u}_{s_k} \\ \vec{y}_k = C \cdot \vec{x}_{s_k} + D \cdot \Delta\vec{u}_{s_k} \\ q_{min} \leq q \leq q_{max} \\ \vec{u}_{min} \leq \vec{u} \leq \vec{u}_{max} \\ \Delta\vec{u}_{min} \leq \Delta\vec{u} \leq \Delta\vec{u}_{max} \end{cases} \tag{2}$$

Constraints on actuator velocity and acceleration can also be implemented. However, for the considered system these limits are not reached in driving simulation and can therefore be omitted.

B. Conversion of Reference Systems

In Figure 1, *Reference Trajectory*, denotes the reference signal the output of the simulator should follow as closely as possible, i.e., $\vec{r}(k+j)$ in Equation (1). The reference comprises of a sequence of N_p discrete samples of future specific forces in all translational axes and rotational rates around each axes in the Inertial reference frame, i.e., $\vec{r} = [f_x, f_y, f_z, \omega_x, \omega_y, \omega_z]^T$. These values are defined in the driver body reference frame where a driver is assumed to perceive motion relative to his/her own. However, the simulator is controlled through Inertial motion setpoints. See Figure 2, where r_I denotes the Inertial and r_b the body reference frame. It should be noted that in Figure 2 the Inertial reference frame is drawn as if attached to the bottom frame of the hexapod system. However, I is earth-fixed and does not move together with the shown XY-table.

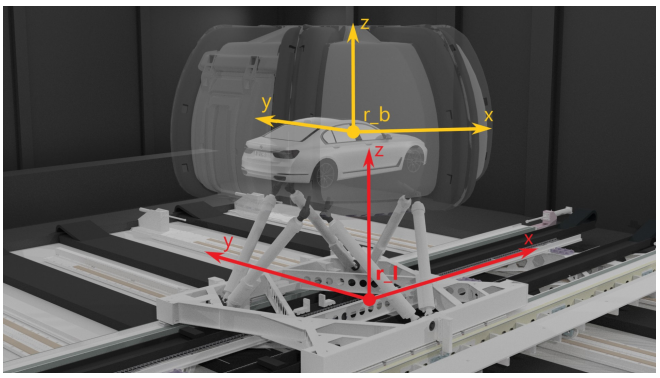


Fig. 2: Body-fixed reference frame following conventions from ISO 8855 [19] and Inertial reference frame. Source: [16].

This requires the inclusion of a sequence of Euler rotations for both translational as well as rotational motion. To transform a vector of specific forces defined in the body reference frame in the Inertial reference frame, the following relation holds true [5]:

$$\begin{aligned} \vec{f}_{I,b} &= T_{I,b} \cdot \vec{f}_b \\ T_{I,b} &= T_z(-\psi) \cdot T_y(-\theta) \cdot T_x(-\phi) \end{aligned} \tag{3}$$

In Equation (3), $\vec{f}_{I,b} = [f_{I,b_x}, f_{I,b_y}, f_{I,b_z}]^T$ denotes the vector of specific forces transformed from the body b, to Inertial reference frame I. $T_{I,b}$ denotes the Euler transformation matrix following a sequence of transformations around x with angle $-\phi$, around y with angle $-\theta$, and finally a transformation around z with angle $-\psi$. The opposite transformation also holds true.

$$T_{b,I} = T_x(\phi) \cdot T_y(\theta) \cdot T_z(\psi) \tag{4}$$

The rotational Jacobian matrix, which transforms rotational rates from a body to the Inertial reference frame is defined as:

$$\vec{\omega}_{I,b} = \underbrace{\begin{bmatrix} c(\psi)c(\theta) & -s(\psi) & 0 \\ s(\psi)c(\theta) & c(\psi) & 0 \\ -s(\theta) & 0 & 1 \end{bmatrix}}_{J_{I,b}} \cdot \begin{bmatrix} \dot{\phi}_b \\ \dot{\theta}_b \\ \dot{\psi}_b \end{bmatrix} \tag{5}$$

Here 'c' and 's' denote the cosine and sine, respectively. Similarly, the following relation also holds:

$$J_{b,I} = \begin{bmatrix} 1 & 0 & -s(\theta) \\ 0 & c(\phi) & c(\theta)s(\phi) \\ 0 & -s(\phi) & c(\theta)c(\phi) \end{bmatrix} \tag{6}$$

A full derivation of the presented matrices can be found in [5].
In order to keep the MPC problem definition linear, these nonlinear transformation matrices cannot be used to provide future state estimations. Therefore, the transformation matrices are assumed constant over the prediction horizon N_p, as discussed in [5].

C. Defining Kinematic Simulator Model Elements

By definition a kinematic simulator model, written in state-space formulation, is required to make predictions about future simulator states, see Equation (2). Simulator states, i.e., $\vec{x}_s$, are represented in the Inertial reference frame and comprise of the positional and angular coordinates, velocities and accelerations.

Considering a 6 Degree-of-Freedom (DoF) hexapod platform, the state vector has a dimensionality of 9, i.e., three inertially defined positions, velocities, and angles. The input state vector has a dimensionality of 6, i.e., three inertially defined accelerations and rotational rates. With $\vec{x} = \left[p_x, \dot{p}_x, p_y, \dot{p}_y, p_z, \dot{p}_z, \phi, \theta, \psi\right]^T \in \mathbb{R}^9$ and $\vec{u} = \left[\ddot{p}_x, \ddot{p}_y, \ddot{p}_z, \dot{\phi}, \dot{\theta}, \dot{\psi}\right]^T \in \mathbb{R}^6$. Here, p denotes the positional coordinate in $[x, y, z]$ in the Inertial reference frame and $[\phi, \theta, \psi]$ the roll, pitch and yaw of the simulator. Assuming the actuator control loops are fast, the kinematic motion of the simulator can be modelled conform a discrete single and double integrator model for angular and positional coordinates, respectively [4]. The integrator models can be written in discrete state-space format, see Equation (2), with subscript k denoting the current time instant. For an arbitrary translational acceleration and a rotational rate the following holds:

$$\begin{aligned} \begin{bmatrix} p_{x_{k+1}} \\ \dot{p}_{x_{k+1}} \end{bmatrix} &= \underbrace{\begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}}_{A_p^*} \cdot \begin{bmatrix} p_{x_k} \\ \dot{p}_{x_k} \end{bmatrix} + \underbrace{\begin{bmatrix} \frac{\Delta t^2}{2} \\ \Delta t \end{bmatrix}}_{B_p^*} \cdot \ddot{p}_x \\ \beta_{k+1} &= \underbrace{[1]}_{A_\beta^*} \cdot \beta_k + \underbrace{[\Delta t]}_{B_\beta^*} \cdot \dot{\beta}_k \end{aligned} \tag{7}$$

Here, β_k denotes the column vector of simulator angles $[\phi, \theta, \psi]^T$. Also, A_p^* and B_p^* are parts of the full state-space matrices, A_p and B_p, and A_β^* and B_β^* are part of A_β and B_β.

$$\begin{aligned} A_p &= blkdiag(A_p^*, A_p^*, A_p^*) \\ B_p &= blkdiag(B_p^*, B_p^*, B_p^*) \\ A_\beta &= blkdiag(A_\beta^*, A_\beta^*, A_\beta^*) \\ B_\beta &= blkdiag(B_\beta^*, B_\beta^*, B_\beta^*) \end{aligned} \tag{8}$$

In Equation (8), $blkdiag$ denotes a block-diagonal matrix. To compute the perceived specific forces and rotational rates, a vestibular model for the otoliths and semi-circular canals is incorporated in the kinematic model of the simulator as well. The transfer functions of both organs were set to the ones reported by [20], and are defined as:

$$\begin{aligned} \vec{f}_{s_p} &= H_{oth}(s) \cdot \vec{f}_s, \quad H_{oth}(s) = \frac{0.4 \cdot (1 + 10s)}{(1 + 5s)(1 + 0.016s)} \\ \vec{\omega}_{s_p} &= H_{scc}(s) \cdot \vec{\omega}_s, \quad H_{scc}(s) = \frac{5.73 \cdot (80s)}{(1 + 80s)(1 + 5.73s)} \end{aligned} \tag{9}$$

With $\vec{f}_{s_p}$ and $\vec{\omega}_{s_p}$ denoting the perceived specific forces and rotational rates, and H_{oth} and H_{scc} denoting the transfer functions in the Laplace domain of the otholiths and the semi-circular canals, respectively. Discretizing results in a set of state-space matrices A_o, B_o, C_o, D_o for the otolith organs and A_{scc}, B_{scc}, C_{scc}, D_{scc} for the semi-circular canals.

In simulators, a specific force can also be generated by using the gravity vector when a tilting motion of the simulator is provided, the so called tilt-coordination. Using the small angle approximation (roll and pitch angles of a car are generally small) and the right hand rule, one can find the following relation for the tilt-coordination [7], where Equation (10) is used in Equation (12):

$$\vec{f}_{tilt} = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} = \begin{bmatrix} -g \cdot s(\theta) \\ g \cdot c(\theta) s(\phi) \\ g \cdot c(\theta) c(\phi) \end{bmatrix} = \begin{bmatrix} -\theta \\ \phi \\ 1 \end{bmatrix} g \tag{10}$$

D. Defining Full State-Space Kinematic Simulator Model

Combining the information from the previous two subsections, the following full state-space system can be defined, with [5]:

$$\begin{aligned} A_S &= \begin{bmatrix} A_o & B_o H_o & 0_{6\times12} \\ 0_{6\times9} & A_{scc} & 0_{6\times6} \\ 0_{6\times9} & 0_{6\times6} & A_s \end{bmatrix} B_S = \begin{bmatrix} B_o T_{b,I} & 0_{6\times3} \\ 0_{3\times3} & I_3 \Delta t \\ 0_{6\times3} & B_{scc} J_{b,I} \\ & B_s \end{bmatrix} \\ C_S &= \begin{bmatrix} C_{o,\beta} & 0_{6\times6} & 0_{6\times6} \\ 0_{3\times9} & C_{scc} & 0_{3\times6} \\ 0_{6\times6} & 0_{6\times6} & I_6 \end{bmatrix} \quad D_S = \begin{bmatrix} D_o T_{b,I} & 0_{3\times3} \\ & 0_{3\times6} \\ 0_{3\times3} & D_{scc} J_{b,I} \\ & 0_{6,6} \end{bmatrix} \end{aligned} \tag{11}$$

H_o is defined as the matrix including the tilt-coordination, see Equation (12). I_n denotes an identity matrix of size n. The nonlinear terms, $J_{b,I}$ and $T_{b,I}$, are kept constant over N_p.

$$H_o = blkdiag(H', A_\beta) \qquad H' = \begin{bmatrix} 0 & -g & 0 \\ g & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{12}$$

The resulting state vector $\vec{x}_{s_k}$, input vector $\vec{u}_{s_k}$ and output vector $\vec{y}_{s_k}$ are defined as:

$$\begin{aligned} \vec{x}_{s_k} &= \left[x_{oth}, \phi, \theta, \psi, x_{scc}, p_x, \dot{p}_x, p_y, \dot{p}_y, p_z, \dot{p}_z\right]^T \in \mathbb{R}^{21} \\ \vec{u}_{s_k} &= \left[\ddot{p}_x, \ddot{p}_y, \ddot{p}_z, \dot{\phi}, \dot{\theta}, \dot{\psi}\right]^T \in \mathbb{R}^6 \\ \vec{y}_{s_k} &= \begin{bmatrix} \vec{f}_{x_p}, \vec{f}_{y_p}, \vec{f}_{z_p}, \phi, \theta, \psi, \vec{\omega}_{x_p}, \\ \vec{\omega}_{y_p}, \vec{\omega}_{z_p}, p_x, \dot{p}_x, p_y, \dot{p}_y, p_z, \dot{p}_z \end{bmatrix}^T \in \mathbb{R}^{15} \end{aligned} \tag{13}$$

As shown in Equation (2), the discrete state-space representation needs to be written as a function of $\Delta\vec{u}_{s_k} = \vec{u}_{s_k} - \vec{u}_{s_{k-1}}$. This can be accomplished by augmenting the state vector, $\vec{x}_{s_k}$ to $X_k = \left[\vec{x}_{s_k}, \vec{u}_{s_{k-1}}\right]^T$, which results in the new augmented state-space system with new state vector X_k written in function of $\Delta\vec{u}_{s_k}$:

$$\begin{aligned} \underbrace{\begin{bmatrix} \vec{x}_{s_{k+1}} \\ \vec{u}_{s_k} \end{bmatrix}}_{X_{k+1}} &= \underbrace{\begin{bmatrix} A_S & B_S \\ 0 & I \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \vec{x}_{s_k} \\ \vec{u}_{s_{k-1}} \end{bmatrix}}_{X_k} + \underbrace{\begin{bmatrix} B_S \\ I \end{bmatrix}}_{B} \Delta\vec{u}_{s_k} \\ Y_k &= \underbrace{\begin{bmatrix} C_S & 0 \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} \vec{x}_{s_k} \\ \vec{u}_{s_{k-1}} \end{bmatrix}}_{X_k} + \underbrace{\begin{bmatrix} D_S \\ 0 \end{bmatrix}}_{D} \Delta\vec{u}_{s_k} \end{aligned} \tag{14}$$

With the augmented state vector equalling:

$$X_k = \begin{bmatrix} x_{oth}, \phi, \theta, \psi, x_{scc}, p_x, \dot{p}_x, p_y, \dot{p}_y, p_z, \dot{p}_z, \ddot{p}_x, \\ \ddot{p}_y, \ddot{p}_z, \dot{\phi}, \dot{\theta}, \dot{\psi} \end{bmatrix}^T \in \mathbb{R}^{27} \tag{15}$$

Using recursive relations, see Equation (16), a state prediction for each $t_k \in [k, k + N_p]$ in J can be established. The resulting state-space system is presented in Equation (17) [5]. In Equation (17), $\bar{X} = \left[X_1 \cdots X_{N_p}\right]$, $\bar{Y} = \left[Y_1 \cdots Y_{N_p}\right]$, and $\Delta\bar{U} = \left[\Delta\vec{u}_{s_0} \cdots \Delta\vec{u}_{s_{N_p-1}}\right]$.

$$\begin{aligned} \vec{x}_1 &= A \cdot \vec{x}_0 + B \cdot \Delta\vec{u}_0 \\ \vec{x}_2 &= A \cdot \vec{x}_1 + B \cdot \Delta\vec{u}_1 \\ \Leftrightarrow \vec{x}_2 &= A \cdot (A \cdot \vec{x}_0 + B \cdot \Delta\vec{u}_0) + B \cdot \Delta\vec{u}_1 \\ \Leftrightarrow \vec{x}_2 &= A^2 \cdot \vec{x}_0 + AB \cdot \Delta\vec{u}_0 + B \cdot \Delta\vec{u}_1 \end{aligned} \tag{16}$$

$$\begin{aligned} \bar{X} &= \underbrace{\begin{bmatrix} A \\ A^2 \\ \vdots \\ A^{N_p} \end{bmatrix}}_{F_x} X_0 + \underbrace{\begin{bmatrix} B & 0 & \cdots & 0 \\ AB & B & 0 & \cdots \\ \vdots & \ddots & \ddots & 0 \\ A^{N_p-1}B & A^{N_p-2}B & \cdots & B \end{bmatrix}}_{S_x} \Delta\bar{U} \\ \bar{Y} &= \underbrace{\begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^{N_p} \end{bmatrix}}_{F_y} X_0 + \underbrace{\begin{bmatrix} CB & 0 & \cdots & 0 \\ CAB & CB & 0 & \cdots \\ \vdots & \ddots & \ddots & 0 \\ CA^{N_p-1}B & CA^{N_p-2}B & \cdots & CB \end{bmatrix}}_{S_y} \Delta\bar{U} \end{aligned} \tag{17}$$

In Equation (17), $F_x \in \mathbb{R}^{(N_p \times m) \times m}$, $F_y \in \mathbb{R}^{(N_p \times m) \times (N_p \times n)}$, $S_x \in \mathbb{R}^{(N_p \times o) \times m}$ and $S_y \in \mathbb{R}^{(N_p \times o) \times (N_p \times n)}$, with m equal to the number of states, n the number of inputs and o the number of output states. $\bar{X}$, $\bar{Y}$ and $\Delta\bar{U}$ denote the state vector, output vector and input vector, respectively, containing all the information for the full prediction horizon N_p.

E. Move Window Blocking

The complexity of solving the optimal control problem is predominantly influenced by the number of included DoFs [21, 22]. The DoFs of the control problem are defined as the number of inputs n multiplied with the prediction horizon N_p. In order to reduce the number of DoFs, and thus complexity, a strategy called *move window blocking* (MWB) is applied to the system, which guarantees stability and feasibility of the problem [23]. When applying MWB to an MPC formulation, a time-dependent blocking matrix, K, is introduced which formulates that several input values or its derivatives remain constant over time. In the simulations presented in this paper, a prediction horizon of $N_p = 4s$ with a sampling time of $dt = 0.01s$ is used. For this problem a MWB strategy equal to $s = [50, 10, 10]$ with $t_s = [0.01s, 0.1s, 0.25s]$ was utilized. This indicates that the original 400 time samples are lowered to a total of 70 samples, of which the first 50 are sampled at $0.01s$, the next 10 are sampled at 0.1s, and the last 10 are sampled at intervals of 0.25s. For the second and third set of samples the inputs between subsequent time intervals remain constant. The blocking matrix K is defined as follows:

$$K = \begin{bmatrix} K_1 & 0 & 0 \\ 0 & K_2 & 0 \\ 0 & 0 & K_3 \end{bmatrix} \; K1 = \begin{bmatrix} I_n & 0 & \cdots & 0 \\ 0 & I_n & \cdots & 0 \\ \vdots & \ddots & I_n & 0 \\ 0 & \cdots & 0 & I_n \end{bmatrix}$$
$$K2/K3 = \begin{bmatrix} K' & 0 & \cdots & 0 \\ 0 & K' & \cdots & 0 \\ \vdots & \ddots & K' & 0 \\ 0 & \cdots & 0 & K' \end{bmatrix} \; with\, K' = \vec{1}_{t_s/dt,1} \otimes I_u \tag{18}$$

With $I_n \in \mathbb{R}^n$ an identity matrix of dimension n, $\vec{1}_{t_s/dt,1}$ a column vector of ones with t_s/dt amount of elements, and $\otimes$ the Kronecker product. Using this strategy, the effective DoFs are reduced from $400 \cdot 6 = 2,400$ samples to $70 \cdot 6 = 420$ samples. The MWB matrix, K, is applied by pre-multiplying it with the input vector $\Delta\bar{U}$ in the cost function J, as well as with the relative constraint equations.

F. Constraints

As shown in Equation (2), no state constraints in the Inertial reference frame are present. However, constraints are in place for the mechanical restrictions of the system by means of implementing actuator constraints. The output vector, described in Equation (13), explicitly includes the Inertial workspace domain coordinates. These can be used to compute the actuator excursions [5]. Considering the following differential equation:

$$\dot{q}_l = \frac{1}{2q_l}\frac{dq_l^2}{dt} = J_l(w_s)\dot{w}_s \tag{19}$$

Here, $\dot{q}_l$ denotes the velocity of actuator l, w_s is the current workspace attenuation, i.e., $w_s = [p_x, p_y, p_z, \phi, \theta, \psi]^T$ the absolute position and angular displacement w.r.t. the neutral point, and $J_l(w_s)$ is the inverse Jacobian relating a change in workspace attenuation with actuator length l. The inverse Jacobian depends on the current workspace attenuation w_s. Transforming the differential equation to infer a linear relationship with the current workspace attenuation and actuator velocity, and subsequently numerically integrating, one finds the following state-space transformation:

$$\begin{aligned} \vec{q}_{k+1} &= \vec{q}_k + A_{act}X_{D_0} + B_{act}\Delta\vec{u}_{s_k} \\ \bar{Q} &= \vec{q}_0 + F_{act}X_{D_0} + S_{act}\Delta\bar{U} \end{aligned} \tag{20}$$

Here, A_{act} and B_{act} contain the relevant parts of the inverse Jacobian $J_l(w_s)$, F_{act} and S_{act} are derived in a similar fashion using recursion as presented in Equation (17). $\vec{q}_0$ is the matrix containing the current actuator lengths, i.e., $\vec{q}_0 = (1_{Np,1} \otimes I_{n_{act}}) \cdot \vec{q}_k$ with n_{act} equal to the amount of actuators. The nonlinear terms of $J_l(w_s)$, present in F_{act} and S_{act}, are kept constant over N_p. The full derivation of the equations can be found in [7].

G. Solver

The MPC problem is defined in Equation (1) and Equation (2), and can be written as a quadratic problem (QP). Solving said optimization problem requires a quadratic solver. In this paper, the open source *qpOases* software is used to solve the QP-problem over the prediction horizon, N_p, for each subsequent discrete time step in the simulation [24]. *qpOases* uses an online active set strategy to solve a QP formulated optimization problem. One of the reasons why such active set strategies can be used for MPC applications is the possibility to hot-start encode the solution. This means that, following the assumption that the active set does not change a lot from one QP solution to the next, the former solution can be used to make the optimization faster.

III. Supervised Neural Networks

The previous Section described the MPC paradigm for motion cueing. As explained, one of the main challenges of using an MPC MCA is including an updated reference trajectory of N_p discrete samples at each time step k. This means providing MPC with not readily available predictions of future vehicle accelerations and rotational rates. This section provides four supervised neural network approaches, able to predict a discrete sequence of vehicle specific forces $\vec{f}_s$ and rotational rates $\vec{\omega}_s$.

An overview on the general MPC MCA block diagram is given in Figure 3. Figure 3 shows that a neural network driver prediction module provides the MPC MCA with a sequence of N_p future samples of specific forces and rotational rates. The MPC MCA uses this prediction to send an updated positional, velocity, and acceleration setpoint to the physical

simulator system. As will be discussed in this section, the proposed neural networks use information from the kinematic vehicle simulation as well as environmental context variables to update the specific force and rotational rate predictions. In this paper the proposed neural networks predict a sequence of length $T_p = 4s$ [8], T_p demeaning the prediction time, using information from the past $5s$. It is assumed that information from more than 5 seconds ago does not influence the future, short-term driving behavior to a significant extent. Providing a neural network with non-rich information can lead to a loss of prediction performance [25].

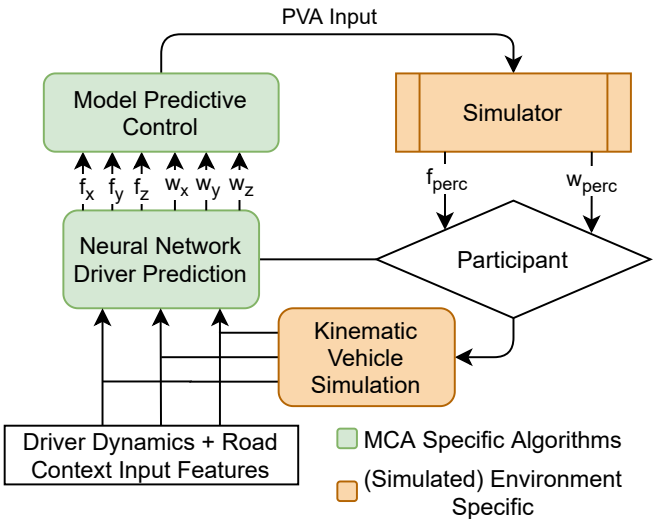


Fig. 3: Blockdiagram showcasing the driver prediction and MPC MCA module [8].

A. Data-set

A supervised machine learning method uses prerecorded data, fed to a model able to learn high dimensional input-output relations. In this paper, prerecorded data from an online experiment on a rural road area were used [11]. The hilly section features multiple speed limit changes, traversing a small village, various curves and a round-about. The route with the different sections is shown in Figure 4. Each colored section, presented in Figure 4, corresponds with a change in legal speed limit, see Table I.

TABLE I: Speed limit change segments.

Road Segment	*Situation*
$1 \rightarrow 2$	Rural road entering village
$2 \rightarrow 3$	Leaving village to rural road
$3 \rightarrow 4$	Sharp hilltop, reduced speed limit rural road
$4 \rightarrow 5$	Sharp hilltop, increased speed limit rural road
$5 \rightarrow 6$	Approaching roundabout, reduced speed limit
$6 \rightarrow 7$	Leaving roundabout, increased speed limit rural road
$7 \rightarrow 8$	Sharp turn, reduced speed limit
$8 \rightarrow 9$	Approaching village, reduced speed limit
$9 \rightarrow 10$	Leaving village, increased speed limit
$10 \rightarrow 11$	Entering village, reduced speedlimit

The data-set contains 33 independent drives, driven in North-to-South (NS) and South-to-North (SN) direction. Participants, driving a virtual vehicle model of a 2018 BMW

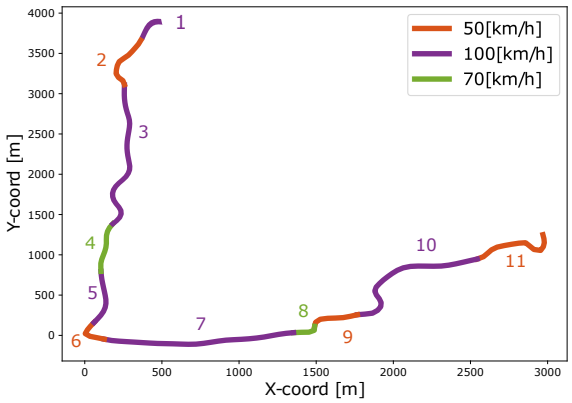


Fig. 4: Road layout, segmentation based on local speed limit.

530i, were asked to drive as they normally do every day, i.e., adhering to speed limits and traffic regulations. The dataset is near symmetric, i.e., 13 drives from NS, 14 from SN. Six prematurely abandoned drives were omitted from the data-set. Each drive, with an average driving time of $7.5mins$, consists of 62 different variables. The variable set consists of road, vehicle kinematic, simulator dynamics, and driver input specific variables. The data is recorded at a frequency of 100Hz resulting in the NS data-set containing around 650,000, and the SN data-set containing around 724,000 datapoints. Some variables, e.g., longitudinal acceleration, showed high-frequency noise which was removed by applying a 1Hz, 2nd-order Butterworth filter [26].

B. Data Preprocessing

The MPC reference vector consists of the vehicle specific force in each direction as well as the vehicle rotational rates around each axis. Therefore, the model output vector consists of the six output features found in Table II. Time traces of the longitudinal and lateral acceleration, as well as the yaw rate profiles of all participants, driving the rural route in NS direction, can be found in Figure 5-7. The profiles in SN direction can be found in the appendix attached to this document. From these figures one can see that the spread in longitudinal acceleration is larger than the spread of the lateral acceleration and yaw rate, this is caused by the latter two being less dominantly influenced by the driver, but more by the road curvature and speed limits [7]. From Figure 5, it can be deduced that the largest peaks in longitudinal acceleration occur around speed limit changes. Road section 6, in Figure 4, features a roundabout. When driving in NS direction a driver has to take the 3rd exit, i.e., almost full circle, whereas in SN direction a driver has to take the 1st exit, greatly reducing the driven radius of curvature. In Figure 6 and Figure 7 this results in higher peaks in both lateral acceleration as well as yaw rate.

Selecting the input feature set is not determined by external requirements, and thus not as straightforward as the set of output features. One option would be to include all 62 measurable features and let the models learn the required dependencies themselves. However by doing so, model complexity might

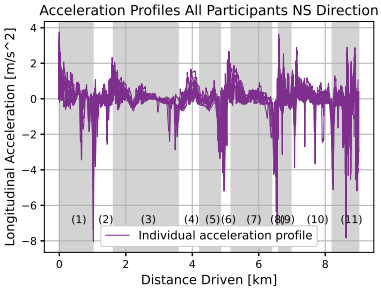


Fig. 5: Longitudinal acceleration profiles in NS direction.

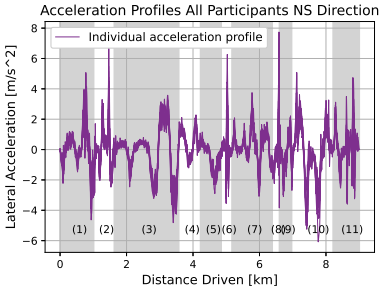


Fig. 6: Lateral acceleration profiles in NS direction.

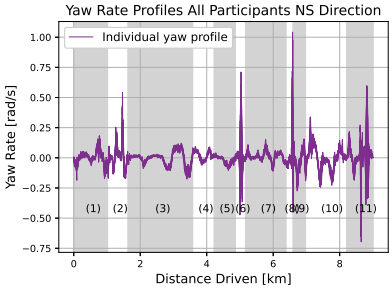


Fig. 7: Yaw rate profiles in NS direction.

TABLE II: Output features determined by MPC MCA reference trajectory requirement.

Output feature	Unit
Longitudinal acceleration	m/s^2
Lateral acceleration	m/s^2
Vertical acceleration	m/s^2
Pitch rate	rad/s
Roll rate	rad/s
Yaw rate	rad/s

unnecessarily increase and the richness of the input feature set might decrease. This could result in a decrease in model performance [25].

Based on literature [8, 27, 28], 18 input features were selected, each expected to influence future driving behavior. These features can be subdivided into two categories, 14 causal and 4 non-causal features. Both causal as well as non-causal road specific features, thus driver independent, are provided to the network. These include the absolute road curvature, road curvature sign, road elevation and speed limit. Explicitly defining input features that have a strong correlation with output features can improve model performance [29]. Therefore, the sign and the absolute value of the road curvature are presented to the networks separately. Here it is assumed that the benefit of adding two extra features outweighs the increase in model complexity. The full list of input features and their respective type is given in Table III.

TABLE III: Input feature selection: type and unit.

Type	Input feature	Unit
Non-Causal	*Road curvature*	$1/m$
Non-Causal	*Road curvature sign*	$-$
Non-Causal	*Road elevation*	m
Non-Causal	*Speed limit*	m/s
Causal	*Longitudinal acceleration*	m/s^2
Causal	*Lateral acceleration*	m/s^2
Causal	*Vertical acceleration*	m/s^2
Causal	*Applied throttle*	$-$
Causal	*Applied brake*	$-$
Causal	*Steering wheel angle*	rad
Causal	*Speed*	m/s
Causal	*Roll rate*	rad/s
Causal	*Pitch rate*	rad/s
Causal	*Yaw rate*	rad/s
Causal	*Road curvature*	$1/m$
Causal	*Road curvature sign*	$-$
Causal	*Road elevation*	m
Causal	*Speed limit*	m/s

From Table III, two points can be noted. First, not only future information, but also past information on the road condition is presented to the network. Second, presenting future information about specific road conditions is only possible when the driving direction is known a priori, i.e., as is the case on the road segment presented in Figure 4. If this is not the case, an external algorithm could provide such information by making a prediction about the future driving direction. In this paper the non-causal information is gathered using a look-ahead distance of $d_{la} = 100m$. This is roughly equal to the average speed limit on the road ($\approxeq 80km/h$) times the used prediction time of $T_P = 4s$.

Using information from the past $5s$ and an output sequence of length $T_p = 4s$, sampled at 100Hz would result in an input space, x_{in}, equal to $17 \times 500 = 8,500$ samples and an output space, y_{out}, equal to $6 \times 400 = 2,400$ samples. Using such a vast input and output dimensionality increases model complexity, which reduces computational efficiency. To reduce input and output dimensionality, 30 logarithmic spaced samples in the in- and output sequence, $x_{in} \in [k-500, k]$ and $y_{out} \in [k, k+400]$, are used to build the input-output mapping. A graphical representation is given in Figure 8.

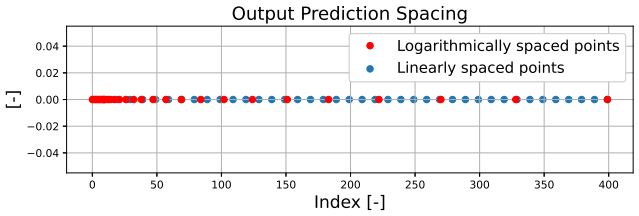


Fig. 8: Example of output feature spacing, showing 30 logarithmically and 40 linearly spaced samples.

In the output sequence a logarithmic sample spacing, instead of linearly spaced samples, has the effect of emphasizing short-term prediction quality while maintaining information about future trends. It was reasoned that the quality of the first sequence of samples has a greater effect on MCA quality than the prediction further in the future [8]. Logarithmic spacing in the input vector has the benefit of giving more recent, high detail information to the network which is assumed to have a larger effect on short-term driving behavior. Since MPC requires a linearly spaced reference sampled at 100Hz, the prediction output is linearly interpolated between time

intervals.

When training the neural networks, it is common practice to subdivide the input-output pairs into training, validation and test sets [29]. To serve this purpose, 70% of the drives are used in the training set (10 SN drives, 10 NS drives), 15% in the validation set (2 SN and 2 NS drives), and 15% in the test set (2 SN and 1 NS drive(s)).

In order to increase performance and generalization, the data is preprocessed [29, 30]. Each input and output feature is scaled using MinMax scaling to the interval $[0-1]$ [29]. Features are scales using minimum and maximum values obtained from the training data-container, such that no leakage of information occurs, i.e., implicitly or explicitly introducing information from the test into the training data-set.

Assuming no large changes in the data occur between samples, generating input-output mappings at the data-rate frequency of $100Hz$ could result in many similar input-output mappings. This could prove to be detrimental for network performance. In order to reduce this risk, the input-output mappings are generated at a lower frequency $f_{i/o}$, i.e., $f_{i/o} = f_{og}/s_n$, with f_{og} equal to the original data sampling rate, and s_n equal to the stride. In this paper a stride, $s_n = 5$ is used, resulting in an input-output mapping sampling rate $f_{i/o} = 20Hz$.

C. Network Model Structures

Many different network structures, capable of performing time sequence predictions, exist. In this paper four different networks will be presented: a three-layer perceptron model (MLP), a single-layer long short-term memory (LSTM), a three-layer deep LSTM, and a single-layer encoder-decoder network [15, 31, 32]. The reasoning for including each of these network model structures is addressed later. Building, training and testing the networks was done using the Keras API using Tensorflow [33].

1) Three-layer Perceptron Model: Figure 9 shows the structure of a three-layer MLP network, featuring three input nodes, three hidden layers, each featuring five hidden neurons and three output neurons. An MLP, often described as a vanilla neural network, is a fully connected, feed-forward network. Although information solely flows from the input to the output layer without information flowing back through the network [34], a vanilla MLP has shown to be competitive in temporal sequence regression tasks [15, 31]. For this reason the trained MLP will be used as network prediction benchmark, here.

In Figure 9, all the circles denote *neurons* with the lines connecting them called *synaptic weights* [32]. Each neuron is a processing unit mapping an input to a certain output. The equation for a single hidden layer can be defined as [32]:

$$h_{out} = \rho_h(\vec{W}_{l_{prev},h} \times \vec{l}_{prev} + b_{l_{prev}}) \tag{21}$$

Here h_{out} is the output vector of the hidden layer, and ρ_h is defined as the activation function of a neuron in layer h. The activation output is a function of the synaptic weight matrix connecting layer h with its previous layer, in this case $\vec{W}_{l_{prev},h}$, times the previous layer's output vector $\vec{l}_{prev}$ and a bias specific term. A neural network, featuring

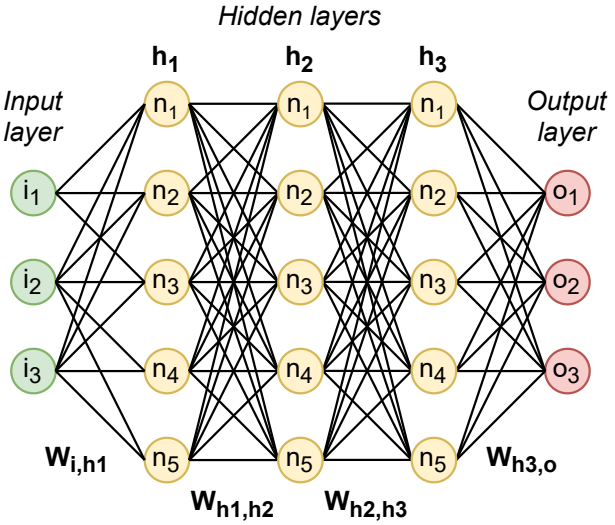


Fig. 9: Structure of a three-layer MLP network.

benchmark rectified linear unit (Relu) activation functions, i.e., $\rho(x) = x \; for \; x \leq 0$ and 0 otherwise, can suffer from a phenomenon called 'dead Relu's'. This results in zero-gradient flow through the network during training [35]. To omit this risk, the scaled exponential linear unit (Selu) is used in MLP layers throughout this research. The definition of the Selu is given in Equation (22) [36]:

$$\text{Selu}(x) = \begin{cases} \lambda x & \text{if } x > 0 \\ \lambda(\alpha e^x - \alpha) & \text{if } x \leqslant 0 \end{cases} \tag{22}$$

In Equation (22), λ and α denote slope tuning parameters. As can be seen in Figure 10, the gradient of a Selu activation function is non-zero for the input interval $x \in [-\infty, \infty]$, whereas the gradient of a Relu is only non-zero on the interval $x \in]0, \infty]$.

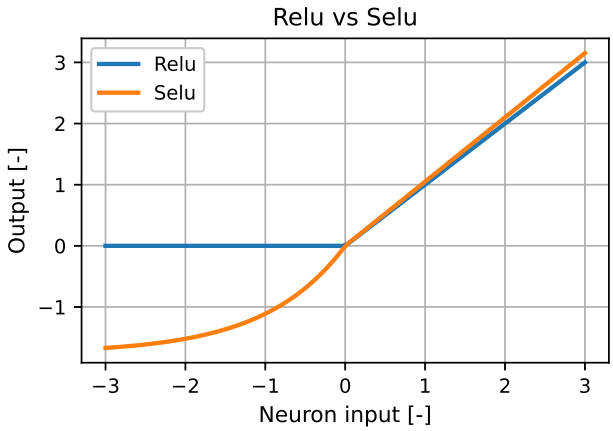


Fig. 10: Graph showing the difference between a Relu and Selu activation function, with $\lambda = 1.05$ and $\alpha = 1.67$.

2) Single- and Multi-layer LSTM Recurrent Neural Network: The second and third type of network structure are recurrent neural networks (RNN). Unlike a vanilla MLP, an

RNN is a feedback neural network, which means that information can be cycled back into the network. An illustration is given in Figure 11. Information feedback enables RNNs to detect patterns in data and perform (discrete) time series forecasting [32]. For this reason, RNNs are also deemed a suitable candidate to predict future, discrete, vehicle states $\vec{f}$ and $\vec{\omega}$.

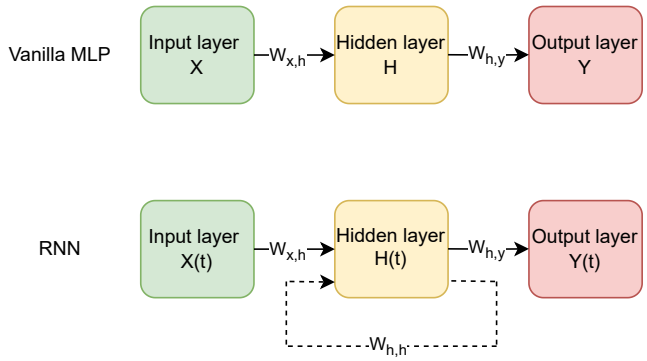


Fig. 11: Difference of information flow between a vanilla MLP and a RNN [32].

In Figure 11, the hidden layers are summarized into one block, which means that an RNN can feature multiple hidden layer cells. It is possible to *unroll* an RNN, resulting in Figure 12. This figure illustrates that the hidden layer actually consists of multiple RNN cells, denoted by H, each providing information to the next cell. Next to this, each cell is copied a set number of times, i.e., each hidden cell features the same topology and weight matrix.

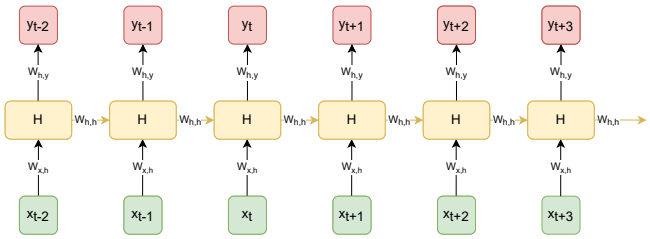


Fig. 12: Schematic of an *unrolled* RNN [32].

The RNNs in this paper are built-up with long short-term memory (LSTM) recurrent cells, because they are able to process information hundreds of time steps in the past without loss of stability [37]. The cell equations for an LSTM are given in Equation (23), with a schematic given in Figure 13.

$$\begin{aligned} f_t &= \sigma(W_{x,f} \times x_t + W_{h,f} \times h_{t-1} + b_f) \\ i_t &= \sigma(W_{x,i} \times x_t + W_{h,i} \times h_{t-1} + b_i) \\ o_t &= \sigma(W_{x,o} \times x_t + W_{h,o} \times h_{t-1} + b_o) \\ c'_t &= \tanh(W_{x,c} \times x_t + W_{h,c} \times h_{t-1} + b_c) \\ c_t &= f_t \cdot c_{t-1} + i_t \cdot c'_t \\ h_t &= o_t \cdot \tanh(c_t) \end{aligned} \tag{23}$$

The LSTM cell consists of a cell or memory state C_t, a hidden state h_t and three *gates*, f_t, i_t, and o_t. These gates

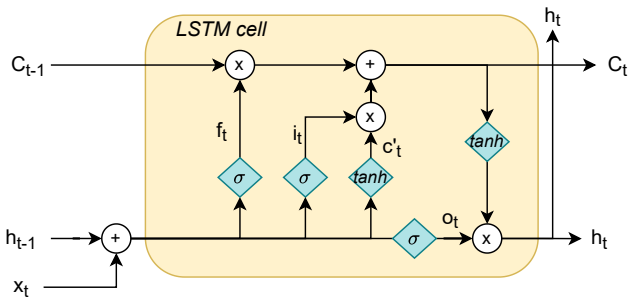


Fig. 13: Schematic of a long short-term memory cell [37].

enable the network to learn how to manipulate the stream of information consisting of the input x_t, the hidden state h_t and the cell state C_t. An interpretation for each gate is given below:

- *Forget gate* f_t: manipulates which information from the previous cell state C_{t-1} should be maintained/forgotten. A Sigmoid maps its input, i.e., addition of hidden state h_{t-1} and the input x_t, to the interval $[0, 1]$. A '0' implies the information should be forgotten, a '1' implies the information should be remembered. The output of the forget gate applies to the cell state, C_{t-1}, through point-wise multiplication.
- *Input gate* i_t: manipulates to which extent new information should be added to the cell state. First, a Sigmoid is used to update specific entries in the cell state C_t. A '0' implies the entry should not be updated, a '1' implies the entry should be updated. Then, the Sigmoid output is point-wise multiplied by a matrix of new candidate values c'_t, obtained by mapping its input, $h_{t-1}+x_t$, to the interval $[-1, 1]$ with a $\tanh$. Finally, the result is added to the cell state.
- *Output gate* o_t: decides which information from the cell state is output by the LSTM cell. The cell state values are mapped to a value range $[-1, 1]$ by applying a $\tanh$. By point-wise multiplication, a Sigmoid is used to either send the updated cell state values to the next cell/output, or to remove them from the hidden state/output h_t.

It is also possible to use the outputs of the sequence of LSTM cells as inputs to a new layer of LSTM cells. In doing so, one can create stacked LSTM layers, introducing the concept of a Deep LSTM network. The third network structure used in this paper is a three-layer Deep LSTM network.

3) Encoder-Decoder Neural Network: The final network structure is an encoder-decoder network, which is a type of LSTM RNN. A single-layer encoder-decoder RNN features two RNNs, one used for encoding the input information of arbitrary length to a fixed-dimensional vector, i.e., the encoder RNN, and the decoder RNN which maps this fixed-dimensional vector to the target sequence [38]. Unlike an LSTM RNN, the input information is fully processed by the encoder before the decoder maps it to the target sequence, this has the benefit that always all information is used before a prediction is made [38]. Also, both the encoder and decoder have their own trainable weighting parameters, giving the

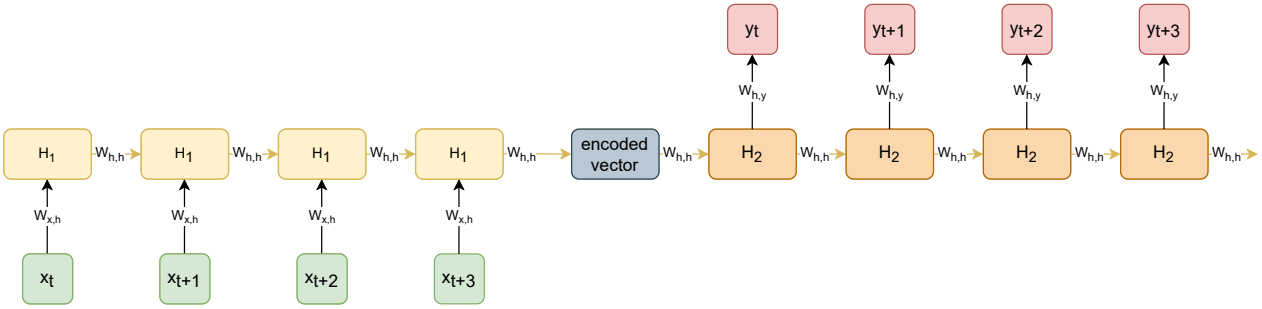


Fig. 14: Schematic of an encoder-decoder RNN [37–39].

network more learning flexibility. A representation of the encoder-decoder RNN is given in Figure 14.

In Figure 14, H_1 denotes the hidden cell of the encoder RNN, and H_2 denotes the hidden cell of the decoder RNN, both featuring LSTM cells similar to Figure 13. Although the encoder and decoder can feature more RNN layers, in this contribution only a single-layer shallow encoder-decoder network is trained.

D. Generalization

Each of the four networks feature the same final three layers. As an example, the schematic of the Deep LSTM RNN network structure is given in Figure 15, the rest of the network structures are presented in the Appendix to this document. In this figure the third-to-last layer is a dense layer 'interpreting' the network output from, i.e., the three-layer dense, single-layer LSTM, three-layer Deep LSTM, and encoder-decoder RNN, to project it to each of the six output features. A dropout generalization layer, balancing between an under- and overfitting network [40], is used to connect them together.

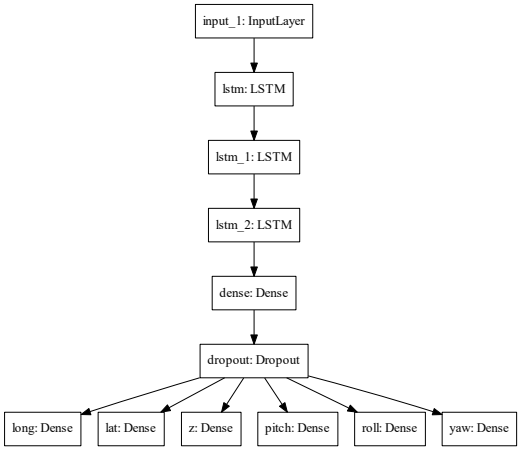


Fig. 15: Schematic of the Deep LSTM network structure.

When dropout is applied to a layer of a network, neurons are randomly dropped during training with probability p. Effectively meaning that for each training iteration a different network structure is trained, i.e., an efficient form of ensemble optimization [40]. At test time, the fully connected network is utilized, with the final weights equal to the optimized value multiplied by p. An example of dropout, applied to a one-layer MLP, with dropout probability $p = 2/5$, is shown in Figure 16. Due to dropout, both hidden neurons h_2 and h_3 have been deactivated during a specific training iteration.

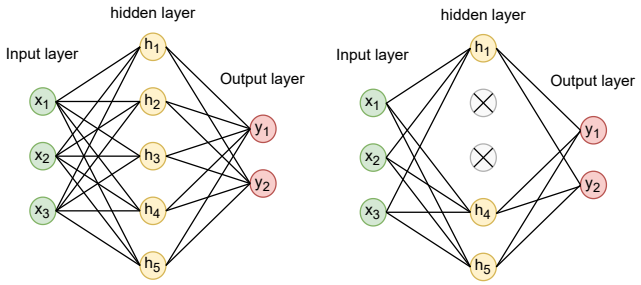


Fig. 16: Example of dropout regularization applied to only the hidden layer of a single-layer MLP, $p = 2/5$ [40].

IV. Methods

A. Simulator

The simulator used for open-loop MPC analysis, shown in Figure 17, is BMW's largest dynamic driving simulator, the Sapphire Space. The Sapphire Space is a 9-DoF simulator consisting of an XY-Drive, on top of which a large 6-DoF, $\pm 1.4m$ actuator stroke hexapod is placed, featuring a top-mounted yawdrive. In the present work only the 6-DoF hexapod of the full motion system is considered. The workspace and actuator limits are presented in Table IV.

TABLE IV: Sapphire Space specifications.

Hexapod					
x_H	$\pm 1.2m$	$\dot{x}_H$	$\pm 1m/s$	$\ddot{x}_H$	$\pm 10m/s^2$
y_H	$\pm 1.2m$	$\dot{y}_H$	$\pm 1m/s$	$\ddot{y}_H$	$\pm 10m/s^2$
z_H	$\pm 0.8m$	$\dot{z}_H$	$\pm 1m/s$	$\ddot{z}_H$	$\pm 8m/s^2$
ϕ_H	$\pm 26°$	$\dot{\phi}_H$	$\pm 20°/s$	$\ddot{\phi}_H$	$\pm 150°/s^2$
θ_H	$\pm 25°$	$\dot{\theta}_H$	$\pm 20°/s$	$\ddot{\theta}_H$	$\pm 150°/s^2$
ψ_H	$\pm 36°$	$\dot{\psi}_H$	$\pm 20°/s$	$\ddot{\psi}_H$	$\pm 150°/s^2$
Hexapod Actuators					
q_{act}	$\in [3.17, 4.58]m$	$\dot{q}_{act}$	$\pm 1.01m/s$	$\ddot{q}_{act}$	$\pm 8.47m/s^2$
XY-Table					
x_{XY}	$\pm 9.55m$	$\dot{x}_{XY}$	$\pm 6m/s$	$\ddot{x}_{XY}$	$\pm 6.5m/s^2$
y_{XY}	$\pm 7.85m$	$\dot{y}_{XY}$	$\pm 6m/s$	$\ddot{y}_{XY}$	$\pm 6.5m/s^2$
Yawtable					
ψ_{yaw}	$\pm 180°$	$\dot{\psi}_{yaw}$	$\pm 40°/s$	$\ddot{\psi}_{yaw}$	$\pm 150°/s^2$

Fig. 17: Sapphire Space. Source: [16].

B. MPC Settings

1) MPC Reference: Three different MPC references will be used in the open-loop analysis, i.e., a constant, oracle, and neural network reference:

(a) *Constant Reference*: in this case it is assumed that the current, measurable state of vehicle accelerations and rotational rates remains constant over the prediction time T_p. A prediction time of $T_p = 1.5s$ is used, as per [11]. In [11], it is shown that, using short prediction times, a constant reference provides better motion cueing compared to the industry standard filter-based MCAs [11]. In order to reduce MPC's computational complexity, MWB is applied. This results in a reduced amount of control variables (60 instead of 150) with $\Delta\bar{U}$ defined on the interval $s = [50, 10]$ with sampling time $t_s = [0.01s, 0.1s]$. The constant reference MPC results are used as the feasible benchmark in the MPC MCA performance.

(b) *Neural Network Reference*: resulting from the neural network performance comparison, the best performing network will be used to generate an open-loop reference trajectory for the MPC MCA. Assuming the chosen neural network is able to provide better predictions than a constant reference, a prediction time of $T_p = 4s$ is used [8]. Applying MWB results in a reduced amount of control variables (70 instead of 400) with $\Delta\bar{U}$ defined on the interval $s = [50, 10, 10]$ with sampling time $t_s = [0.01s, 0.1s, 0.25s]$.

(c) *Oracle Reference*: here it is assumed that all future vehicle states are known a priori. This case is only possible in open-loop simulation and serves as an upper limit regarding prediction quality. Since all future motions are known, no theoretical upper limit in prediction time exists. However, to enforce a fair comparison, it was chosen to use the same prediction time used for the neural network reference, i.e., $T_p = 4s$, allowing for the same MWB strategy to be applied. The oracle reference MPC results are used as the 'most optimal' benchmark in MPC MCA performance.

2) Limiting MPC Optimization Iterations: The MPC formulation, as defined in Section II, does not require to run real-time. However, for time constraint purposes, the available optimization time is limited. According to qpOases [24], the default amount of iterations is specified by $Iter = 5(nV + nC)$, with nV equal to the amount of optimization variables and nC the amount of constraints. This amounts to $62,400$ maximum possible iterations. In order to reduce the MPC optimization runtime, the amount of optimization iterations per MPC iteration is reduced by a factor of 10 from the default value, i.e., to $6,240$. If a solution cannot be found within the amount of iterations, the previous solution is applied to the system.

3) MPC Weight Settings: MPCs cueing performance is heavily influenced by the chosen weight settings [7]. Two different weight settings will be used in the open-loop simulations to show the weight influence on motion cueing fidelity.

The first weight setting is retrieved from [7]. This setting was used in a real-time MPC used on a 9-DoF, hexapod and tripod, kinematic configuration called the Ruby Space. Since the overall translational limits between the hexapod of the Sapphire and the Ruby Space are similar, this set of weights are deemed a valid starting point. Still, adjustments to the weights are made based on the following two considerations. First, in case of the Ruby, the translational specific forces and yaw rate are cued by using the independent tripod, whereas tilt-coordination cues are provided by the hexapod system. In case of the hexapod of the Sapphire, motions cannot be independently cued, reducing the realistic available workspace in each DoF. Second, to keep the optimization process linear, it was noted in Section II that the transformation matrices, required to perform state estimation, were kept constant for future time steps. As argued in [7], this assumption becomes less valid for large angular displacements within the prediction horizon N_p, e.g., costly large yaw excursions. Due to the non-linear correlations between the hexapod angular displacement and the available actuator workspace, this assumption poses the risk that the optimization process finds solutions that do not adhere to the actual actuator limits. If, due to excessive simulator inputs, a breach of constraints occurs, the MPC MCA optimization process cannot recover.

Based on these two considerations, the state weights Q_x, affecting both yaw (most expensive angular displacement) as well as translational motion are increased by a factor of 4. The state weights in Q_x, affecting pitch and roll (less expensive angular motion), are scaled up by a factor 1.25. For stability purposes it was also decided to increase the weights on the change in input, Q_{du}, for translational accelerations, by a factor of 10. This reduces the possibility of high fluctuations between subsequent QP solutions, therefore reducing jerky motion cues. Since yaw motion is an expensive motion for a hexapod system, only a conservative 60% of the yaw reference is provided to the MPC. If no large motion space yaw system is available, e.g., a $\pm 180\,\mathrm{deg}$ yaw drive, a scaling factor in the order of $[40\text{-}60\%]$ can be found in literature [7, 8]. The other directions are not scaled down. The first set of weights, W_1, can be found in Table V.

The second setting, W_2, used in the analysis, is the same set as provided in Table V, but with the neutral push 80% of the original set, i.e., $Q_{x,W2} = 0.8 \cdot Q_{x,W1}$. It should be noted that, although a cost reduction of 20% is severe, the ratio

between absolute costs is the driving factor [41]. This means that, per definition, the simulator is not allowed to use exactly 20% more actuator stroke. Therefore, it is expected that the simulator will be able to use more of its workspace. However, due to the other involved costs and natural system limits, the effect of the more mild neutral push might be marginal.

C. Hyperparameter Tuning

The presented neural network models feature many different parameters, these can be divided into two categories: trainable, and manually adjustable parameters. The manually adjustable parameters, i.e., hyperparameters, are the parameters set by the designer. These need to be assigned values to ensure good model performance. The hyperparameters can be divided into three different categories:

1) *Structure hyperparameters*: These hyperparameters define the structure of the network. Parameters included are: amount of hidden layers, neurons per layer, neuron activation functions and for which layers regularization applies.
2) *Process hyperparameters*: These hyperparameters have a direct influence on the training process and therefore, implicitly, an influence on the network's performance. Parameters included are: learning rate, regularization parameters (such as the dropout rate), optimization algorithm parameters, the batch size and the chosen metric for training and validation loss.
3) *Data hyperparameters*: These hyperparameters have an influence on the data, how it is structured and how the input and output sequences are devised. Parameters included are: resampling frequency, look-ahead distance, stride, scaling, and length of input and output sequence.

In order to choose the set of parameters that will be used in the networks, a brute-force technique called *grid search* is employed [42]. *Grid search* comprises of two steps. The first step is to set-up an initial, wide discrete search space for the different parameters that need to be optimized. The search space is divided into five equidistant parts. For all the possible combinations within that search space, the network is trained for 10 epochs, the parameter subset that gives the best validation error is selected. The second step is to set-up a more narrow, 10% wide search space around the selected parameter subset, also using five equidistant values, repeating the same optimization process as mentioned before. These final set of values are then used to set-up the final network structures. Since this method does not scale well with the amount of hyperparameters that need to be tuned, only learning rate, amount of neurons/cells per layer, batch size and dropout rate are tuned, the rest were, as discussed previously, kept at a fixed value, see Table VI. The initial heuristic values are given in Table VII.

TABLE V: One of two weight matrix values, i.e., W_1, used for the open-loop MPC analysis [7].

Weighting Matrix Q_y					
f_x	*0.1*	f_y	*0.1*	f_z	*0.1*
ω_x	*1*	ω_y	*1*	ω_z	*1*
Weighting Matrix Q_x					
p_x	*0.38*	p_y	*0.38*	p_z	*0.6*
$\dot{p}_x$	*0.11*	$\dot{p}_y$	*0.11*	$\dot{p}_z$	*0.3*
ϕ	*7.5*	θ	*7.5*	ψ	*7.5*
Weighting Matrix Q_u					
$\ddot{p}_x$	*0.01*	$\ddot{p}_y$	*0.01*	$\ddot{p}_z$	*0.5*
$\dot{\phi}$	*40*	$\dot{\theta}$	*40*	$\dot{\psi}$	*1*
Weighting Matrix Q_{du}					
$\Delta\ddot{p}_x$	*5*	$\Delta\ddot{p}_y$	*5*	$\Delta\ddot{p}_z$	*0.5*
$\Delta\dot{\phi}$	*40*	$\Delta\dot{\theta}$	*40*	$\Delta\dot{\psi}$	*1*
Weighting Matrix Q_t					
W_t	*250*				

TABLE VI: Fixed set of hyperparameter values used for networks.

	Network Structure			
Hyperparameter	***MLP***	***LSTM***	***Deep LSTM***	***Enc-Dec LSTM***
# Layers	*3*	*1*	*3*	*1*
Dense Activation	*Selu*	*Selu*	*Selu*	*Selu*
Dropout Layer	1	1	1	1
Data Resampling	*Log*	*Log*	*Log*	*Log*
Stride	*5*	*5*	*5*	*5*
No. In/out Samples	*30*	*30*	*30*	*30*
Look-Ahead	*100m*	*100m*	*100m*	*100m*
Data Scaling	*MinMax*	*MinMax*	*MinMax*	*MinMax*
Input Sequence	*5s*	*5s*	*5s*	*5s*
Output Sequence	*4s*	*4s*	*4s*	*4s*

TABLE VII: Heuristic grid search value ranges used for optimization of selected hyperparameters.

Hyperparameter	**Initial value range**
Learning rate	$[5e^{-4} - 1e^{-2}]$
Dropout rate	$[0.3 - 0.7]$
Batch size	$[16 - 128]$
Neurons MLP	$[128 - 1024]$
Cells LSTM	$[64 - 512]$
Cells Deep-LSTM	$[64 - 512]$
Cells Enc-Dec LSTM	$[64 - 512]$

The two-step hyperparameter optimization process results in the crudely optimized parameter values of Table VIII:

TABLE VIII: Optimized set of hyperparameter values used for the different neural network structures.

	Network Structure			
Hyperparameter	***MLP***	***LSTM***	***Deep LSTM***	***Enc-Dec LSTM***
Learning Rate	*1.44e-3*	*1.44e-3*	*1.44e-3*	*1.44e-3*
Neurons/Cell	*128/256/128*	*256/256*	*128/256*	*128/256*
Dropout Rate	*0.5*	*0.5*	*0.5*	*0.5*
Batch Size	*128*	*128*	*128*	*128*

In this table, the neurons/cells depict the amount of neurons and cells per layer of the proposed network. In case of the RNN structured networks, the last value, 256, denotes the amount of neurons in the input-output connecting dense layer. The total amount of neural network trainable parameters equal to: 219,956 for the three-layer dense MLP, 289,556 for the single-layer LSTM RNN, 284,084 for the encoder-decoder network, and 415,668 for the three-layer Deep LSTM network.

D. Training Neural Networks

Training neural networks effectively means altering the weights such that the error between the network's predicted output and the labelled (true) output is as small as possible. This optimization task, for a vanilla MLP, can be formulated as follows [35]:

$$Minimize \;\; E = \frac{1}{N}\sum_{i=1}^{N} L_i(f(x_i, \vec{W}), y_i) + \lambda \cdot R(\vec{W}) \quad (24)$$

Here, N depicts the amount of training input-output mappings that are used, L_i is defined as the loss function, which is a function of both $f(x_i, \vec{W})$, i.e., the output of the network (function of input x_i and weights $\vec{W}$), and the true output y_i. The added term $\lambda \cdot R(\vec{W})$ is a regularization penalty, where λ is a weighting term that is used to trade-off between model performance on training data and obtaining a simpler neural network [43, 44]. Often the simpler model has a better fit on unseen data than a complex model which tends to overfit [45]. Since the structures of an RNN, Deep LSTM and an encoder-decoder network are different from a vanilla MLP network, the optimization task is different as well. The loss function can be written as a sum of all individual loss terms, i.e., sum of each loss for each output [32]:

$$Minimize \;\; E = \sum_{t=1}^{T} L_i(Y_t, \hat{Y}_t) \quad (25)$$

Here, T is the total amount of outputs in the output sequence, Y_t the true values at time t and $\hat{Y}_t$ the predicted values at time t. After defining the loss function, the gradient of the loss, with respect to the output, can be calculated. The RNN optimization is usually called *backpropagation through time* (BPTT) due to the time dependence of each individual loss function [32].
For all models, the loss function is determined to equal the mean-squared-error loss: a loss function that weights extreme errors more than smaller deviations.

All network models are trained using the efficient and widespread *Adapative momentum estimation* (ADAM) optimizer. ADAM is an advanced backpropagation algorithm often used for optimizing machine learning networks due to its efficient solving capabilities [46]. Backpropagation performs weight optimization by computing the gradient of the error with respect to the output. By applying the chain rule it is then possible to formulate an expression for the change in error with respect to each input and each individual weight, $\delta E/\delta w$ [47].

ADAM combines two other algorithms, firstly Adagrad [48] which accelerates optimization in dimensions with small gradients, holds back optimization in dimensions with large gradients by introducing an implicit learning rate decay factor (with bad parameterization this can lead to an optimization stop). Secondly, RMSprop [47] which is similar to Adagrad but adds in an explicit decay factor which reduces the implicit learning rate decay over time. The result is an algorithm which includes ideas from momentum (building up a velocity term) as well as ideas from RMSprop (acceleration in dimension with small gradients and vice versa without strong learning rate decay). Decaying bias terms are added to minimize the null initialization effect (a large first update step). The mathematical background can be found in [46].

Before the training process can start, all the weights in the network need to be initialized to retrieve an initial gradient value. The Glorot uniform initialization was used [49], where for each neuron a value is generated by sampling from a uniform distribution with limits dependent on the amount of inputs and outputs of that weight tensor. Each of the four networks was trained for 100 *epochs* incorporating an early stopping strategy, i.e., training is stopped if the validation error does not decrease significantly in the next 5 *epochs*.

E. Neural Network Driver Prediction Metrics

A framework is required to understand the performance of the four different networks. The performance for each of the neural networks' most important reference outputs (longitudinal, lateral acceleration, and yaw rate) will be analyzed [8, 11, 41]. Three different methods will be used: an analysis on the mean-squared-error (MSE) over time signal per DoF, an analysis on the average mean-absolute-error (MAE) over the prediction horizon N_p for all neural network predictions, and a qualitative analysis on three random time sequence predictions. The different methods are elaborated upon in the following segment.

For the three drives in the test-set, the MSE per time instance per output, for the prediction time of $T_p = 4s$, will be computed and evaluated. The MSE is a metric often used for neural network regression performance analysis, because it weights large prediction errors more heavily than smaller ones [35]. The MSE per time instance per DoF for a constant prediction over the same prediction time, $T_p = 4s$, will be computed as well, giving a benchmark prediction quality metric. This analysis will provide a deep insight on how the prediction performance alters during the scenario, and for which maneuvers the neural network prediction shows a significant improvement over a constant prediction strategy. In this analysis, it is expected for all neural networks to show significantly lower MSEs during dynamic maneuvers since a constant prediction in these cases is per definition a bad predictor.

Next to this, the best performing neural network, based on the previously discussed metric, as well as the constant prediction will be analyzed using the average MAE and MAE standard deviation (σ) over all samples in the prediction horizon of $N_p = 400$ samples. The MAE will be computed for one drive in the test-set, averaging all MAEs of all predictions. This analysis will provide information on the average regression quality over N_p. It was chosen to use the MAE because the unit of the error is the same as the unit of the predicted signal, making the results better interpretable [35]. It is expected that for the first samples in the prediction horizon N_p, the constant prediction outperforms the neural network on both the average MAE as well as providing a lower σ in prediction MAE. However, it is also expected that the constant prediction and σ MAE continuously increase over N_p. Assuming long-term predictions are less accurate than short-term predictions, i.e., due to the logarithmic output spacing and subsequent interpolation, the neural network MAE is also expected to increase over N_p, albeit at a lower rate.

As will be shown in the analysis later, one problem with neural network predictions is that there may exist a mismatch

between the initial predicted state and the actual current state. This behavior will become apparent when the neural network MAE shows a significantly higher σ for the first samples. In order to solve this, a half-period cosinebell function, see Figure 18 [50], is proposed to 'anchor' the predictions to the actual state. This function smoothly 'fades' the current state into the predicted state. Its effectiveness will be revisited later in Section V.

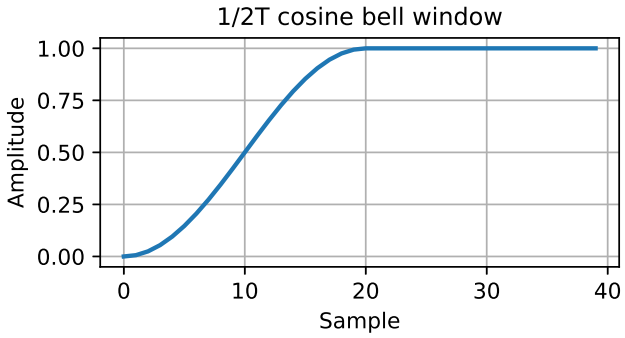


Fig. 18: Half-period cosinebell function with periodicity equal to 41 samples [50].

Finally, to understand how the network behaves on prediction level itself, prediction sequences at three random time instances will be qualitatively analyzed as well.

F. MPC Motion Cueing Quality Metrics

When the best performing neural network is found, prediction results can be used as reference in an open-loop MPC simulation. Two methods proposed in literature are used in order to estimate the resulting MPC cueing quality [13, 14].

1) Subjective Motion Incongruence Estimation: This method finds its basis on the work done in [13, 14], in which a model is proposed, able to make a validated estimation on the *continuous motion incongruence ratings* (MIR) in an urban driving scenario. This model was developed to provide a means of estimating subjective MCA cueing performance based on open-loop simulations without the requirement of an expensive driver-in-the-loop (DIL) experiment. The output of the model is a time signal, i.e., the continuous rating, having a lower bound of 0 and a set upper bound of 10. Lower ratings denote less motion cueing incongruence, i.e., better perceived motion cueing performance. Larger values denote a worse perceived motion cueing performance. This model only requires open-loop processed cueing information.

In [14], it was shown that the rating behaviour of participants can be modelled conform a time-delayed, low-pass filtered, and weighted combination of both longitudinal, as well as the lateral specific force mismatches. First, the total weighted error contribution is calculated by [14]:

$$E(t) = W_{f_x}\left(|f_x^v - f_x^s|\right) + W_{f_y}\left(\left|f_y^v - f_y^s\right|\right) \tag{26}$$

The weights W_{f_x} and W_{f_y} denote the relative weighting contribution, these are set to 1.17 and 1.63, respectively [14]. The predicted continuous rating is found when applying a low-pass filter to $E(t)$, defined in:

$$H_{x,y}(j\omega) = \frac{\omega_{c_{x,y}}}{j\omega + \omega_{c_{x,y}}} e^{-j\omega\tau} \tag{27}$$

Here, $\tau = 0.008s$ it the time-delay constant, $\omega_{c_x} = 0.26rad/s$ and $\omega_{c_y} = 0.37rad/s$ denote the cut-off frequencies for the x and y channels [14]. Since no tuned rural model exists, these tuned settings are assumed to be a valid approximation since it only considers longitudinal and lateral specific force perception.

2) Motion Cueing Error Type Analysis: Next to this analysis, a motion cueing error analysis, as proposed in [14], is performed. In this analysis four different types of motion cueing errors are investigated comprising of false direction cues, false cues, missing cues, and scaling errors. A schematic showing all four types of errors is given in Figure 19.

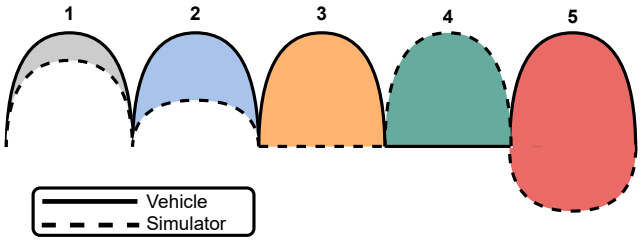


Fig. 19: Schematic showing the four types of cueing classification errors. **1**: No cueing error, **2**: scaling error, **3**: missing cue error, **4**: false cue error, and **5**: false direction cue error. Source: [14].

First, for each of the specific forces and rotational rates, an error classification is performed. This results in a time signal per channel giving the type of error at each time instance k. It should be noted that only one type of error can be present at a single time instance. Using the time signal per channel, the frequency at which a specific error occurs, expressed by a percentage in function of total simulation time, can be computed. As explained in [51] and [52], a perception threshold of $f_{s_t} = 0.05m/s^2$ and $\omega_{s_t} = 3deg/s$ is implemented for both the perceived specific forces and rotational rates. Errors lower than the perception threshold are assumed to be cued correctly and not reported. Therefore, the sum of all cueing error percentages does not need to equal 100%.

G. Expectations

Following the presented analysis methods, the following expectations are made.

Regarding the neural network models, all neural network models are expected to outperform the constant prediction approach (i.e., have a lower average MSE). The expected performance of the networks is as follows (from worst to best): the three-layer MLP, the single-layer LSTM RNN, the encoder-decoder RNN, and finally the Deep LSTM RNN. Where it is expected that the more complex networks, i.e., encoder-decoder and Deep LSTM, perform better. Since longitudinal acceleration is generally more difficult to predict than lateral acceleration, due to more variability between participants

[7], it is expected that the neural network prediction quality improvement will be lower for longitudinal acceleration.

It is also expected that assuming a constant prediction is not bad for the first few samples in N_p because of high vehicle inertia and a high data refresh rate. However, the MAE will most likely keep increasing over N_p together with the MAE standard deviation, showcasing that depending on the road conditions, e.g., long straight with no curves (low dynamic changes) vs. a roundabout (high dynamic changes), a constant prediction is not a consistent strategy. The neural network prediction is assumed to start outperforming the constant prediction after the first n samples, also showing a lower σ over N_p, proving the neural network strategy to be more consistent. However, since the neural network prediction is per definition not anchored to the current state, oscillatory behavior for the first samples is expected, which is undesirable. For this reason, the prediction most likely needs to be anchored to the current state using a half-period cosinebell function.

Because the output mapping is logarithmically spaced, it is expected that the neural network is able to predict low frequency trends at the tail of the prediction sequence, not being able to capture more high dynamic changes. In contrast, it is expected that the neural networks are able to predict the more high-frequency short-term dynamic trends.

Assuming improved prediction quality, using the prediction of the best network in an MPC MCA framework, it is expected that the estimated motion cueing performances will be somewhere in between the performance of a constant MPC and an oracle MPC. This is also assumed to be the case for motion cueing errors, where it is expected that the amount of detrimental cues will be lowest for (from best to worst): oracle MPC, Deep LSTM MPC, and constant MPC. Because of the limited and costly yaw range of the hexapod system, together with the scaled down yaw rate reference, it is expected that the cued yaw rate of all three reference strategies will show large missing cue errors.

Since W_1 features a stronger neutral push than W_2, it is expected that the MPC MCA using W_1 will show larger and more false direction cues than the one tuned with W_2. Next to this, it is also expected that the subjective motion cueing rating, for the MPC using W_2 for all three references compared to the MPC tuned with W_1, will be better.

V. Results

A. Neural Network Performance Analysis

In order to make a decision about which neural network predictions to use for the MPC, the individual performance metrics are assessed. As discussed in Section IV the first metric that will be analyzed is the MSE per time instance for both test set drives. Figure 20 shows the MSE over time for all four neural networks, compared to a constant prediction metric. The MSE per time instance was calculated on the interpolated prediction data over the prediction horizon, $T_p = 4s$ for both the neural networks, as well as the constant prediction. Although the constant reference in MPC has a prediction time equal to $T_p = 1.5s$, it was argued that to fairly compare the neural network prediction quality to the constant prediction quality, the same prediction time, $T_p = 4s$ needs to be applied. In Figure 20, it can be seen that in case of longitudinal acceleration, large constant MSE error peaks occur between speed limit transitions. In case of lateral acceleration and yaw rate, two distinct constant MSE peaks are present, the first one occurring in road segment 8 and the other in segment 2. These peaks belong to two medium speed sharp turns. No such large peak is present in segment 6 where a roundabout is located. Also here an MSE peak is expected due to a large radius of curvature. However, the reported MSE signal is obtained from a drive in SN direction, where the first exit is taken, reducing the radius of curvature significantly. In this comparison it can also be seen that, of the three predicted DoFs, the Deep LSTM has the lowest prediction error of all networks. An average MSE for the three test drives, and an MSE percentage improvement compared to the constant prediction, is given in Table IX.

From Table IX it can be concluded that, overall, the Deep LSTM prediction performs the best showing a reduction in MSE of approximately 65% in longitudinal and 90% in lateral acceleration and yaw rate prediction. The difference between the LSTM, Deep LSTM and Encoder-Decoder networks is small in lateral acceleration and yaw rate prediction, however, a drop in performance of around 5% can be noticed for the LSTM and Encoder-Decoder network. The worst performing network, although still better than the constant prediction, is the simple three-layer Dense neural network, showcasing an improvement of 42%, 58% and 67% in longitudinal, and lateral acceleration and yaw rate prediction, respectively.

Although analyzing the MSE over time through Table IX and Figure 20 proves to be insightful, it does not show the full picture. It is possible that individual time point prediction errors are averaged out on the MSE calculation interval $t_k \in [k, k + T_p]$. Figure 21 shows three time trace predictions, for both a constant, as well as a Deep LSTM, for the accelerations in *x* and *y* direction, as well as the yaw rate. As expected, one can see that for very short-term predictions, i.e., the first $0.01s$ in T_p, the constant assumption holds true, however evidently, it does not capture any dynamics in the future. The Deep LSTM, as expected, does follow the short-term trends better as well as capturing low frequency long-term trends, without capturing far future more high-frequency trends. It can also be seen that the Deep LSTM time trace predictions are not anchored to the current state, which creates an offset in prediction for the first time sample in N_p. To understand the specific quantitative effect of what can be qualitatively seen in Figure 21, the average MAE is investigated for all time samples within the prediction horizon $N_p = 400$ samples.

Figure 22 shows the average MAE for each predicted time sample for both a constant as well as a Deep LSTM lateral acceleration prediction for one of the three test drives. From this figure it can be seen that a constant prediction predicts lateral acceleration better in the first 15 samples, after which the average constant MAE keeps increasing while the Deep LSTM MAE starts to stabilize. This follows the proposed expectation that due to vehicle inertia and a high refresh rate in the data, assuming constant vehicle states within the first 15 samples is generally a valid assumption. Whereas

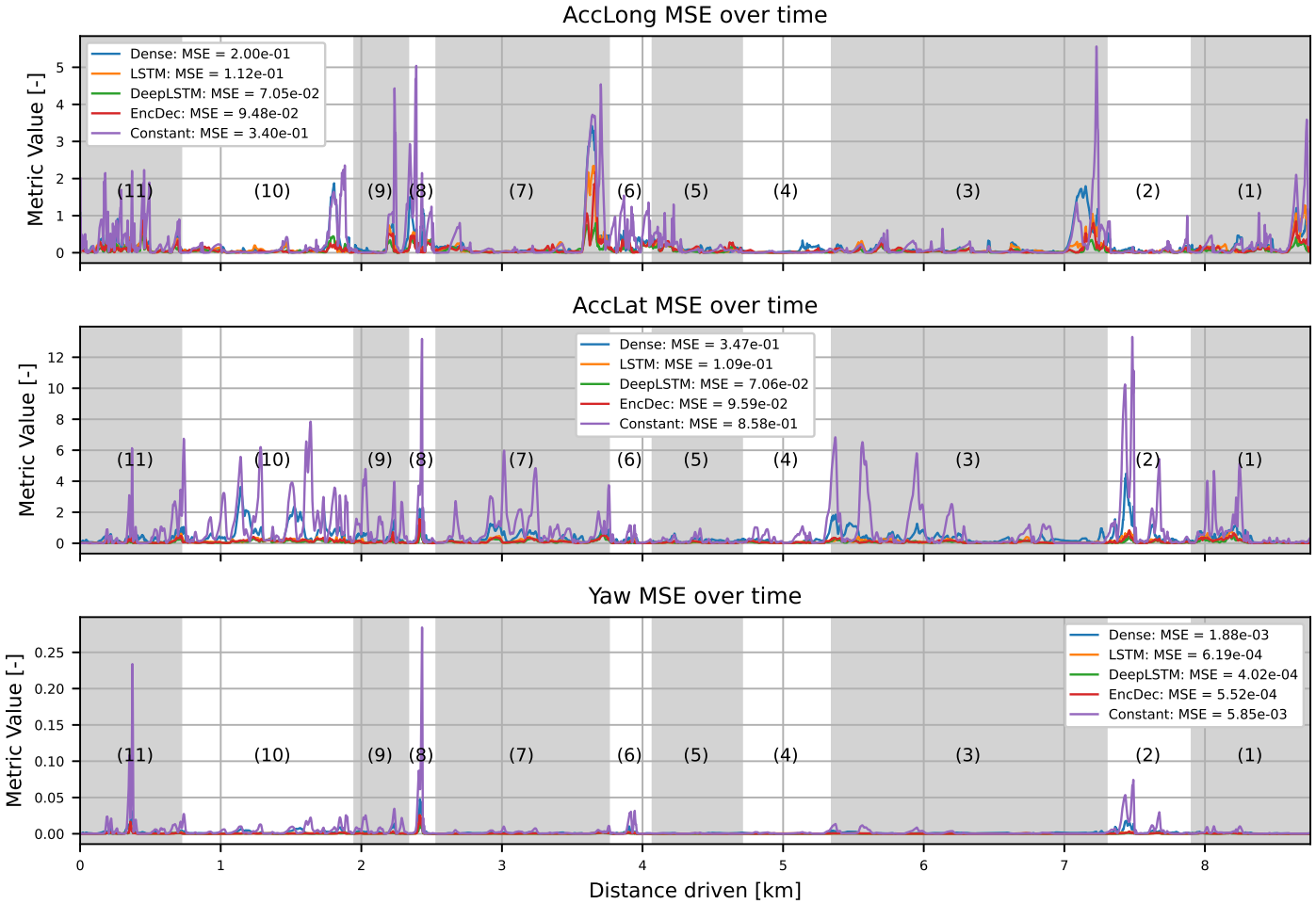


Fig. 20: MSE over time plot for a SN test drive for predicted longitudinal and lateral acceleration as well as predicted yaw rate. Prediction horizon for neural network models $N_p = 4s$, and for the constant prediction $N_p = 4s$.

TABLE IX: Average MSE prediction results comparing average NN MSE with average constant MSE for $N_p = 4s$.

	MSE					*% improvement wrt. Constant*			
	Constant	*Dense*	*LSTM*	*Deep LSTM*	*EncDec*	*Dense*	*LSTM*	*Deep LSTM*	*EncDec*
Acceleration in x	*3.90E-01*	*2.24E-01*	*1.63E-01*	*1.41E-01*	*1.58E-01*	*-42%*	*-59%*	*-65%*	*-60%*
Acceleration in y	*6.70E-01*	*2.79E-01*	*9.70E-02*	*7.93E-02*	*8.70E-02*	*-58%*	*-85%*	*-88%*	*-87%*
Yaw Rate	*5.03E-03*	*1.66E-03*	*6.03E-04*	*5.08E-04*	*5.42E-04*	*-67%*	*-88%*	*-90%*	*-89%*

the Deep LSTM prediction does not explicitly anchor itself to the current vehicle state, thus featuring altering offsets in between subsequent predictions for the first samples in N_p, resulting in a worse predictor. When used as reference in MPC the first 15 samples, especially when these feature high variance in between samples, affect the tuning of Q_{du}, i.e., the weights on the change in input. When high variance is present, higher weighting on the change in input, in order to reduce oscillations in applied input, is required. To limit this behavior, a half-period cosinebell window function is implemented on the first 20 samples of the predicted Deep LSTM sequence, which provides a smooth transition from the current state to the predicted sequence without loss of information [50]. Figure 22 shows the result of incorporating such window function on the first 20 samples, which confirms that the MAE decreases significantly in the first 20 samples compared to the original Deep LSTM prediction, while also retaining long-term prediction quality.

Table X shows the average MAE for six equidistant time samples within $N_p = 400$ samples. Starting from $t_1 = 0.01s$, to each 80th sample onwards, i.e., $t_{80} = 0.8s$. In this table the MAE as well as the standard deviation is given. It can be seen that for all samples, except the first sample, both the MAE and standard deviation of the Deep LSTM prediction are lower compared to the constant prediction. After applying the 20-sample window function, the same MAE and standard error as the constant prediction is obtained for the first sample.

What can also be seen from both Figure 22 and Table X, is that the average MAE of the Deep LSTM prediction is already lower than that of the constant prediction after approximately the 8th sample in N_p. Next to this, considering the general prediction time of $T_p = 1.5s$ used for the constant prediction

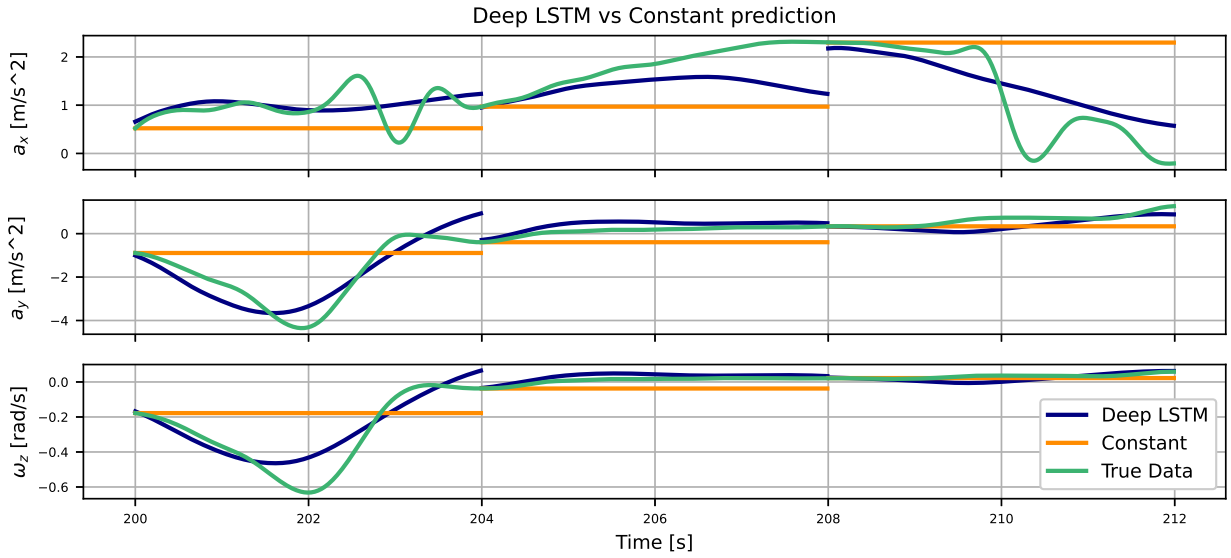


Fig. 21: Three time traces comparing the Deep LSTM with a constant sequence prediction for one of the three test drives.

TABLE X: Constant and Deep LSTM prediction MAE and σ results for five time instance predictions 0.8s apart.

	MAE			σ			*% improvement wrt. Constant*			
t_k	*Constant*	*Deep LSTM*	*Deep LSTM Cosinebell*	*Constant*	*Deep LSTM*	*Deep LSTM Cosinesbell*	*MAE Deep LSTM*	*σ Deep LSTM*	*MAE Deep LSTM Cosinebell*	*σ Deep LSTM Cosinebell*
0.01s	*0.005*	*0.053*	*0.005*	*0.007*	*0.065*	*0.007*	*960%*	*829%*	*0%*	*0%*
0.81s	*0.305*	*0.192*	*0.192*	*0.39*	*0.231*	*0.231*	*-37%*	*-41%*	*-37%*	*-41%*
1.61s	*0.462*	*0.21*	*0.21*	*0.526*	*0.238*	*0.238*	*-55%*	*-55%*	*-55%*	*-55%*
2.41s	*0.600*	*0.215*	*0.215*	*0.644*	*0.249*	*0.249*	*-64%*	*-61%*	*-64%*	*-61%*
3.21s	*0.718*	*0.23*	*0.23*	*0.73*	*0.262*	*0.262*	*-68%*	*-64%*	*-68%*	*-64%*
4.00s	*0.826*	*0.248*	*0.248*	*0.787*	*0.271*	*0.271*	*-70%*	*-66%*	*-70%*	*-66%*

[7], the Deep LSTM already provides an improvement in average MAE of around 55%. This shows, that even for shorter prediction times, a Deep LSTM, trained for longer prediction times, provides better prediction quality than the current constant prediction benchmark.

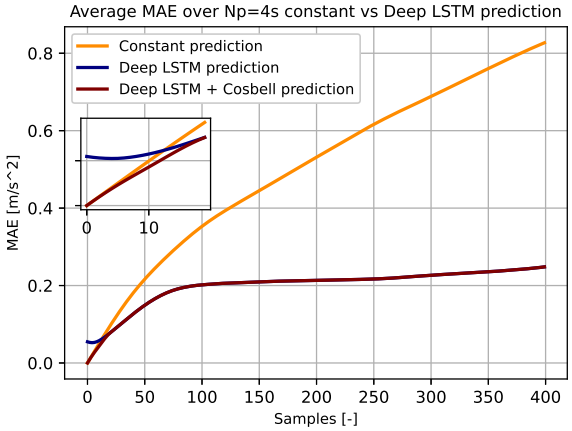


Fig. 22: Average mean absolute error constant and Deep LSTM NN lateral acceleration prediction for each time point in $N_p = 4s$.

The provided analysis suggests that the average MSE significantly improves using neural network predictions compared to a constant prediction over a prediction time, $T_p = 4s$. It was also shown that all neural networks outperformed the constant prediction approach, in following order (from worst to best): the three-layer MLP, single-layer LSTM, Encoder-Decoder RNN, and the Deep LSTM. Subsequently, a Deep LSTM, anchored to the current state using a half-period cosinebell function, provides more consistent predictions than a constant prediction over the full prediction window N_p. Where the former already shows a lower average MAE compared to the latter after the first 8 samples in N_p.
When considering the general constant MPC prediction time of $T_p = 1.5$, the $T_p = 4s$ trained Deep LSTM shows an average MAE prediction reduction of approximately 55% at $T_p = 1.5s$.

B. Predicted Continuous MIR

The three different test drives were analyzed using the evaluation model presented in Section IV, resulting in a predicted continuous MIR. Figure 23 shows part of the estimated MIR for one of the three test drives for an MPC using constant, Deep LSTM, and oracle reference. All three cueing results were obtained using MPC tuned with W_2, i.e., a lower neutral push. In this picture it can be seen that the estimated MIR for an MPC MCA using oracle reference is around 20% and 6% lower compared to the MPC using constant, and the Deep LSTM prediction as reference, respectively. With the estimated MIR using Deep LSTM MPC being around 16% lower than the estimated MIR using the constant MPC. This behavior

is confirmed when looking at the specific force and rotational rate graphs in Figure 24, where it can be seen that the majority of the original signal is cued by the simulator, albeit heavily scaled down because of the system's limits and the high cost on input Q_u. From this picture, it can also be seen that due to the limited yaw capabilities of the hexapod system, yaw cannot be cued to the same degree as accelerations in both *x* and *y* direction. Since yaw motion cueing quality is not considered in Equation (26), bad yaw rate tracking has no influence on the projected MIR values. Specific force and estimated MIR figures for the MPC MCA tuned with W_1, i.e., a stronger neutral push, are attached as Appendix to this document.

The same MIR prediction can be made for each of the three drives, using both weight setting W_1 as well as W_2. When combining the predicted ratings per reference type and weight setting, boxplots can be constructed showing the distribution of ratings per reference per weight. The resulting boxplots can be found in Figure 25, also showing the median rating value per scenario.

When investigating the results, one can see that the average ratings for W_2 are marginally lower compared to W_1 for each of the three reference types. This is an expected outcome as more of the simulator's actuator workspace can be used in the optimization process resulting in better motion cueing. Next to this, it can also be observed that the findings in Figure 23 are also valid when looking at the combined results of the three drives, i.e., oracle MPC outperforms Deep LSTM MPC which, in turn, outperforms the constant MPC. A summary of the provided boxplot information can be found in Table XI.

TABLE XI: Boxplot summary of Figure 25.

	Constant	**Deep LSTM**		**Oracle**	
	Value	Value	% Difference wrt Constant	Value	% Difference wrt Constant
Median W_1	*1.15*	*0.99*	-14%	*0.94*	-18%
Median W_2	*1.13*	*0.94*	-17%	*0.89*	-21%
Max MIR W_1	*3.67*	*2.96*	-19%	*2.98*	-19%
Max MIR W_2	*3.65*	*2.84*	-22%	*2.86*	-22%
IQR W_1	*[0.78-1.63]*	*[0.68-1.41]*	-13/-13%	*[0.62-1.37]*	-20/-16%
IQR W_2	*[0.77-1.61]*	*[0.65-1.35]*	-16/-16%	*[0.60-1.30]*	-22/-19%

It should be noted that the percentage differences published in Table XI are relative to the constant MPC values. For MPC weight setting W_1, the median MIR of the oracle MPC lies around 18%, and of the Deep LSTM MPC 14% lower than the median of the constant MPC. For MPC weight setting W_2, the median MIR of the oracle MPC lies around 21%, and of the Deep LSTM MPC around 17% lower than the median of the constant MPC. The maximum estimated MIR, including outliers, corresponding to the worst predicted motion experience, was found to be around 19% lower for the oracle and Deep LSTM MPC relative the constant MPC, for both W_1 and W_2 respectively. When looking at the boxplots it can also be seen that all six conditions show a non-symmetric rating distribution, skewed to the minimum. This means the dispersion of ratings is more narrow towards the minimum, which makes sense considering the MIR is closer to the lower imposed limit than the upper limit.

Table XI, also suggests that the lower 25% limit of the interquartile range (IQR) for the Deep LSTM and oracle MPC, lies roughly 13% and 20% lower compared to the constant MPC, respectively. Another difference can be seen on the upper limit of the IQR, or 75% limit W_1. The upper limit of the Deep LSTM and oracle MPC lies roughly 13% and 16% lower compared to the constant MPC for W_1. Using the W_1 tuning set, it can be concluded that a participant, on average, would rate the perceived motion cueing by the Deep LSTM MPC to be between 13-19$\%$ better, and the oracle MPC to be between 16-20$\%$ better than the motion cueing provided by the constant MPC.

Similarly, using the W_2 tuning set, it can be concluded that a participant, on average, would rate the perceived motion cueing by the Deep LSTM MPC to be between 16-22$\%$ better, and the oracle MPC to be between 19-22$\%$ better than the motion cueing provided by the constant MPC. With the chosen weight settings, this shows that when a simulator is able to use more of its available workspace, the difference between the more accurate references, i.e., Deep LSTM and oracle, and the constant reference becomes marginally more substantial in favour to the Deep LSTM and oracle reference.

C. Motion Cueing Error Analysis

Figure 26 to Figure 28 shows the distribution of percentages for MPC using the three different references for the three testdrives. For this analysis ω_z, i.e., simulated yaw rate, is also included. In ω_z not much difference can be seen between the three MPC reference types. As expected, due to the limited and expensive yaw motion, as well as the heavily scaled down yaw reference, the yaw cueing errors are dominated by missing cues, occurring 35% of the time. As can also be deduced from Figure 28, the remaining 65% of the time, the yaw rate error is smaller than the $3deg/s$ imposed perception threshold.

When looking at the longitudinal specific force one can see that the amount of time scaling errors are present is largest when using the oracle reference. This means that the total amount of time other errors are perceived is lowest when using oracle reference. As expected oracle MPC features the lowest percentage of missing, false and false direction cues.

Comparing the Deep LSTM MPC and constant MPC, one can see that the Deep LSTM MPC predominantly limits the percentage of false direction cues. This is confirmed by looking at the median percentage of occurring false direction cues, i.e., these occur 3.7% of the time for the Deep LSTM MPC. For the constant reference this equals to 7.3% of the time. Oracle provides false direction cues the least amount of time, i.e., 0.9% of the time.
The median time percentage of false cues equals to 4.3%, 3.5%, and 2.1% for the constant, Deep LSTM and oracle reference, respectively. The spread of false cueing errors is largest when using the Deep LSTM reference, showing that motion cueing is more prone to provide false cues when using the Deep LSTM prediction for longitudinal acceleration.

The median percentage of missing cues between the Deep LSTM and constant reference are almost equal, with the error spread being larger in case of the constant reference. This shows that the motion cueing, using the constant reference, features more missing cue errors.

The error percentages in lateral direction show more difference between the types of MPC references. The median

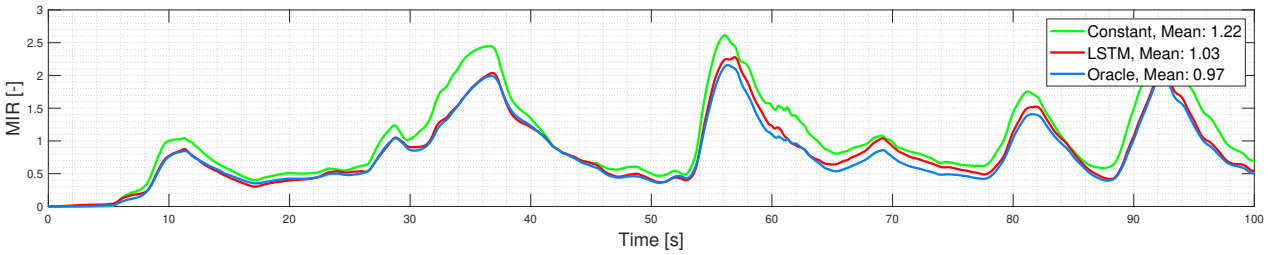


Fig. 23: Continuous motion incongruence ratings for one of the three selected drives. The weight setting equals W_2, i.e., less neutral push is enforced during the optimization.

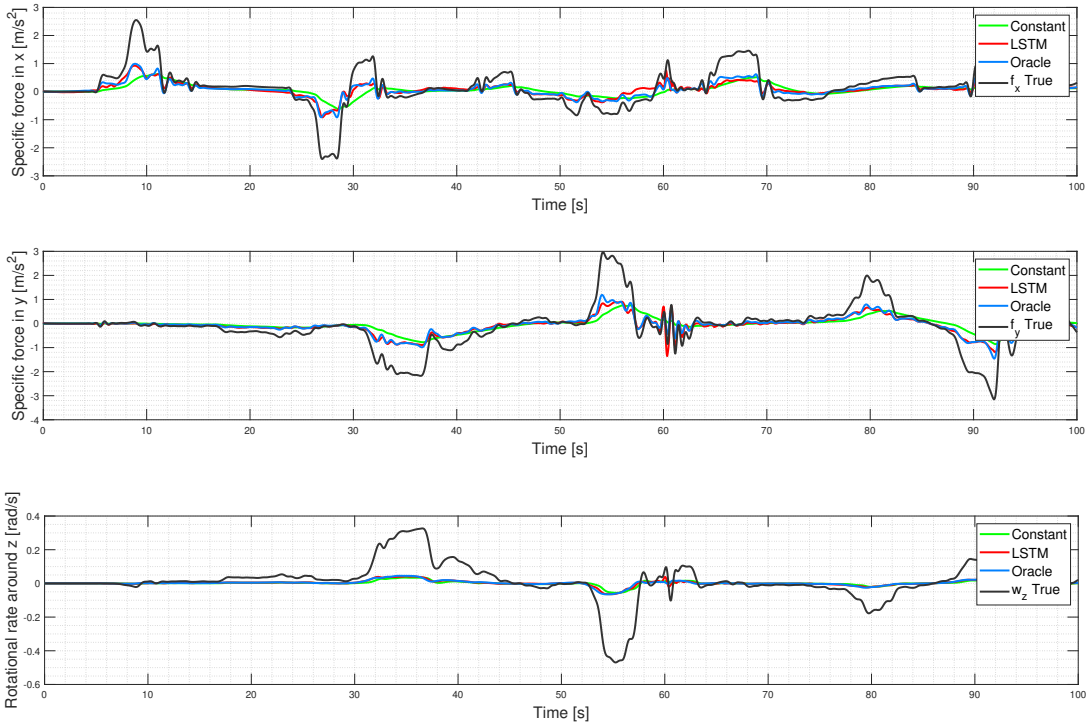


Fig. 24: 100 seconds of MPC MCA outputs, comparing the three types of references for one of the SN drives in the test set. The weight setting equals W_2, i.e., less neutral push is enforced during the optimization.

percentage of missing lateral cues equals 13%, 10%, and 9%, the median percentage of false cues equals 4%, 3.2%, and 1.9% when using the constant, Deep LSTM, and oracle reference, respectively. The largest difference can be found when looking at the false direction cues, where, per median, a false direction cue occurs roughly 7.6%, 1.9%, and 0.9% of the time when using the constant, Deep LSTM, and oracle reference, respectively. For the latter two the spread is narrow. Using the constant lateral acceleration reference provides, in the worst case, 10.7% of the time false direction lateral cues, compared to 2% and 0.8% of the time when using Deep LSTM and oracle MPC, respectively. These findings also lie in line with the prediction quality improvements showed in Table IX, where the lateral acceleration prediction showed an average decrease of 88% in MSE.

To conclude, for both specific force cueing in x and y the oracle reference provides the shortest time of missing, false, and false direction cues, resulting in the longest periods of, less dramatic, scaling cueing errors [14]. Comparing the specific force cueing in x to a constant MPC reference, the Deep LSTM provides less false direction cues, i.e., 3.7% vs 7.3%. However, the difference in missing and false cues does not show a strong advantage to using the Deep LSTM prediction, for which the median amount of false cues is only 0.8% lower compared to a constant reference. Not only that, the spread of presented false cues over the three test drives is larger than compared to a constant reference, with the maximum amount of false cues being around 5% higher. In lateral case, the difference between using a constant and Deep LSTM MPC reference is found to be more significant, with the Deep LSTM providing less missing, false, and false direction cues compared to a constant MPC reference. These two results follow the findings from the Deep LSTM performance analysis, where it was found that a greater reduction in MSE was found in lateral case than in longitudinal case. Yaw rate tracking was found to be equally bad regardless of which reference is used.

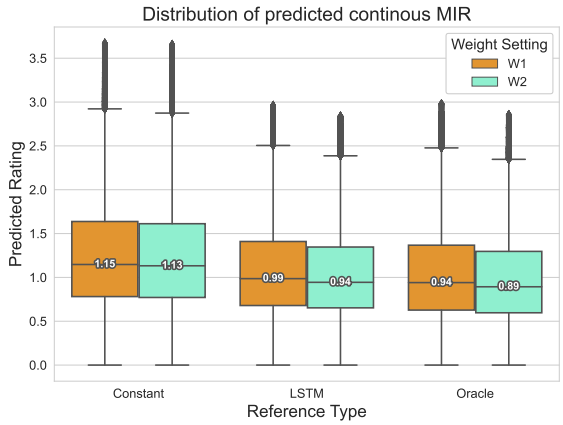


Fig. 25: Boxplots, using simulation results all three test drives, showcasing the distribution and mean of the predicted continuous MIR for the three types of reference per MPC weight setting.

VI. Discussion

A. Neural Network Prediction

Compared to a constant prediction, it was shown that neural networks can provide improved prediction quality over the horizon typically used for MPC in driving simulation. The neural network prediction quality improvement can be ranked (from worst to best) as follows: the three-layer MLP, the single-layer LSTM RNN, the encoder-decoder RNN and finally the Deep LSTM RNN. With the middle two being very comparable in performance. This shows how efficient an encoder-decoder RNN is compared to a normal LSTM RNN, with the former featuring 32% less trainable parameters and only a 6-10% decrease in performance. In future work it is proposed to further optimize the network structures, where it is expected that an efficient multi-layered encoder-decoder neural network will be able to outperform the multi-layered LSTMs for this sequence-to-sequence modelling task.

In [8] a neural network approach is proposed, used to predict a discrete sequence of future vehicle states in an urban driving simulation scenario. In [8] a three-layer dense neural network is utilized, predicting 10 non-linearly spaced samples over $T_p = 4s$. Using a different performance metric, comparing to a constant prediction, an average improvement of around 30% in longitudinal, and 50% in lateral direction and yaw rate was found. This means the three-layer MLP from [8] performs worse by around a 10% margin compared to the reported three-layer MLP in Section V. A reason for this apparent improvement could either result from the fact that in this report 30 output samples are predicted, resulting in a higher temporal resolution, or because future vehicle kinematics in a rural scenario are more predictable compared to an urban scenario. In order to compare results, it is proposed to also train the presented networks for other scenarios such as an urban environment.

Although the overall performance of the proposed neural network models was significantly better than the constant counterpart, it was found that the first samples in the prediction sequence were not anchored to the current state, producing a mismatch between the current and predicted state. Even when using a logarithmically spaced input-output mapping, which implicitly emphasizes a stronger cost on error on the first samples, this behavior persisted. In order to reduce fluctuations on the first samples of the prediction sequence, it was shown that using a half-period cosinebell window omits this fluctuation and anchors the predictions, while keeping the transition smooth. This resulted in the ability to keep the cost on $\Delta\bar{U}$ low, omitting the risk of noisy inputs. In future work, it would be interesting to address this issue when preprocessing the data instead of postprocessing the prediction data. One idea would be to let the neural networks make incremental value predictions, anchored to the previous state, instead of making absolute predictions. This way the first samples in the output mapping are close to the value 0 due to the sampling rate of $100Hz$ and inertia of the vehicle. Introducing such change could possibly provide more consistent short-term predictions.

B. MPC Motion Cueing Quality

Implementing the three different prediction strategies, the MPC cueing results were found to be, from best to worst: the oracle MPC, Deep LSTM MPC, and constant MPC. Using the oracle prediction, it was found that the predicted MIR for both the average as well as the maximum rating is expected to decrease between 16%-22%. However, this is lower than what was found in, e.g., [11], where a decrease of 50% in average ratings was found in a similar rural experiment setup. This could be a result of [11] using a 9-DoF platform which provides a larger motion range, allowing for lower state weighting. In this paper it was found that, marginally lowering the state weighting, a marginal increase in performance when using a more accurate prediction, i.e., Deep LSTM and oracle reference, can be seen. This insinuates that a correlation between allowing for lower neutral push has a positive effect on motion cueing. In future work, it would be interesting to investigate this insinuated correlation between the state weighting and the reference types even more. For this investigation it is proposed to apply the MPC on the full 9-DoF Sapphire Space workspace, i.e., including the XY-Drive as well as the yaw drive, allowing for much different state weightings. In doing so, it is expected that the Deep LSTM and oracle MPC would provide an even larger increase in motion cueing quality compared to the constant MPC, highlighting the potential of improved prediction quality on large motion systems.

Another factor that could explain the more mild decrease in MIR of using an oracle MPC reference, compared to what was found in [11], is explained in the following. [14] assumes a correlation between cueing mismatches in longitudinal and lateral direction only without consideration of yaw rate, an important motion in driving simulation [6]. Adding yaw cueing mismatches to the MIR prediction signal is an area of improvement proposed in [14] as well. Although such addition to the MIR estimation model would not alter the results stated in this paper much, an implicit difference in perceived cueing quality stated in [11], due to the correlation between reference

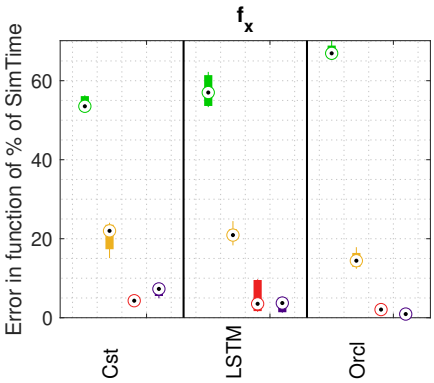


Fig. 26: Cueing error percentages for each reference type for f_x.

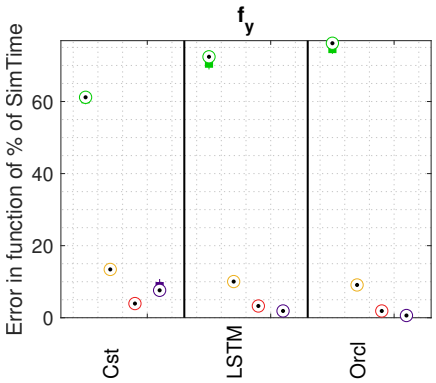


Fig. 27: Cueing error percentages for each reference type for f_y.

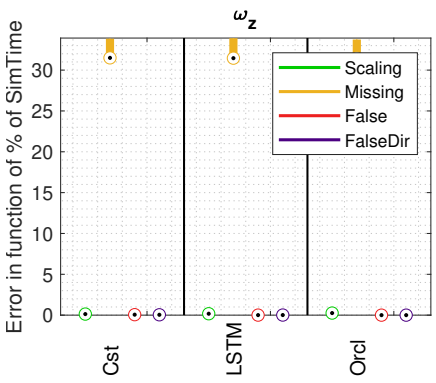


Fig. 28: Cueing error percentages for each reference type for ω_z.

and (yaw) motion capabilities of the system [7, 14], cannot be excluded. If the assumption is made that improved yaw cueing does influence perceived motion cueing performance, a stronger difference in results could be observed when enabling the yaw drive of the Sapphire Space. This would possibly make the relative differences between the Deep LSTM MPC and constant MPC even larger, since the yaw rate prediction error improved by roughly 90% compared to a constant prediction. Next to this, if the independent yaw drive is enabled, the 60% yaw scaling factor can be omitted. It would also allow for more available workspace to be used for specific force cueing in x, y, and z by the hexapod system.

From the estimated MIR analysis, it was found that the average MIR is expected to decrease by 13%-22% when using the Deep LSTM MPC compared to the constant MPC. With respect to oracle MPC, the Deep LSTM MPC would perform around 5% worse. When looking at the maximum MIR peaks, i.e., worst motion cueing experiences, the peaks are expected to drop by 19%-22% when using the Deep LSTM MPC compared to the constant MPC, showing similar performance to the oracle MPC. This shows that using a Deep LSTM reference can be expected to significantly increase the average perceived motion cueing quality compared to the constant reference. However, the results are based on a simulation results, and not on validation data obtained through a DIL experiment. Therefore, it is advised to validate these results by performing a DIL experiment on a dynamic driving simulator. In such an experiment it would be advised to follow the set-up described in [11]. As in [11], 40-50 participants are proposed to participate in the experiment. This way the sample size is large enough to obtain accurate statistical significance results. In this experiment each participant would perceive several prerecorded, rural drives using an open-loop MPC using all three types of references. An open-loop experiment enables the use of the rating method presented in [14], resulting in a high-temporal resolution MIR. To understand the full benefit of using a Deep LSTM reference, it is advised to use the full workspace of a simulator such as the Sapphire Space. Using the full motion capabilities of the Sapphire Space would also allow for validating the correlation between different weight settings and the different reference types. However, to keep the experiment simple, it is advised to use at most two different weight settings.

Another improvement was found when looking at the motion cueing mismatches, where less missing cues and false direction cues were found when using the Deep LSTM and oracle reference. In x direction the results were found to be less conclusive than in y direction, where 80% less false direction cues occurred when using Deep LSTM MPC compared to the constant MPC. Through this analysis, the yaw limit of the system was also highlighted, where only missing cues were present, regardless of the reference type. This leaves the discussion open about which effect a better yaw prediction has on motion perception quality, opening the door for an investigation on using better yaw references on more capable motion systems such as the full Sapphire Space. The effect of these objective cueing errors on motion cueing perception could be analyzed with data gathered from the advised experiment in the preceding paragraph, relating specific error types with peaks in the continuous MIR time trace.

Due to time constraints, it was not possible to compare the performance of the different neural network types when used as reference for the MPC MCA. Because a high-temporal resolution prediction error signal is present, a direct relation with specific maneuvers and the associated subjectively rated MIR can be made. This would allow for a direct correlation between neural network prediction and perceived motion cueing quality, linking an objective prediction rating with a subjective cueing metric.

C. Recommendations

Some other, general recommendations for future work are given in the following. First of all, the resulting neural networks work well in the proposed scenario, but are not tested and validated on a different rural road. This should be investigated further to analyze generalizability of the models. Furthermore, it would be interesting to extend these models to other scenarios, e.g., an urban city scenario and see how the models perform. The proposed models only work in scenarios where the driving route is known a priori, expanding the models with, e.g., a driving direction classification model and a path planning module (resulting in flexible non-causal road

information), would result in more flexible, adaptable and real-time usable prediction generators for use in MPC MCA. One aspect not touched upon in this paper is the real-time capability of using such neural networks in MPC MCA at high control frequency. Although [8] has effectively shown that a three-layer Dense network can be made real-time capable, extending such analysis to more advanced, complex networks is important.

Furthermore, in this paper only open-loop simulation results are evaluated, using only one subpart of a larger kinematic simulator chain. In future work it would be of high value to validate the presented results using previously discussed DIL experiments in a kinematic simulator. It would also be of interest to further develop the evaluation tool proposed in [14] to include the other cued DoFs, as these are not considered, yet might play an important role. Since it was found that reducing the state weighting resulted in a marginally larger gap in MIR between a worse, constant prediction and a better, Deep LSTM or oracle prediction, this effect should be investigated further and eventually validated using DIL experiments as well.
Performing these evaluations and experiments could result in more advanced MIR estimation models, allowing to perform an even better open-loop motion cueing analysis than the ones presented in this paper.

VII. Conclusions

The goal of this paper was to compare vehicle state predictions, performed by different data-driven machine learning frameworks. These predictions could be used as reference tracking signal in a model predictive control based motion cueing algorithm (MPC MCA) framework in closed-loop driving simulation. In this paper four different data-driven neural network structures were trained and analyzed. It was found that, of the four network structures, a three-layer Deep LSTM showed the lowest prediction error. Using the Deep LSTM predictions in an open-loop MPC MCA framework, results showed that the subjective motion cueing quality rating is estimated to decrease significantly compared to the benchmark constant MPC, while nearly matching the estimated cueing quality rating of an MPC with perfect prediction capabilities (oracle). Next to this, results also show a stronger, albeit marginally, increase in motion cueing quality when lowering the state weighting, i.e., neutral push, for both the Deep LSTM and oracle MPC. This finding highlights the potential in using more accurate prediction strategies for larger motion systems where less neutral push is required. Through analyzing the objective motion cueing error mismatches, a significant decrease in false direction lateral cues was found for the Deep LSTM MPC compared to the constant MPC. However, the difference in longitudinal cue errors was less significant.
From the results it can be concluded that, predicting a discrete sequence of vehicle states using different neural network structures improves the prediction quality significantly. Using the Deep LSTM recurrent neural predictions in an MPC-based MCA framework network, is expected to enhance the perceived motion cueing quality, nearly matching the expected quality of an oracle MPC-based MCA.

References

[1] B. Aykent, F. Merienne, G. Christophe, D. Paillot, and A. Kemeny, "Motion sickness evaluation and comparison for a static driving simulator and a dynamic driving simulator," *Proceedings of the Institution of Mechanical Engineers Part D Journal of Automobile Engineering*, 06 2014.

[2] S. F. Schmidt and B. Conrad, "Motion Drive Signals for Piloted Flight Simulators," National Aeronautics and Space Administration, Ames Research Center, Tech. Rep. NASA CR-1601, 1970.

[3] L. D. Reid and M. A. Nahon, "Flight Simulation Motion-Base Drive Algorithms. Part 1: Developing and Testing the Equations," University of Toronto, Institute for Aerospace Studies, Tech. Rep. UTIAS 296, Dec. 1985.

[4] M. Dagdelen, G. Reymond, A. Kemeny, M. Bordier, and N. Maïzi, "MPC Based Motion Cueing Algorithm: Development and Application to the ULTIMATE Driving Simulator," in *Proceedings of the Driving Simulation Conference 2004 Europe*, Paris, France, 2004, pp. 221–233.

[5] F. Ellensohn, F. Oberleitner, M. Schwienbacher, J. Venrooij, and D. Rixen, "Actuator- based optimization motion cueing algorithm," in *2018 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, Auckland, New Zealand, 2018, pp. 1021–1026.

[6] Z. Fang, M. Tsushima, E. Kitahara, N. Machida, D. Wautier, and A. Kemeny, "Motion cueing algorithm for high performance driving simulator using yaw table," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 15 965–15 970, 07 2017.

[7] F. J. Ellensohn, "Urban motion cueing algorithms – trajectory optimization for driving simulators," Ph.D. dissertation, University of Technology München, dec 2019.

[8] A. Lamprecht, T. Emmert, D. Steffen, and K. Graichen, "Learning-based driver prediction for mpc-based motion cueing algorithms," in *Proceedings of the Driving Simulation Conference 2021 Europe VR*, A. Kemeny, J.-R. Chardonnet, and F. Colombet, Eds. Munich, Germany: Driving Simulation Association, 2021, pp. 133–140.

[9] F. M. Drop, M. Olivari, M. Katliar, and H. H. Bülthoff, "Model predictive motion cueing: Online prediction and washout tuning," in *Proceedings of the 17th Driving Simulation Conference 2018 Europe VR*, A. Kemeny, F. Colombet, F. Merienne, and S. Espié, Eds. Antibes, France: Driving Simulation Association, 2018, pp. 71–78.

[10] M. Bruschetta, C. Cenedese, and A. Beghi, "A real-time, mpc-based motion cueing algorithm with look-ahead and driver characterization," *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 61, pp. 38–52, 2019, special TRF issue: Driving simulation. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1369847816306593

[11] F. Ellensohn, D. Hristakiev, M. Schwienbacher, J. Venrooij, and D. Rixen, "Evaluation of an optimization based

motion cueing algorithm suitable for online application," in *Proceedings of the Driving Simulation Conference 2019 Europe VR*, A. Kemeny, F. Colombet, F. Merienne, and S. Espié, Eds. Strasbourg, France: Driving Simulation Association, 2019, pp. 93–100.

[12] A. Mohammadi, H. Asadi, S. Mohamed, K. Nelson, and S. Nahavandi, "Future reference prediction in model predictive control based driving simulators," in *Australian conference on robotics and automation (ACRA2016)*, Queensland, Brisbane, Australia, 12 2016.

[13] D. Cleij, J. Venrooij, P. Pretto, D. M. Pool, M. Mulder, and H. H. Bülthoff, "Continuous subjective rating of perceived motion incongruence during driving simulation," *IEEE Transactions on Human-Machine Systems*, vol. 48, no. 1, pp. 1–13, 09 2017.

[14] M. Kolff, J. Venrooij, M. Schwienbacher, D. M. Pool, and M. Mulder, "Motion cueing quality comparison of driving simulators using oracle motion cueing," in *Proceedings of the 21st DSC 2022 Europe VR*, Strasbourg, France, 10 2022, pp. 111–118.

[15] D. Brezak, T. Bacek, D. Majetic, J. Kasac, and B. Novakovic, "A comparison of feed-forward and recurrent neural networks in time series forecasting," in *2012 IEEE Conference on Computational Intelligence for Financial Engineering and Economics (CIFEr)*, 2012, pp. 1–6.

[16] BMW, "BMW group sets new standards for driving simulation." https://www.press.bmwgroup.com/global/article/detail/T0320021EN/, 11 2020.

[17] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design 2nd Edition*. Nob Hill Publishing, 2017.

[18] M. Behrendt, "Receding horizon optimal mpc," https://commons.wikimedia.org/w/index.php?curid=7963069.

[19] *ISO 8855:2011 Road vehicles — Vehicle dynamics and road-holding ability — Vocabulary*, 2011-12.

[20] R. J. Telban and F. M. Cardullo, "Motion Cueing Algorithm Development: Human-Centered Linear and Nonlinear Approaches," National Aeronautics and Space Administration, Langley Research Center, Hampton, Virginia 23681-2199, Tech. Rep. NASA CR-2005-213747, May 2005.

[21] P. Grieder, F. Borrelli, F. Torrisi, and M. Morari, "Computation of the constrained infinite time linear quadratic regulator," in *Proceedings of the 2003 American Control Conference, 2003.*, vol. 6, 2003, pp. 4711–4716.

[22] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica (Journal of IFAC)*, vol. 38, no. 1, pp. 3–20, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0005109801001741

[23] R. Cagienard, P. Grieder, E. Kerrigan, and M. Morari, "Move blocking strategies in receding horizon control," *Journal of Process Control*, vol. 17, no. 6, pp. 563–570, 07 2007.

[24] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[25] O. Kwon and J. M. Sim, "Effects of data set features on the performances of classification algorithms," *Expert Systems with Applications*, vol. 40, no. 5, pp. 1847–1857, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417412010718

[26] I. Selesnick and C. Burrus, "Generalized digital butterworth filter design," *IEEE Transactions on Signal Processing*, vol. 46, 05 1998.

[27] A. Aksjonov, P. Nedoma, V. Vodovozov, E. Petlenkov, and M. Herrmann, "A novel driver performance model based on machine learning," *15th IFAC Symposium on Control in Transportation Systems CTS 2018*, vol. 51, no. 9, pp. 267–272, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896318307675

[28] A. Jain, H. Koppula, B. Raghavan, S. Soh, and A. Saxena, "Car that knows before you do: Anticipating maneuvers via learning temporal driving models," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Santiago, Chile, 12 2015, pp. 3182–3190.

[29] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas, "Data preprocessing for supervised learning," *International Journal of Computer Science*, vol. 1, no. 12, pp. 111–117, 01 2006.

[30] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *Nuclear Science, IEEE Transactions on*, vol. 44, no. 3, pp. 1464 – 1468, 07 1997.

[31] Z. Tang and P. Fishwick, "Feedforward neural nets as models for time series forecasting," *INFORMS Journal on Computing*, vol. 5, no. 4, pp. 374–385, 11 1993.

[32] R. Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," 11 2019.

[33] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[34] M. Sazli, "A brief review of feed-forward neural networks," *Communications Faculty Of Science University of Ankara*, vol. 50, no. 1, pp. 11–17, 01 2006.

[35] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[36] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Long Beach, California, USA, 2017.

[37] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 12 1997.

[38] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, vol. 2, Montreal, Canada, 2014, p. 3104–3112.

[39] A. Amidi and S. Amidi, "Recurrent neural networks cheatsheet," https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks, 2019.

[40] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and

R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 06 2014.

[41] J. Ploeg, D. Cleij, D. M. Pool, M. Mulder, and B. H.H., "Sensitivity analysis of an mpc-based motion cueing algorithm for a curve driving scenario," in *Proceedings of the 19th Driving Simulation Conference 2020*, Antibes, France, 09 2020.

[42] P. Liashchynskyi and P. Liashchynskyi, "Grid search, random search, genetic algorithm: A big comparison for nas," 12 2019.

[43] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 01 1986.

[44] J. Kukačka, V. Golkov, and D. Cremers, "Regularization for deep learning: A taxonomy," 2018. [Online]. Available: https://openreview.net/forum?id=SkHkeixAW

[45] C. E. Rasmussen and Z. Ghahramani, "Occam's razor," in *Proceedings of the 13th International Conference on Neural Information Processing Systems*, ser. NIPS'00. Cambridge, MA, USA: MIT Press, 2000, p. 276–282.

[46] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations (ICLR)*, Y. Bengio and Y. LeCun, Eds., San Diego, California, USA, 2015.

[47] A. Graves, "Generating sequences with recurrent neural networks," 08 2013.

[48] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 07 2011.

[49] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13-15 May 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[50] J. Smith, *Spectral Audio Signal Processing*, 01 2008.

[51] G. Reymond and A. Kemeny, "Motion Cueing in the Renault Driving Simulator," *Vehicle System Dynamics*, vol. 34, no. 4, pp. 249–259, 2000.

[52] L. D. Reid and M. A. Nahon, "Flight Simulation Motion-Base Drive Algorithms. Part 1: Developing and Testing the Equations," University of Toronto, Institute for Aerospace Studies, Tech. Rep. UTIAS 296, Dec. 1985.

II

Preliminary Thesis

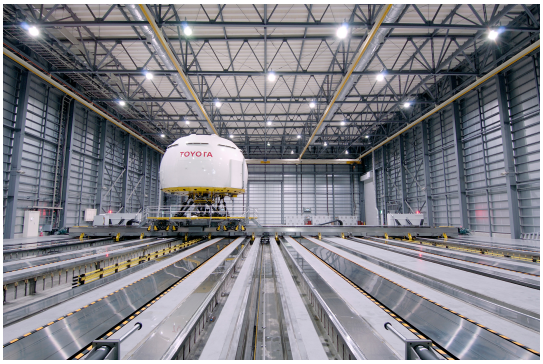
1

Driving Simulation

The following chapter presents important general background information on the topic of the thesis. In Section 1.1, the chapter starts out with a general description of driving simulators, what their purpose is and how different kinematic configurations can be defined. An extension to this section is Section 1.4, where an example of a dynamic simulator is presented, including an example of its motion space definitions. Section 1.2 introduces detailed information on the human sensory system, and more specifically on the vestibular system. How this information translates to the objective of motion cueing and how control engineers use the information to trick the human brain is discussed in Section 1.3. Then, Section 1.5 presents derivations of important relative kinematic equations between a body-fixed and inertial reference frame. To conclude the chapter, Section 1.6 serves as a short summary as well as a discussion on what is next.

1.1. Simulator History

The idea of immersing humans in a dynamic simulated environment dates back to the late 1920's when Edwin Link, by many regarded as the founder of modern flight simulation, developed a dynamic flight training device which enabled instrument flying training. In the late 60' and 70' after the development of the transistor, major advancements in digital computing were made which lead to faster computer speeds, the development of programming languages and ever increasing capacity of storage devices. These advancements would ultimately lead to the micro-electronics revolution, introducing fast computers that were able to solve aircraft equations of motions at at least 60Hz. In the 90's these technological advancements would finally enable smooth operation of hydraulics present in flight simulators since the late 60's [3]. In 1974, it was Volkswagen which developed the first dynamic driving simulator featuring a three degree-of-freedom (DoF) motion system enabling motion in roll, pitch and yaw. However, it was in 1985 when Daimler-Benz in Berlin took this concept to the next level by introducing a 6-DoF hexapod driving simulator, at the time, featuring the world's largest available workspace. Today almost every car manufacturer has a large dynamic driving simulator in its possession [86]. In the last 20 years we have seen that large original equipment manufacturers (OEM) in the automotive industry have pursued the development of costly, large high dynamic driver-in-the-loop (DIL) simulators. In 2007 Toyota finalized the development stages of its own high acceleration capable driving simulator. The simulator features a 9 DoF system consisting of a two-dimensional XY-rails system with a workspace of $35m$ by $20m$, a large hexapod structure mounted atop the rails which features a 330° movable yawtable [97]. In 2011 Daimler reported that, they also, developed a high-dynamic DIL driving simulator capable of reaching velocities up to $10m/s^2$ [23]. In 2017 Renault published that they started with a €25 million project to build a $2000m^2$ driving simulator facility [79]. In more recent news, BMW has opened its 14 simulator rich driving simulation center, containing two large dynamic simulators having a movable mass of 83 and 27 tons, respectively. The center also includes a fleet of 12 smaller dynamic systems [11]. Two examples of large dynamic workspace driver simulators are given in fig. 1.1a and fig. 1.1b.



(a) Toyota's large driver-in-the-loop research simulator [97].

(b) BMW's large driver-in-the-loop research simulator [11].

Figure 1.1: Two examples of large DIL dynamic research driving simulators.

The reasons for OEM's pursuing such vast and expensive projects is manifolded. They entail controllability, predictability and reproducibility without risking expensive (prototype) hardware while guaranteeing safety of the driver. Various use case examples are summarized below.

- **Advanced driver assistance systems (ADAS)**: In the last decade safety requirements have become more stringent for OEM's releasing a new vehicle, and from 2022 all new cars must be equipped with advanced safety systems [66]. Using a DIL driving simulator a driver is able to drive a car on the edges of its working envelope while ensuring a safe conduct of the experiment. In these circumstances many benefits arise in the development and testing of safety critical systems in a reproducible environment [79].
- **Autonomous Vehicle Development**: The development of autonomous vehicle software, such as the transition software required between a state of manual driving to a state of autonomous driving requires validation on a vast amount of testing kilometers in various driving conditions and environments [79]. Unlike companies, without a driving simulator, that need to drive millions of kilometers with an actual car [72], a DIL simulator can help developers perform a large part of autonomous feature validation in a simulated environment before applying it to real vehicles.
- **Car Development**: As mentioned before, in a virtual environment drivers are able to drive cars on the edges of its dynamic envelope without compromising safety. With realistic motion simulators, already early in the development process, valuable feedback can be provided about driving dynamics [11], making the development process more cost efficient.
- **Motion Sickness Studies**: Especially important for autonomous car development, motion simulators can be used to investigate the influence of motion types on e.g. motion sickness, alertness, and sleepiness when drivers are not in direct control of the vehicle, e.g. drivers are assigned a supervisory task [97][11].

In contrast to the main purpose of simulators used in the aerospace industry, i.e. tools used for training such as the device built by Edwin Link, driving simulators are mainly used for research purposes.
Understanding the principle that enables dynamic simulators to provide realistic cues to a human participant is paramount to be able to grasp the challenges engineers face when developing control algorithms. A key point in this discussion lies in human biology, more specifically on how a human is able to perceive motion. This aspect is elaborated upon in the next section.

1.2. Human Perception

The goal of a dynamic driving simulator is to mimic the motion of an actual vehicle as realistic as possible. However, even systems as large as presented earlier, having a motion range of up-to 20-35 meter, have a motion range orders of magnitude smaller than an actual vehicle on the road which can travel several hundreds of meters in the horizontal plane. Because of this discrepancy in available motion space, providing realistic cues is a challenging task. However, weaknesses in human perception

can be exploited to present illusions of motion.
A human is able to perceive motion through various biological mechanisms:

1. The auditory sensory system is an important mechanism in providing cues about ego motion. Three main cues are responsible: sound intensity, binaural cues and the Doppler effect. By internally analysing these cues a human is able to deduce information about velocity and distance of object traveling through the environment [99].

2. The visual sensory system, which comprises of the eye (sensory organ) and parts of the central nervous system, is able to give a human an idea about the motion of its body within an environment. However, the visual sensory system should be complemented with other stimuli from other perception channels to give accurate motion perception [103].

3. The somatosensory/proprioceptive channel is the mechanism that enables a human to perceive touch, pressure, pain, position and movement through muscles, joints, ligaments and skin. This is the driving force for a human to perceive ego motion, i.e. the position and motion of body parts [38].

4. The vestibular system, located in the inner-ear, involves itself with providing stimuli such that a human is able to keep balance. Through the vestibular system a human is able to perceive rotational velocities/rates and translational accelerations by perceiving specific force (accounts for gravity) [38].

The combination of these four channels gives a human the ability to get an accurate perception of reality, his ego motion and motion of other objects within an environment. Since motion cueing mainly concerns itself with the vestibular system, this perceptional channel will be the main consideration for this thesis.
The vestibular organ lies inside the inner-ear and is composed of the semi-circular canals (SCC) and the otolith membrane (OTH), a schematic diagram is presented in fig. 1.2.

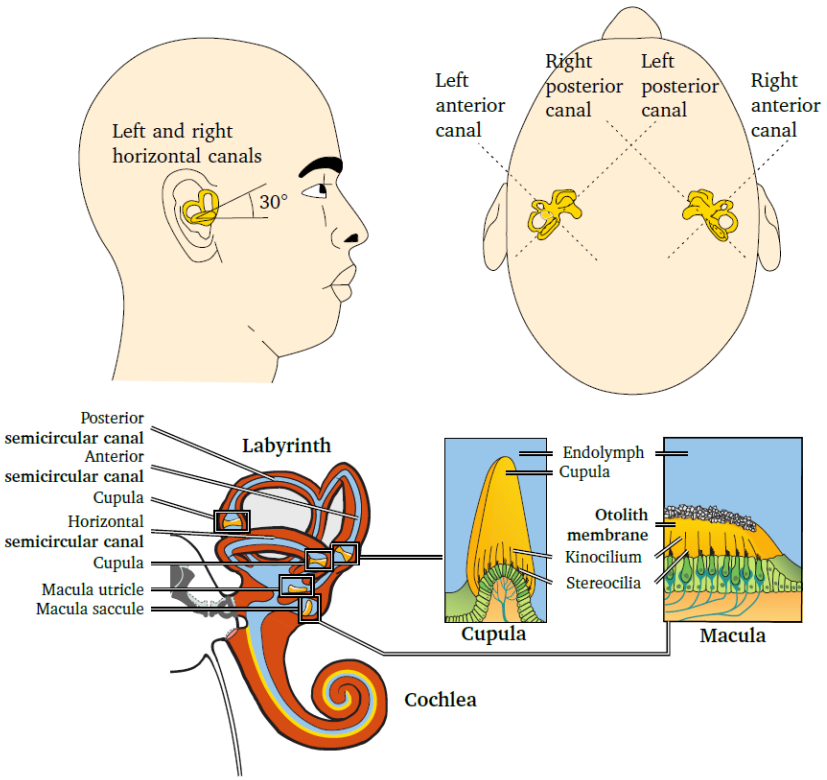


Figure 1.2: Upper figure: Location of the vestibular organs inside the ear, from [28] based on [95]. Lower figure: Schematic of the labyrinth and cochlea, the former consisting of the semi-circular canals, utricle and saccule, from [28] based on [38].

The semi-circular canals contain a small mass of gelatinous material called the cupula. The cupula contains bundles of hair cells, i.e. the sensory receptors. Next to the cupula each of the canals are filled with an endolymph fluid, when the body is rotating the endolymph fluid lags behind and will remain stationary, with respect to the wall of the canal the fluid seems to move in the opposite rotational direction. Because of the rotational differential between the cupula and fluid, a force acts on the cupula by the fluid. This force causes the cupula and hairs to bend, which leads to nerve impulses that a human perceives as rotational rate [38].
The utricle and saccule both contain a macula, which are positioned perpendicular to each other. These provide sensory information about the orientation of as well as dynamic forces that act on the head. The latter of which enables a human to perceive both acceleration as well as deceleration in all three dimensions. Both the utricle and the saccule consist of sensory hair cells which are bundled together, encompassed by a gelatinous material called the otolith membrane. On top of the hair cells sits a dense crystalline layer called the otolith. When a human is accelerating forward a force acts on the otolith. Because the otolith lags behind with respect to the base, i.e. where the hairs attach, the hair bundles bend together with the otolith membrane which leads to nerve impulses: the human perceives an accelerative force [38]. However, as can be seen in fig. 1.3, when the head is tilted forward or backward gravity acts on the otolith in a similar fashion as when one decelerates or accelerates, respectively. What is ultimately perceived is a specific force which is the vectorial sum of the linear accelerations and the gravity vector [21]. Without any additional perceptional cues, a human is not able to distinguish between acceleration/deceleration and tilt [28]. With this property it is possible to cue specific forces other than using translational motion which reduces required motion range.

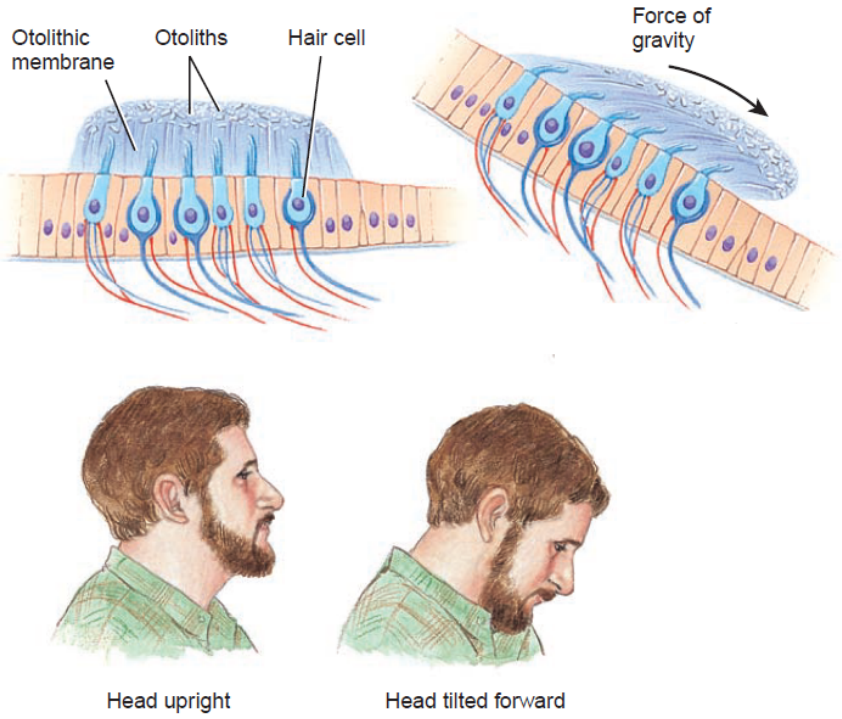


Figure 1.3: Effect of head tilt on the otolith membrane [38].

Many scientists, have performed experiments, considering both subjective sensation as well as physiological mechanical properties of the organs (e.g. hair cell transduction properties) to mathematically model the vestibular organs [106] [95] [68][105]. For this thesis, the reduced-order models proposed by [95] are considered. The mathematical models for the otolith and semi-circular canal are presented in Equation (1.1) and Equation (1.2) respectively.

$$Tf_{oth} = \frac{\hat{a}_i(s)}{a_i(s)} = K_{oth} \cdot \frac{1 + \tau_{a_1} \cdot s}{(1 + \tau_{a_2} \cdot s)(1 + \tau_{a_3} \cdot s)} \tag{1.1}$$

$$Tf_{scc} = \frac{\hat{\omega}_i(s)}{\omega_i(s)} = K_{scc} \cdot \frac{\tau_{\omega_1} \cdot s}{(1 + \tau_{\omega_2} \cdot s)(1 + \tau_{\omega_3} \cdot s)} \tag{1.2}$$

Where:

1. $\hat{a}_i(s)$ and $\hat{\omega}_i(s)$ depict the perceived translational acceleration and rotational rate, respectively.
2. $a_i(s)$ and $\omega_i(s)$ depict the vehicle translational acceleration and rotational rate as input to the model.
3. K_{oth} and K_{scc} are the organ specific gains.
4. τ depicts the filter specific time shifts necessary to model each organ.

The values of the parameters are shown in table 1.1.

Table 1.1: Vestibular system transfer function parameters.

Parameter	Value
$[K_{oth}, K_{scc}]$	*[0.4, 5.73]*
$[\tau_{a_1}, \tau_{a_2}, \tau_{a_3}]$	*[10, 5, 0.016]*
$[\tau_{\omega_1}, \tau_{\omega_2}, \tau_{\omega_3}]$	*[80, 80, 5.73]*

In fig. 1.4 the Bode plots of both the otolith as well as the semi-circular canal are shown. Analyzing the Bode plot the following points can be deduced.
For frequencies above $0.1Hz$ the magnitude response decreases by $20dB/decade$ which depicts integrator dynamics (1/s), effectively meaning for frequencies larger than $0.1Hz$ the SCC acts as an integrated accelerometer or velocity transducer which lags 90 degrees behind, i.e. phase shift of around minus 90 degrees. For low frequencies, between $[10^{-2.5}Hz, 10^{-1}Hz]$, the SCC serves as an accelerometer, depending on the magnitude, featuring leading or lagging behavior [95].
The Bode plot of the otolith organ shows that in the region $[10^{-1}Hz, 10Hz]$ the gain remains constant with little leading behavior for lower frequencies and little lagging behavior for larger frequencies in the region. This means that in this region the otolith organ acts as an accelerometer, i.e. specific force transducer [95].

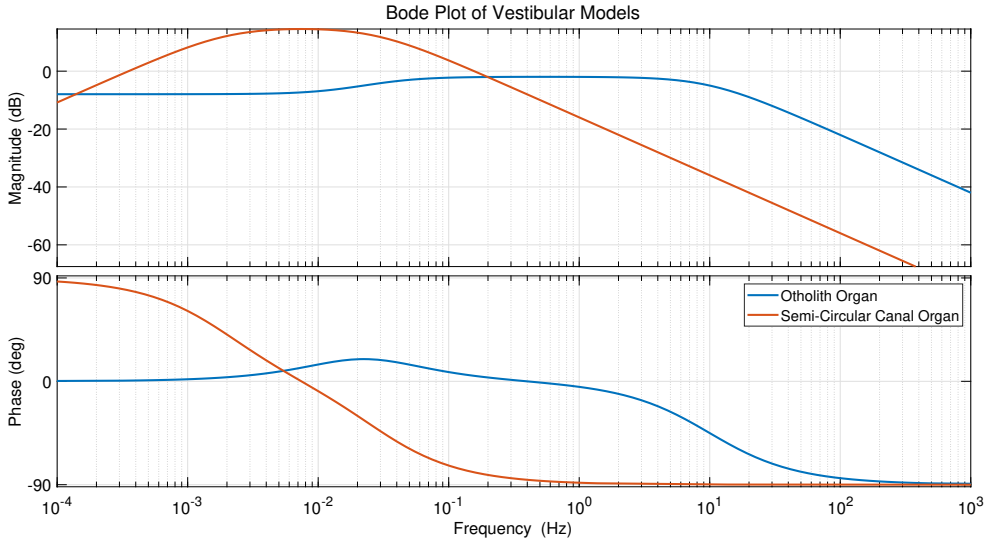


Figure 1.4: Frequency response of the vestibular models presented by [95].

1.3. Tilt Coordination

In the previous section the different perceptional channels of a human were highlighted and the vestibular system and its models were investigated. With this information the main objective of a motion cueing algorithm (MCA) can be explained. The goal of a driving simulator is to minimize the error between:

1. The actual, measurable specific forces acting on the driver in a (virtual) vehicle and simulated specific forces.
2. The actual, measurable rotational rates acting on the driver in a (virtual) vehicle and simulated rotational rates.

As briefly discussed in the previous section, this is not easy. In order to make the most out of the available workspace of a simulator motion cueing algorithms utilize tilt-coordination to induce specific forces, effectively increasing the reproducible translational acceleration range. An example that shows the foundation of why this is required is given in the following.
Because of the large differences in motion range between a simulator and a real-world vehicle, cueing translational accelerations is challenging. Low frequency, long lasting accelerations are impossible to cue by solely performing translational motions. An example: if a driver wants to accelerate to $100km/h$, i.e. $27.8m/s$, from standstill with a small and constant acceleration of $2m/s^2$, this would take $27.8/2 = 13.9sec$. The distance spanned during this accelerative manoeuvre equals $193m$ ($x = 1/2 \cdot a \cdot t^2$). Even for a large dynamic simulator, it would be impossible to cue the full accelerative manoeuvre by translational movement only. However, due to the fact that one is able to induce specific forces by tilting, given that no other perceptional cues are given, it is possible to simulate such long lasting accelerations by simply tilting the simulator with a certain angle without using the translational channel at all, see fig. 1.5. In this specific example, the angle required to simulate a constant acceleration of $2m/s^2$ equals $11.8°$, the mathematical background given in Equation (1.3).

$$\begin{aligned} f_{x,tilt} &= g * sin(\theta) \\ \Leftrightarrow \theta &= asin\left(\frac{f_{x,tilt}}{g}\right) \\ \Leftrightarrow \theta &= asin\left(\frac{-2}{-9.81}\right) = 11.8° \end{aligned} \tag{1.3}$$

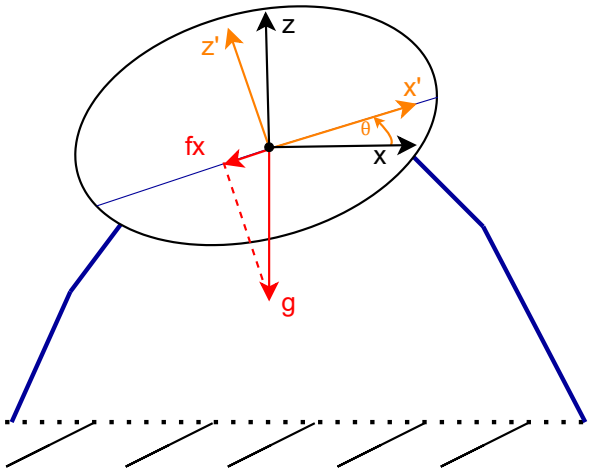


Figure 1.5: Schematic showing how tilt-coordination induces a longitudinal specific force.

This is what is called tilt-coordination, a technique to simulate long lasting horizontal accelerations by performing a tilt motion. In the example given above only one direction was considered. In driving manoeuvres rotations around both the roll- and pitch-axis occur, i.e. ϕ and θ respectively. Therefore

the general formulas include both rotation angles. From geometry the formulas can be represented as shown in Equation (1.4) [7], with the specific force $f = -a_i$, i.e. specific force is the inverse of the to be induced acceleration.

$$\begin{aligned} f_{x,tilt} &= -g \cdot sin(\theta) \\ f_{y,tilt} &= g \cdot cos(\theta) \cdot sin(\phi) \\ f_{z,tilt} &= g \cdot cos(\theta) \cdot cos(\phi) \end{aligned} \tag{1.4}$$

1.4. BMW's High-Fidelity Simulator

In this thesis, the specifications of the high-fidelity simulator (HF) from BMW will be used in exploratory investigations, it is the largest dynamic simulator currently in BMW's fleet.
The HF is a large 9-DoF dynamic simulator shown in fig. 1.6. It consists of a XY-rail system, on top of which a hexapod is mounted. The legs of the hexapod are connected to a lightweight dome structure in which a mock-up is connected to a 360° movable yawtable. 15 projectors project the virtual environment on the wall of the dome to give full 360° immersion.

Figure 1.6: BMW's high-fidelity research simulator [11].

The dimensional specifications of the simulator's workspace are provided in table 1.2.

Table 1.2: HF simulator dimensional specifications.

Hexapod					
x_H	$\pm 1.2m$	$\dot{x}_H$	$\pm 1m/s$	$\ddot{x}_H$	$\pm 10m/s^2$
y_H	$\pm 1.2m$	$\dot{y}_H$	$\pm 1m/s$	$\ddot{y}_H$	$\pm 10m/s^2$
z_H	$\pm 0.8m$	$\dot{z}_H$	$\pm 1m/s$	$\ddot{z}_H$	$\pm 8m/s^2$
ϕ_H	$\pm 26°$	$\dot{\phi}_H$	$\pm 20°/s$	$\ddot{\phi}_H$	$\pm 150°/s^2$
θ_H	$\pm 25°$	$\dot{\theta}_H$	$\pm 20°/s$	$\ddot{\theta}_H$	$\pm 150°/s^2$
ψ_H	$\pm 36°$	$\dot{\psi}_H$	$\pm 20°/s$	$\ddot{\psi}_H$	$\pm 150°/s^2$
XY-Table					
x_{XY}	$\pm 9.5m$	$\dot{x}_{XY}$	$\pm 6m/s$	$\ddot{x}_{XY}$	$\pm 6.5m/s^2$
y_{XY}	$\pm 7.9m$	$\dot{y}_{XY}$	$\pm 6m/s$	$\ddot{y}_{XY}$	$\pm 6.5m/s^2$
Yawtable					
ψ_{yaw}	$\pm 180°$	$\dot{\psi}_{yaw}$	$\pm 40°/s$	$\ddot{\psi}_{yaw}$	$\pm 150°/s^2$

1.5. Simulator Kinematics

A motion cueing algorithm uses kinematic information from the virtual vehicle, i.e. accelerations and rotational rates to control the simulator. This means virtual information is used to control a physical object in the world. How this influences control design will be explained in the following section where the relevant reference frames as well as the required transformations and mathematical derivations will be described.

1.5.1. Relevant Reference Frames

Describing the virtual vehicle-simulator relation, two reference frames are important. The body fixed, denoted by 'b', and inertial reference frame, denoted by 'I'. The body fixed reference frame is located at the driver's head and follows the ISO 8855 convention for vehicle reference frames [1]. A schematic of the body-fixed reference frame and its definitions is given in fig. 1.7. The x-axis is defined to go through the nose of the vehicle, the z-axis is defined upwards perpendicular to the floor of the vehicle, from the right-hand rule the y-axis is defined to go through the left side of the vehicle. Rotation around the y-axis is defined to as pitch (θ), roll (ϕ) is defined as the rotation around the x-axis and yaw (ψ) is defined as the rotation around the z-axis. All rotational directions follow the right-hand rule.

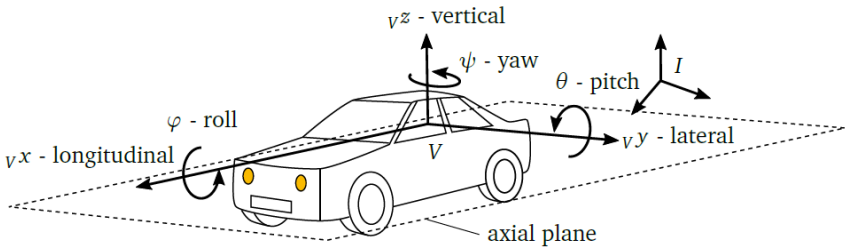


Figure 1.7: Body-fixed reference frame following conventions from ISO 8855 [1], picture from [28].

The inertial reference frame is also a right-hand orthogonal axis-system and is fixed in the real environment with respect to the defined inertial orientation of the HF simulator, which is independent of simulator movement. A schematic representation is given in fig. 1.8.

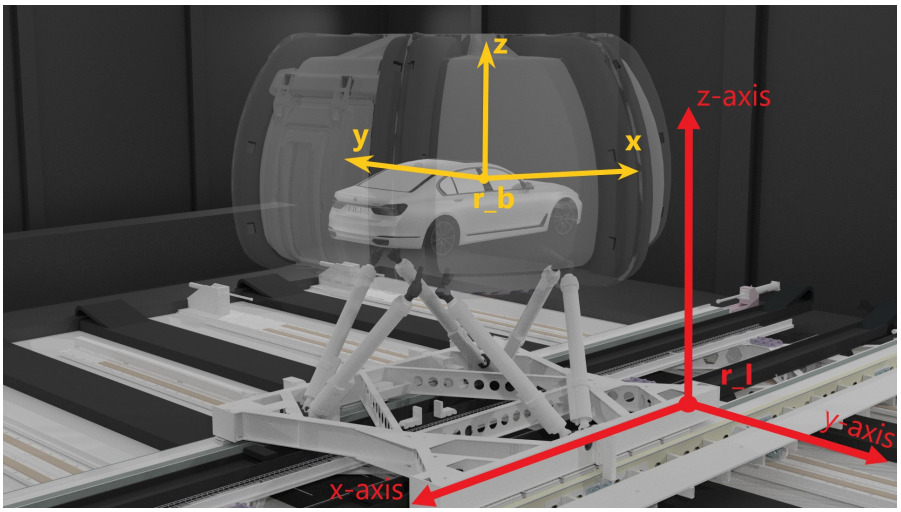


Figure 1.8: Body-fixed reference frame following conventions from ISO 8855 [1] and inertial reference frame, picture retrieved from [11].

1.5.2. Reference Frame Transformations

In order to describe the orientation of a rigid body defined by its angles (ϕ, θ, ψ), i.e. defined in body-fixed reference frame, in the inertial coordinate system one can use the cardan angles description [98]. The angles are defined as a sequence of three elementary rotations: x-y-z. When doing so one has to take

care of the angle direction definitions. A schematic is given in fig. 1.9. When doing the transformation two subsidiary axis systems are required, in the figure denoted by a prime and double prime for the first and second subsidiary system respectively. The sequence can be explained as follows:

1. Rotate the body around the b_x axis with $-\phi$ (right-hand rule applies) to the first subsidiary axis system b', for which $b_x = b'_x$.
2. Rotate the body around the b'_y axis with $-\theta$ (right-hand rule applies) to the second subsidiary axis system b'', for which $b'_y = b''_y$.
3. Rotate the body around the b''_z axis with $-\psi$ (right-hand rule applies) to the inertial reference system I, for which $b_x = b'_x$.

Mathematically, this can be written as:

$$\mathrm{T}_{I,b} = \mathrm{T}_z(-\psi) \cdot \mathrm{T}_y(-\theta) \cdot \mathrm{T}_x(-\phi) \tag{1.5}$$

With the transformation matrix around x, y and z defined in Equation (1.6).

$$\begin{aligned} \mathrm{T}_x(\phi) &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -sin(\phi) & cos(\phi) \end{pmatrix}, \quad \mathrm{T}_y(\theta) = \begin{pmatrix} cos(\theta) & 0 & -sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & cos(\theta) \end{pmatrix}, \\ \mathrm{T}_z(\psi) &= \begin{pmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{1.6}$$

A rotational transformation from inertial to the body reference frame can be described by the inverted sequence:

$$\mathrm{T}_{b,I} = \mathrm{T}_x(\phi) \cdot \mathrm{T}_y(\theta) \cdot \mathrm{T}_z(\psi) \tag{1.7}$$

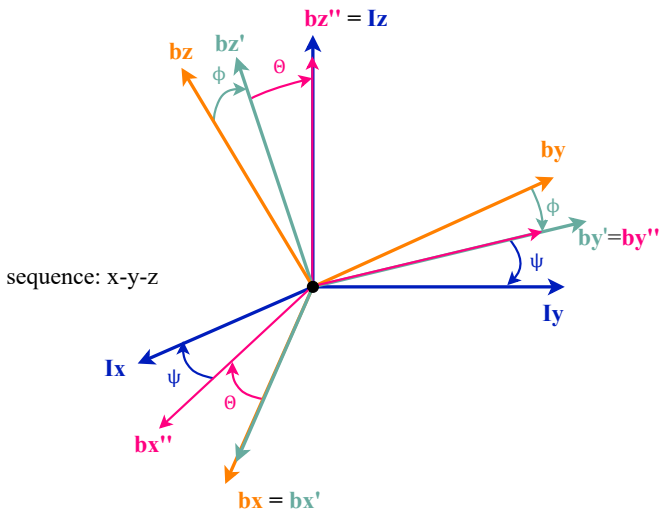


Figure 1.9: Cardan rotation sequence from body to inertial reference frame [98].

Naming conventions and relative kinematic definitions are given in table 1.3 [64].

Table 1.3: Naming conventions and definitions of relative kinematics.

Translational	
${}_I\vec{r}_{P,Q}$	Vector $\vec{r}$ defined in reference frame 'I' starting at point P and ending at point Q.
Rotational	
$\mathrm{T}_{I,b}$	Rotation matrix defined to transform a vector from reference frame 'b' to 'I'.
${}_I\vec{\omega}_{I,b}$	Relative angular velocity between reference frame 'I' and 'b' as seen from reference frame 'I'.
Relevant Points	
Control reference point (CRP)	Point located at the driver's head in the body reference frame of both the virtual vehicle as well as the physical simulator.
Motion reference point (MRP)	Point located at the center of the yawtable, for the high fidelity simulator this is located on the floor. Motion commands are computed wrt MRP defined in inertial reference frame.

Given the objective of motion cueing, presented in Section 1.3, one is able to understand why these reference frames and vector transformation descriptions are important. The error between perceived specific forces and rotational rates that motion cueing tries to minimize is the error in the **body reference frame**. However, motion commands to the simulator, often consisting of positional, velocity and acceleration commands in all DoF's, need to be given in the **inertial reference frame**. The following sequence of actions can be deduced:

1. Measure vehicle kinematic information defined in its own body reference frame.
2. Use these measurements as inputs to the motion cueing algorithm.
3. The MCA transforms these inputs to desirable simulator motion commands in body reference frame.
4. Transform simulator motion commands from body reference frame to defined inertial reference frame to obtain actual motion commands.

In the following example the HF simulator is considered. Derivations to define the position, velocity and acceleration of a point, defined in an inertial reference frame, in the body-fixed reference frame will be presented in the following. Transforming a positional vector from a body to inertial reference frame follows the same analogy. For illustration purposes, relevant vectors and reference frames are defined in fig. 1.10.

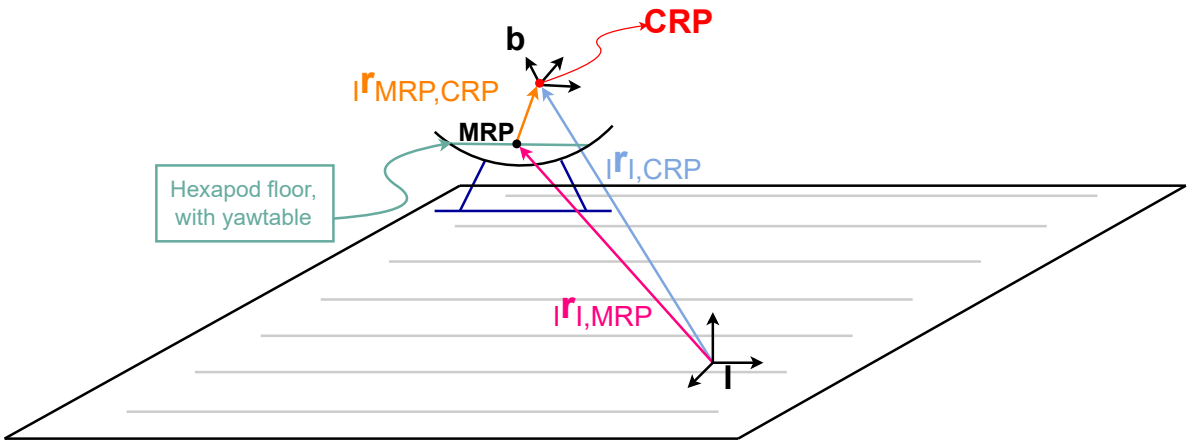


Figure 1.10: Vector definition of a point, i.e. CRP, relative to the inertial frame and body reference frame for a system with three different kinematic elements.

The position of the CRP in inertial space is represented by the vector ${}_I\vec{r}_{I,CRP}$, which can be written as a superposition of two other vectors:

$$\begin{aligned} {}_I\vec{r}_{I,CRP} &= {}_I\vec{r}_{I,MRP} + {}_I\vec{r}_{MRP,CRP} \\ \Leftrightarrow {}_I\vec{r}_{I,CRP} &= {}_I\vec{r}_{I,MRP} + \mathrm{T}_{I,Y} \cdot {}_Y\vec{r}_{MRP,CRP} \end{aligned} \tag{1.8}$$

Where 'I' demeans the inertial reference frame, and 'Y' depicts the reference frame that is connected to the yawtable on top of the hexapod. When the simulator is in neutral position, both these reference frames align.
One needs to note that one reference frame is omitted in this schematic which is 'H', i.e. the reference frame that is connected to the hexapod. 'H' is represented by 'Y' through a rotational transformation around their common z-axis with ψ_{yaw}:

$$\mathrm{T}_{I,Y} \cdot {}_Y\vec{r}_{MRP,CRP} = \mathrm{T}_{I,H}\mathrm{T}_{H,Y} \cdot {}_Y\vec{r}_{MRP,CRP} \tag{1.9}$$

Which effectively means that:

$${}_I\vec{r}_{I,CRP} = {}_I\vec{r}_{I,MRP} + \mathrm{T}_{I,H}\mathrm{T}_{H,Y} \cdot {}_Y\vec{r}_{MRP,CRP} \tag{1.10}$$

The change in position of CRP over time can be written as, with ${}_Y\dot{\vec{r}}_{MRP,CRP} = 0$:

$$\begin{aligned} {}_I\vec{v}_{I,CRP} &= \frac{d_I\vec{r}_{I,CRP}}{dt} \\ &= {}_I\dot{\vec{r}}_{I,MRP} + \dot{\mathrm{T}}_{I,H}\mathrm{T}_{H,Y} \cdot {}_Y\vec{r}_{MRP,CRP} + \mathrm{T}_{I,H}\dot{\mathrm{T}}_{H,Y} \cdot {}_Y\vec{r}_{MRP,CRP} \end{aligned} \tag{1.11}$$

Taking the derivative of the velocity vector and adding the gravitational force vector results in:

$$\begin{aligned} {}_I\vec{a}_{I,CRP} &= \frac{d_I\vec{v}_{I,CRP}}{dt} \\ &= {}_I\ddot{\vec{r}}_{I,MRP} + \ddot{\mathrm{T}}_{I,H}\mathrm{T}_{H,Y} \cdot {}_Y\vec{r}_{MRP,CRP} + 2\dot{\mathrm{T}}_{I,H}\dot{\mathrm{T}}_{H,Y} \cdot {}_Y\vec{r}_{MRP,CRP} + \mathrm{T}_{I,H}\ddot{\mathrm{T}}_{H,Y} \cdot {}_Y\vec{r}_{MRP,CRP} + {}_I\vec{g} \end{aligned} \tag{1.12}$$

Defining the acceleration in the body reference frame gives:

$$
\begin{aligned}
{}_b\vec{a}_{I,CRP} &= \mathrm{T}_{b,I}\frac{d_I\vec{v}_{I,CRP}}{dt} \\
&= \mathrm{T}_{b,I}\cdot {}_I\ddot{\vec{r}}_{I,MRP} + \mathrm{T}_{b,I}\ddot{\mathrm{T}}_{I,H}\mathrm{T}_{H,Y}\cdot {}_Y\vec{r}_{MRP,CRP} + 2\mathrm{T}_{b,I}\dot{\mathrm{T}}_{I,H}\dot{\mathrm{T}}_{H,Y}\cdot {}_Y\vec{r}_{MRP,CRP} \\
&\quad + \mathrm{T}_{b,I}\mathrm{T}_{I,H}\ddot{\mathrm{T}}_{H,Y}\cdot {}_Y\vec{r}_{MRP,CRP} + \mathrm{T}_{b,I}\cdot {}_I\vec{g}
\end{aligned} \tag{1.13}
$$

Given the cardan angles, one is able to derive the Jacobian matrix, which is a transformation matrix describing rotational rate in function of a change in cardan angles over time. Using the fact that the rotational rate vector equals the sum of each individual relative angular velocity vectors [64], where $\vec{e}_{ref,sub}$ is the unit vector defined in reference frame 'ref' for axis 'sub'. Using fig. 1.8:

$$
\vec{\omega} = \dot{\psi}\cdot\vec{e}_{I,z} + \dot{\theta}\cdot\vec{e}_{\prime,y} + \dot{\phi}\cdot\vec{e}_{\prime\prime,x} \tag{1.14}
$$

Writing in body-fixed reference frame gives:

$$
{}_b\vec{\omega}_{I,b} = \dot{\psi}\cdot {}_b\vec{e}_{I,z} + \dot{\theta}\cdot {}_b\vec{e}_{\prime,y} + \dot{\phi}\cdot {}_b\vec{e}_{\prime\prime,x} \tag{1.15}
$$

$$
\Leftrightarrow {}_b\vec{\omega}_{I,b} = \dot{\psi}\mathrm{T}_{b,I}\cdot {}_b\vec{e}_{I,z} + \dot{\theta}\mathrm{T}_{b,\prime}\cdot {}_b\vec{e}_{\prime,y} + \dot{\phi}\mathrm{T}_{b,\prime\prime}\cdot {}_b\vec{e}_{\prime\prime,x} \tag{1.16}
$$

$$
\Leftrightarrow {}_b\vec{\omega}_{I,b} = \dot{\psi}\mathrm{T}_x\mathrm{T}_y\mathrm{T}_z\cdot {}_b\vec{e}_{I,z} + \dot{\theta}\mathrm{T}_x\mathrm{T}_y\cdot {}_b\vec{e}_{\prime,y} + \dot{\phi}\mathrm{T}_x\cdot {}_b\vec{e}_{\prime\prime,x} \tag{1.17}
$$

$$
\Leftrightarrow {}_b\vec{\omega}_{I,b} = \mathrm{T}_x\mathrm{T}_y\mathrm{T}_z\cdot\begin{pmatrix}0\\0\\1\end{pmatrix}\cdot\dot{\psi} + \mathrm{T}_x\mathrm{T}_y\cdot\begin{pmatrix}0\\1\\0\end{pmatrix}\cdot\dot{\theta} + \mathrm{T}_x\cdot\begin{pmatrix}1\\0\\0\end{pmatrix}\cdot\dot{\phi} \tag{1.18}
$$

$$
\Leftrightarrow {}_b\vec{\omega}_{I,b} = \begin{pmatrix}1 & 0 & -sin(\theta)\\ 0 & cos(\phi) & cos(\theta)sin(\phi)\\ 0 & -sin(\phi) & cos(\theta)cos(\phi)\end{pmatrix}\cdot\begin{pmatrix}\dot{\phi}\\\dot{\theta}\\\dot{\psi}\end{pmatrix} \tag{1.19}
$$

$$
\Leftrightarrow {}_b\vec{\omega}_{I,b} = J_{b,I}\cdot\begin{pmatrix}\dot{\phi}\\\dot{\theta}\\\dot{\psi}\end{pmatrix} \tag{1.20}
$$

With $J_{b,I}$ defined as the Jacobian matrix that correlates the rotational velocity with the change in cardan angles in the body-fixed reference frame. Similarly, the Jacobian for the inertial reference frame can be found to be:

$$
J_{I,b} = \begin{pmatrix}cos(\psi)cos(\theta) & -sin(\psi) & 0\\ sin(\psi)cos(\theta) & cos(\psi) & 0\\ -sin(\theta) & 0 & 1\end{pmatrix} \tag{1.21}
$$

1.6. Conclusion and Discussion

This chapter started off with a discussion on the background of automotive dynamic driving simulation. Automotive driving simulation is a direct spin-off from technology used in the aerospace industry. Contrary to dynamic flight simulators, that were initially developed to serve training purposes, simulators in the automotive industry mainly serve research purposes. Today, almost every large car manufacturer utilizes a dynamic driving simulator. Since the mid 80's, when development of large dynamic driving simulators started, technological advancements have seen continuous growth. It was argued that to understand why these advancements are still being pursued, a better understanding of human biology, more specifically human motion perception, was required.

A human can perceive motion through different channels, and organs. Mainly four different channels are utilized. They can be summarized into the auditory, visual, somatosensory and vestibular sensory system. When combining these four channels a human is able to get an accurate estimate of ego-motion, as well as motion of other objects in the environment. Motion cueing, i.e. the act of dynamically controlling the motion of a simulator, mainly concerns itself with the latter vestibular system. The vestibular system consists of three important organs, the semi-circular canals, utricle and saccule organ. The semi-circular canals provide information about ego-rotational rate, whereas the latter two

organs provide information about ego-acceleration in all directions by sensing specific force. Although some large dynamic driving simulators feature a translational workspace that can span several meters, the translational workspace of a car is orders of magnitude larger. To make sure sufficient cues are presented to the driver, a trick can be utilized. This trick is called tilt-coordination. Without any perceptional cues, a human is not able to distinguish between acceleration or deceleration and tilting motion. This means that by tilting a simulator, a sense of acceleration or deceleration can be simulated. This effectively reduces the required motion workspace of a simulator.
Finally, the chapter concerns itself with an investigation into simulator kinematics. When controlling a simulator, the motion commands are given in the inertial frame of reference, however, the motion that a driver perceives is described in the body frame of reference. It was investigated how translational as well as rotational motion in one of the frames of reference can be described in the other frame of reference. Several expressions for transformation as well as Jacobian matrices were derived. These expressions were found to be required when deriving mathematical expressions that can transform motion from one frame to another.
To conclude, this chapter served to investigate the mathematical fundamentals required to support an in-depth investigation in motion cueing control techniques. The first control technique that will be investigated is called the Classical Washout Algorithm and is presented in Chapter 2. Chapter 2 will utilize mathematical expressions derived in the preceding chapter.

2

Filter-Based Motion Cueing Algorithm

In the following chapter the filter-based classical washout algorithm (CWA) will be presented. In Section 2.1 the block diagram of the CWA will be discussed, with each individual block carefully explained. In Section 2.1 the correlation between a change in filter parameters and system response will also be investigated. Based on these findings, the performance of the CWA is analyzed in Section 2.2 using real vehicle data as input to the algorithm. This chapter will be concluded in Section 2.3 where the advantages and shortcomings of the presented CWA will be discussed.

2.1. Classical Washout Algorithm

As explained in Chapter 1 the goal of a driving simulator is to mimic the motion of a vehicle. This proves to be quite a difficult task because the dynamic workspace of an actual vehicle is orders of magnitude larger than that of a motion simulator. In order to make the most out of the available workspace, motion cueing control algorithms (MCA) are required. These algorithms transform the virtually driven vehicle output to a control input to the simulator. The goal of the MCA is to mimic the cues of the vehicle to the best of its capabilities.

Although motion perception is very subjective, 50 years ago Schmidt and Conrad [85] presented an objective framework that allows one to investigate motion cueing algorithms by looking at the error between the specific forces and rotational rates a pilot would perceive in the actual vehicle, and the ones perceived in the simulator. A smaller error is assumed to provide better motion cueing. They also proposed a filter-based washout algorithm that is able to simulate aircraft motions, named the ”classical washout algorithm”. The block diagram of the algorithm is presented in fig. 2.1.

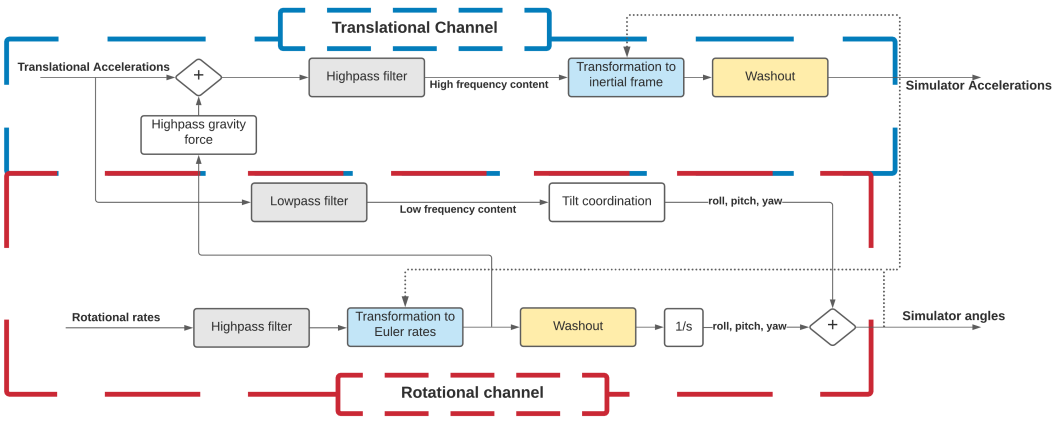


Figure 2.1: Control structure of the classical washout algorithm as defined by Conrad et al. [20].

On the left of the block diagram two sets of inputs are present (either synthetic signals or in-vehicle acceleration/rotational rate measurements). The first set comprises of the specific forces to be sim-

ulated and are expressed as the vehicle reference translational accelerations in longitudinal, lateral and vertical direction. The second set of inputs comprises of the three dimensional angular accelerations: the vehicle roll, pitch and yaw rate. Outputs are the translational acceleration and rotational rate setpoints in all DoF to the motion system. To get a better understanding of how the input signals are manipulated, the main blocks are subdivided into five categories, each of which will be individually explained:

1. **Scaling**: A scaling vector will be introduced that applies before filtering, this keeps functionality local. Variable filter gains are omitted and set to a value of '1'.
2. **Splitting the frequency**: Low- and high-pass filter that splits the incoming signal into low and high frequency signals.
3. **Transformation body-to-inertial reference frame**: Transformation from vehicle body reference frame to the inertial reference frame and transformation from vehicle rates to Euler rates.
4. **Washout**: Filter that makes sure the simulator returns to its neutral position after performing a manoeuvre.
5. **Transformation inertial-to-body reference frame**: Simulator movements are defined in the inertial reference frame, however the pilot feels the movement in his/her own body reference frame. Therefore the motions have to be transformed back to a body-fixed reference frame in order to compare results.

2.1.1. Scaling

The vehicle acceleration envelope is broad, ranging from around 1g ($\approx 10m/s^2$) in pure forward (acceleration) to around -2g in pure backward longitudinal direction (braking). Depending on the driver behind the wheel, only parts of the vehicle's full capabilities are utilized [10]. Because of the unknown nature of the driver's acceleration envelope and in order to restrict the simulator displacement [77] a scaling factor on the inputs to the CWA is used. This means the inputs need to be scaled based on the worst case scenario.
Next to this, given the framework by Schmidt and Conrad [85], one-to-one cueing of the vehicle's acceleration and rotational rate would be desirable. However, it was found that in flight and driving simulation one-to-one cueing does not necessarily provide the best perception of self-motion [44]. To this extent an appropriate scaling factor is beneficial.
The effect of gain selection can be defined as follows. Higher gains will provide higher amplitude system setpoints, therefore they need to be carefully tuned in order to not reach the limits of simulator workspace. Next to larger system excitations, higher gains will also enlarge the effect of false cues but will also make the driving task more challenging as found by [44]. However, selecting a gain that is too low will result in numbed down simulator excitation, reducing the motion cueing quality as well as the driver performance.

2.1.2. Splitting the frequency

As explained in Section 1.2 longitudinal and lateral motions can be replicated by translational accelerations and tilt motions. Due to simulator workspace limitations, low frequency accelerations need to be replicated by tilt-coordination whereas high frequency accelerations, which require lower simulator displacement, can be emulated by the translational channel of the motion system. It should be noted that mathematically the high-frequency content can also be simulated by performing tilt-coordination. However, this will induce large false rotational cues when the rotational rate lies above the human perception threshold [95][77][45]. To split the input signal into the respective low and high frequency content, a first order low- and high-pass filter are present, their transfer functions are given in Equation (2.1) and Equation (2.2) respectively.

$$LP(s) = \frac{K_{lp}}{\frac{s}{\omega_{lp}} + 1} \tag{2.1}$$

$$HP(s) = \frac{K_{hp} \cdot s}{s + \omega_{hp}} \tag{2.2}$$

Two distinctive parameters are present in the transfer function: the gain "*K*" and the cut-off frequency ω. The gain determines the scaling, shifting Bode plot up and down, while the cut-off frequency determines the pass and stop-band of the filter. The Bode plot of the low-pass and high-pass filter can be found in fig. 2.2.

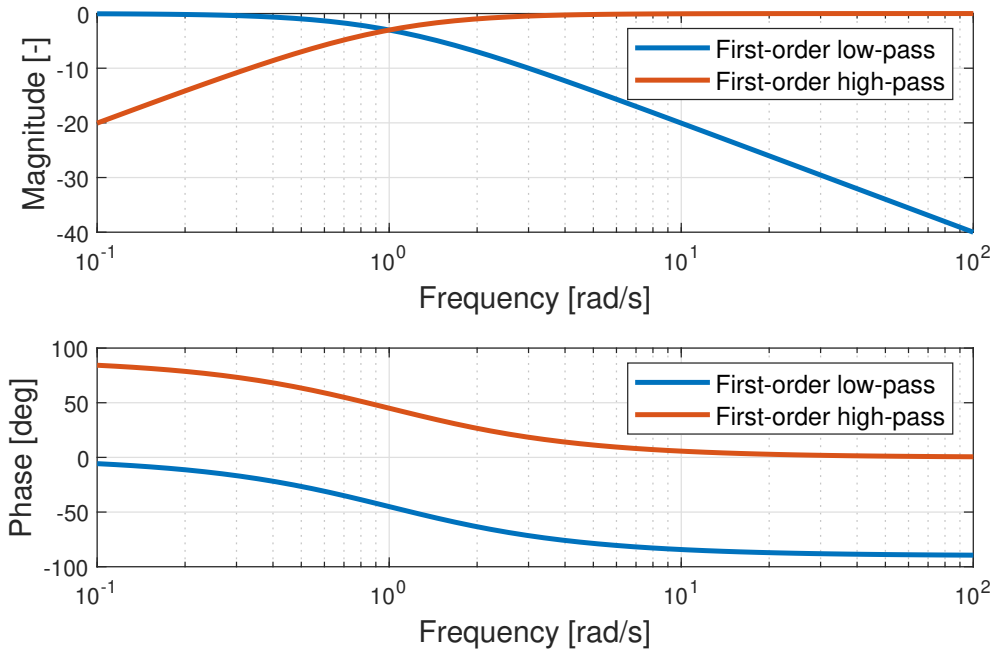


Figure 2.2: Frequency response of 1st-order low- and high-pass filters used to split the CWA input frequencies.

Filter parameters are: $K_{lp} = K_{hp} = 1$ and $\omega_{lp} = \omega_{hp} = 1 rad/s$. When using complementary cut-off frequencies, e.g. $\omega_{lp} = \omega_{hp}$, all input frequencies are utilized. If non-complementary filters are used, the transition in specific forces due to translation and tilt-coordination creates a non-smooth function, which could lead to undesirable motion perception. An example of this behaviour (where $\omega_{lp} = 0.35 rad/s$ & $\omega_{hp} = 4.30 rad/s$) is shown, for a step input on the longitudinal acceleration channel, in fig. 2.3. Remember that this figure shows the output specific force generated by the simulator, this means the transformation, washout and conversion to specific forces are also incorporated.

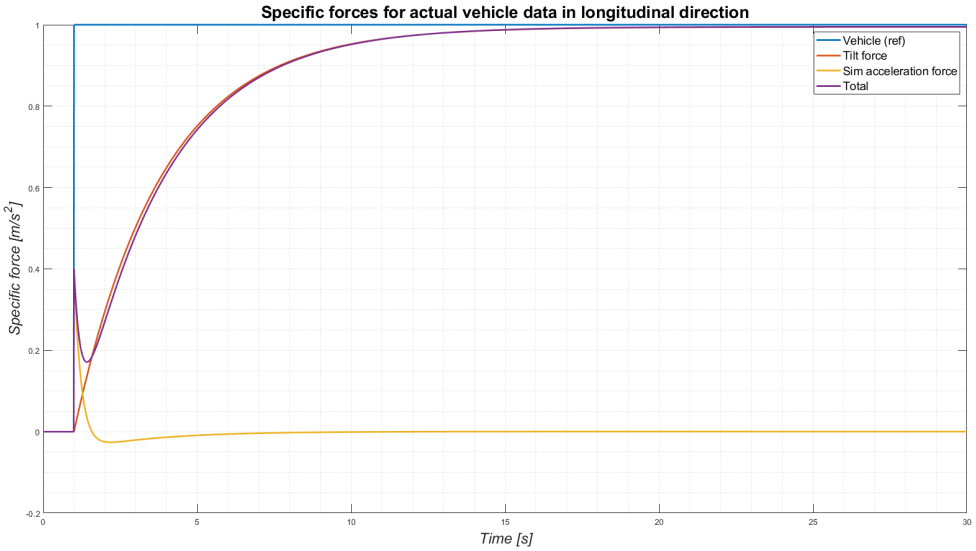


Figure 2.3: Simulated specific force due to a longitudinal acceleration step input to the CWA, featuring non-complementary first order split filters.

2.1.3. Transformation from body-to-inertial reference frame

When using a MCA it is important to keep track of the relevant reference frames. As discussed in Section 1.5, two reference frames are of importance, the vehicle body reference frame and the inertial reference frame, the specific mathematical transformations concerning both frames are also elaborated upon in Section 1.5. Two transformations are present in the algorithm presented in fig. 2.1. One transformation featured in the translational channel and one in the rotational channel.

Translational channel

The input to the translational channel is defined in the body coordinates of the vehicle, in this frame the driver perceives the motion, it thus makes sense to apply the first order low- and high-pass filter in this frame, which also prevents cross-coupling of the perceived motion stimuli. To show the effect of cross-coupling, the high-pass translational channel will be singled out, as shown in fig. 2.4 [77]. Whereas, washing out the motion, e.g. making sure the simulator returns to its neutral position after a manoeuvre, should occur in the inertial reference frame to avoid drift on the simulator position. This drift can occur due to numerical integration/derivation errors.

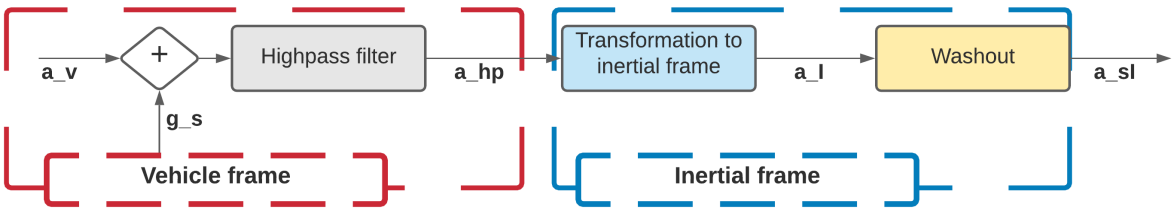


Figure 2.4: High-pass translational channel of the CWA algorithm presented by Reid and Nahon [77].

The mathematical substantiation that highlights how cross-coupling can occur is given in Equation (2.3)-(2.7).
When the a_b is high-pass filtered in the inertial frame (e.g. "Highpass" filter block is located aft the transformation) we get for $a_{b,I}$ (the simulator acceleration defined in inertial ref frame). One should note that the washout term is omitted for simplification (it has no effect on the outcome):

$$\vec{a_{b,I}} = tf_{hp}(\vec{a_I}) \cdot \vec{a_I} \;\; and\; we\; know : \vec{f} = \vec{a} - \vec{g} \tag{2.3}$$

$$\vec{f_s} = \mathrm{T}_{b,I} \cdot \vec{a_{b,I}} - \vec{g_b} \tag{2.4}$$

Where f_s denotes the specific force in simulator body reference frame, and $\mathrm{T}_{b,I}$ denotes the transformation matrix from inertial to simulator body reference frame. Substituting Equation (2.3) in Equation (2.4) gives:

$$\vec{f_s} = \mathrm{T}_{b,I} \cdot tf_{hp}(\vec{a_I}) \cdot \vec{a_I} - \vec{g_b} \tag{2.5}$$

From the figure we also know:

$$\vec{a_I} = \mathrm{T}_{I,b} \cdot [\vec{a_b} + \vec{g_b}] \tag{2.6}$$

$$\Leftrightarrow \vec{f_s} = \mathrm{T}_{b,I} \cdot tf_{hp}(\vec{a_I}) \cdot \mathrm{T}_{I,b} \cdot [\vec{a_b} + \vec{g_b}] - \vec{g_b} \tag{2.7}$$

From Equation (2.3) it can be deduced that if no cross-coupling is to occur, e.g. the x-component of f_s consists of only the x-component of a_I, the term $\vec{f_s} = \mathrm{T}_{b,I} \cdot tf_{hp}(\vec{a_I}) \cdot \mathrm{T}_{I,b}$ should be diagonal. In Section 1.5, the transformation matrices were computed, $\mathrm{T}_{b,I}$ and $\mathrm{T}_{I,b}$ are defined in Equation (2.8) and Equation (2.9):

$$\mathrm{T}_{b,I} = \begin{pmatrix} c(\theta)c(\psi) & c(\theta)s(\psi) & -s(\theta) \\ s(\phi)s(\theta)c(\psi) - c(\phi)s(\psi) & c(\phi)c(\psi) + s(\phi)s(\theta)s(\psi) & s(\phi)c(\theta) \\ c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) & c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi) & c(\phi)c(\theta) \end{pmatrix} \tag{2.8}$$

$$\mathrm{T}_{I,b} = \begin{pmatrix} c(\theta)c(\psi) & c(\psi)s(\theta)s(\phi) - s(\psi)c(\phi) & c(\psi)s(\theta)c(\phi) + s(\psi)s(\phi) \\ c(\theta)s(\psi) & s(\phi)s(\theta)s(\psi) + c(\phi)c(\psi) & c(\phi)s(\theta)s(\psi) + s(\phi)c(\psi) \\ s(-\theta) & s(\phi)c(\theta) & c(\phi)c(\theta) \end{pmatrix} \tag{2.9}$$

In which "c" and "s" denote the cosine and sine respectively. Looking at the matrices in Equation (2.8) and Equation (2.9) it can be deduced that for $\vec{f_s} = \mathrm{T}_{b,I} \cdot tf_{hp}(\vec{a_I}) \cdot \mathrm{T}_{I,b}$ to be diagonal, either the filter is applied before transforming to the inertial frame (e.g. transformation not present in the equation), or $\mathrm{T}_{b,I} \approx I$ and $\mathrm{T}_{I,b} \approx I$ which is the case when the simulator angles are small, e.g. $\phi \approx \theta \approx \psi \approx 0$ such that $s(\phi, \theta, \psi) \approx 0$ & $c(\phi, \theta, \psi) \approx 1$. The same philosophy holds regarding the filter present in the rotational channel.

Rotational channel

The vehicle rotational rates also need to be transformed to its respective Euler rates. The transformation is given in Equation (2.10).

$$\dot{\beta}_I = J_{Is} \cdot \dot{\beta}_v \tag{2.10}$$

With the Jacobian $J_{I,b}$ equal to Equation (2.11). For which the mathematical derivation is given in Section 1.5.

$$J_{I,b} = \begin{pmatrix} cos(\theta)cos(\psi) & -sin(\psi) & 0 \\ cos(\theta)sin(\psi) & cos(\psi) & 0 \\ -sin(\theta) & 0 & 1 \end{pmatrix} \tag{2.11}$$

2.1.4. Washout

Translational channel

As explained earlier, the translational motion bandwidth of a simulator is very limited w.r.t. the actual vehicle range of motion. Therefore it is necessary that the simulator moves back to its neutral position

after performing a manoeuvre. To establish this behaviour, the single high-pass filter used in the frequency divider is not sufficient. An extra washout filter is necessary that induces an acceleration back to neutral. In order to examine what order is required, the final value theorem can be used [67]. The response to an input in the time domain can be defined as the convolution shown in Equation (2.12).

$$y(t) = \int_0^t h(t-\tau)u(\tau)d\tau \tag{2.12}$$

Where $y(t)$ is the output due to an input u at time τ. In order to know the steady state response characteristics, e.g. $t \to \infty$, the limit operator is applied to Equation (2.12). However, dealing with convolutions in a limit is unnecessarily difficult, transforming the convolution integral to the Laplace domain produces a multiplication, see Equation (2.13).

$$\lim_{t\to\infty} y(t) = \lim_{t\to\infty} \int_0^t h(t-\tau)u(\tau)d\tau \Leftrightarrow \lim_{s\to 0} s \cdot U(s) \cdot Y(s) \tag{2.13}$$

To check what order the extra washout filter should have, 3 scenarios will be investigated. It should be noted that for this specific final value analysis, standard order filters are used instead of the CWA filters presented by [20]. In [20] the third-order filter is represented by a multiplication of a first-order high-pass filter with low cut-off frequency ($\omega_h p = 0.2 - 0.3 rad/s$) with a second-order high-pass filter.

1. Only the 1st-order high-pass filter, no washout.
2. Extra 1st-order washout, behaving like a 2nd-order filter.
3. Extra 2nd-order washout, behaving like a 3rd-order filter.

The equations used are given in Equation (2.14) - (2.16).

$$HP_{1st}(s) = \frac{K_{hp} \cdot s}{s + \omega_{hp}} \tag{2.14}$$

$$HP_{2nd}(s) = \frac{s^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{2.15}$$

$$HP_{3rd}(s) = \frac{s^3}{s^3 + 3\omega_n s^2 + 3\omega_n^2 s + \omega_n^3} \tag{2.16}$$

Since the input to the translational channel is on acceleration level, it has to be integrated twice to give insight in the behaviour at position level, e.g. $U(s) = \frac{1}{s} \cdot \frac{1}{s^2} = \frac{1}{s^3}$. Applying the final value theorem to each of these equations, results in the steady state values found in Equation (2.17) - (2.19).

$$\lim_{t\to\infty} y_{1st}(t) = \lim_{s\to 0} s \frac{1}{s^3} \frac{K_{hp} s}{s + \omega_{hp}} = \infty \tag{2.17}$$

$$\lim_{t\to\infty} y_{2nd}(t) = \lim_{s\to 0} s \frac{1}{s^3} \frac{s^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} = \frac{1}{w_n^2} \tag{2.18}$$

$$\lim_{t\to\infty} y_{3rd}(t) = \lim_{s\to 0} s \frac{1}{s^3} \frac{s^3}{s^3 + 3\omega_n s^2 + 3\omega_n^2 s + \omega_n^3} = 0 \tag{2.19}$$

From this calculation it can be clearly seen that only a 3rd-order filter returns the simulator to its neutral position, next to the 1st-order high-pass filter, a 2nd-order washout filter should be applied. This is highlighted in fig. 2.5 where the simulator position, velocity and acceleration response to a step input on acceleration level is plotted.

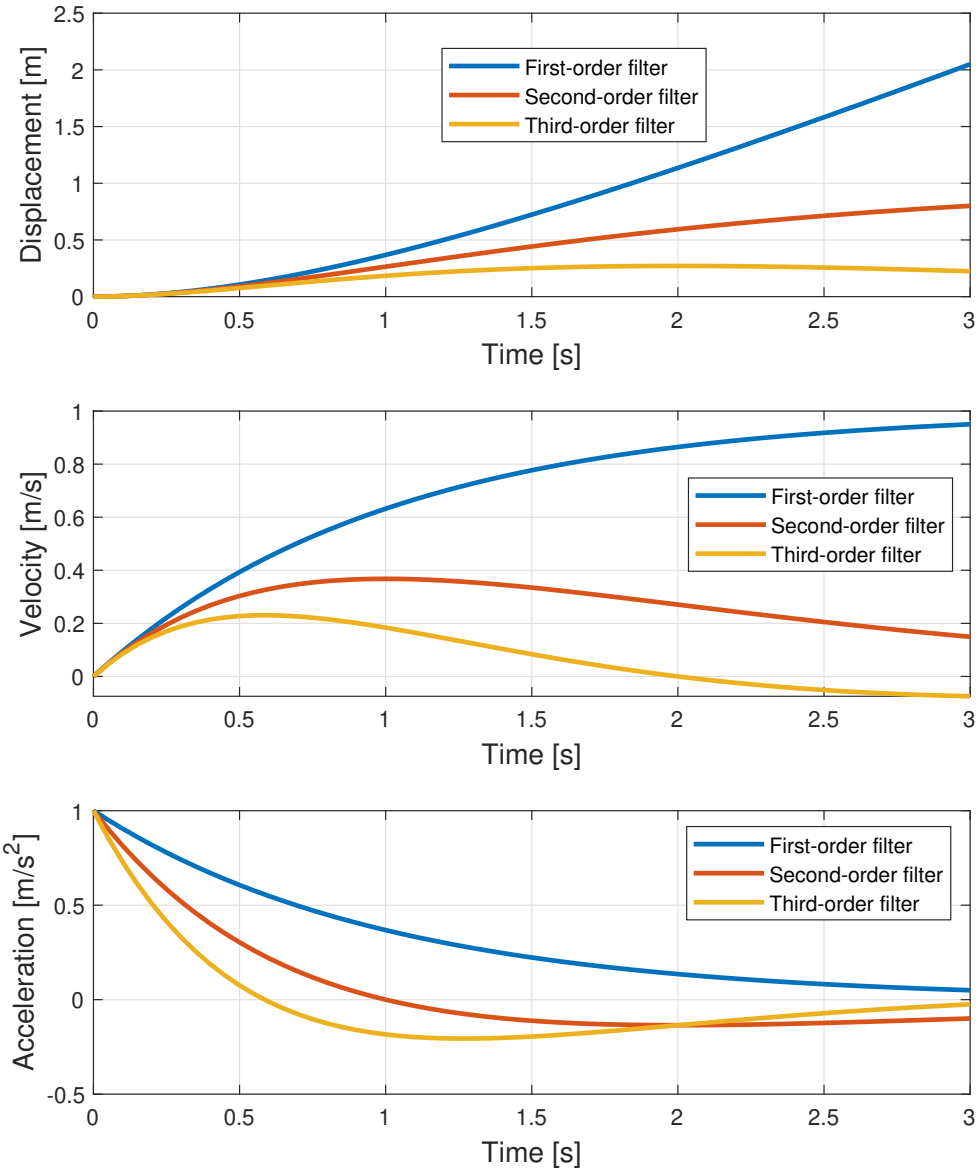


Figure 2.5: Influence of washout strategy on simulator response due to a step input on acceleration level.

Rotational channel

Rotational car motions, e.g. roll, pitch and yaw-rate, can be emulated by the equivalent simulator rotational motion. Two key differences between aircraft and car movement are present. First of all, the rolling and pitching motion range of a car is limited compared to the motion range of an aircraft. Next to this, in absence of a long-lasting banked turn, a car cannot perform a coordinated roll motion, this means that the specific force vector does not align with the driver cell (as is the case in an aircraft coordinated turn). This makes it possible to feed the input pitch and roll rate directly to the algorithm without applying a filter. According to [80], this significantly enhances motion cueing performance. Mind that an analogous reasoning cannot be followed when considering the yaw rate. Due to a car's extensive yaw motion range, the rate has to be filtered, unless an unlimited range of motion yaw table is

present. Therefore a 2nd-order washout filter is present in the yaw-rate channel, the generic 2nd-order filter is shown in Equation (2.20).

$$HP_{2nd} = \frac{s^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{2.20}$$

Analogous to Equation (2.19) it can be proven that a step input on the yaw rate ($U(s) = 1/s$) is washed out by a 2nd-order filter on the yaw angle level ($U(s)$ has to be integrated once $U(s) = 1/s^2$). The mathematical proof is given in Equation (2.21).

$$\lim_{t\to\infty} y(t) = \lim_{s\to 0} s \cdot HP_{2nd} \cdot U(s) = \lim_{s\to 0} s \frac{1}{s^2} \frac{s^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} = \lim_{s\to 0} \frac{s}{s^2 + 2\zeta\omega_n s + \omega_n^2} = 0 \tag{2.21}$$

2.1.5. Transformation from inertial-to-body reference frame

The outputs of the CWA are the simulator accelerations and rotational rates for all simulator DoF's ($\phi/\theta/\psi$) in the inertial reference frame, integrating the accelerations twice gives the full 6 DoF simulator displacement. However, in order to perform a quantifiable investigation on motion cueing performance, a comparison needs to be made between the reference perceived specific forces and the simulated specific forces. Therefore, the simulator accelerations need to be transformed to the equivalent accelerations in the body reference frame. The accelerations, e.g. specific forces, in simulator reference frame are given by the equation in Equation (2.22):

$$\vec{f_s} = \mathrm{T}_{b,I} \cdot \vec{a_{b,I}} - \vec{g_b} \tag{2.22}$$

With $\mathrm{T}_{b,I}$ equal to Equation (2.8). From Equation (1.4), and using the small angle approximation, $cos(\phi) \approx 1$ and $f_{z,tilt} \approx g$, the equations in Equation (2.23) and Equation (2.24) hold.

$$f_{x,tilt} = g \cdot sin(\theta) \tag{2.23}$$

$$f_{y,tilt} = -g \cdot sin(\phi) \tag{2.24}$$

For the rotational rates the transformation, found in Equation (2.25), applies.

$$\dot{\beta}_s = J_{b,I} \cdot \dot{\beta_{b,I}} \tag{2.25}$$

The Jacobian $J_{b,I}$ is equal to the one found in Equation (1.19).

2.2. Classical Washout Algorithm Performance

Section 2.1 explained the complete working of the CWA. However, one critical point is not yet explained. A lot of variables can be tweaked to significantly alter the behaviour and performance of the algorithm, an example was already given in fig. 2.3 which showed the result of choosing non-complementary cut-off frequencies for the low- and high-pass filter. This section aims to give a clear overview of how a change in a certain parameter changes the overall performance. First the individual parameters for each specific filter will be investigated using simple synthetic input signals. After these results are presented, the output to real vehicle data will be examined.

2.2.1. Filter analysis

Two different kind of filters are present in this algorithm, 1st- (low-pass and high-pass) and 2nd-order high-pass filters (translational and rotational washout). It is assumed that investigating the two types of filters, present in the translational channel, will give a complete overview on how the filter parameters relate to the system response. In order to compare the responses, the following unity baseline parameters are used throughout this investigation, they can be found in table 2.1.

Table 2.1: Unity filter parameters for the translational channel of the CWA in x-direction.

Parameter	Base Value	Unit
K_{hp}	1	[-]
ω_{hp}	1	$[rad/s]$
$K_{hp_{wash}}$	1	[-]
$\omega_{hp_{wash}}$	1	$[rad/s]$
$\zeta_{hp_{wash}}$	1	[-]

The influence of changing the gain of the 1st-order filter will be examined first. The expectation is that the initial acceleration of the simulator will be scaled with respect to the step input, however due to the 2nd-order washout filter this initial response will be damped out quickly. Due to the larger amplitude response, effective for larger gains, the (negative) accelerations due to washout are expected to be more significant, this effect could ultimately result in false cues: a motion direction in which no motion is expected. It is also expected that larger gains result in larger simulator displacement. This could lead to disruptive false cues when the simulator reaches its limits and as such is not desirable. The simulator acceleration response and the simulator displacement due to a unit step input on the x-acceleration channel are shown in fig. 2.6.

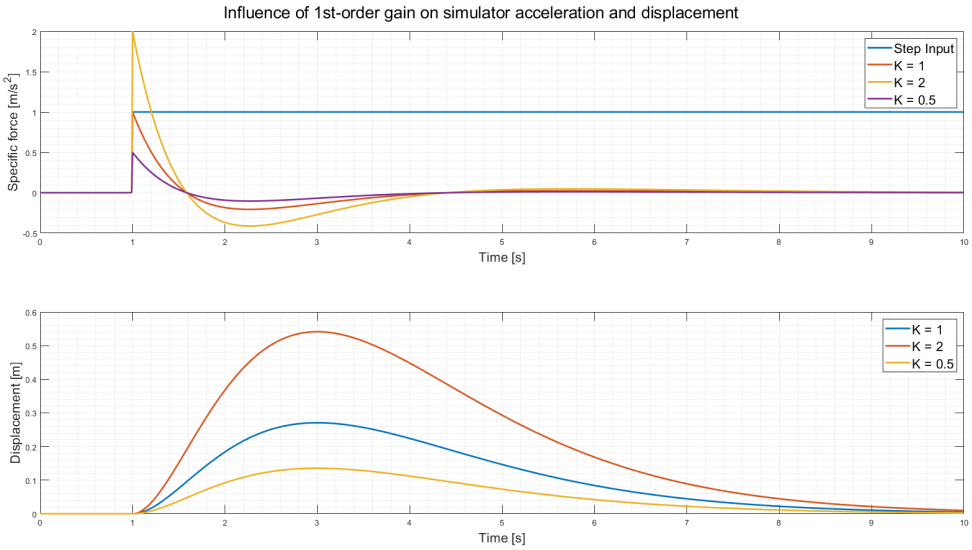


Figure 2.6: Specific force and simulator displacement response for a high-pass filter gain equal to $K_{hp} = [0.5, 1, 2]$.

From the figure it can clearly be seen that there is a strong correlation between the gain and the washout effect, higher gains results in larger changes in acceleration due to the washout. As expected larger gains also increase the simulator displacement significantly. A gain of 2 was also tested, when thinking about the real life implication of having a gain larger than ”1”, the use of such a gain is questionable. The simulator response would be an overreaction to the actual input signal, this can and, as experimentally proved, will result in unrealistically strong motion perception [46].

The other parameter that can be altered in the 1st-order filter is the cut-off frequency. Changing this variable will shift the magnitude and phase plots shown in fig. 2.2 to the right for larger ω_{hp} and to the left for decreasing values of ω_{hp}. Practically this means that the pass-band of the filter decreases with increasing ω_{hp}, meaning less of the low frequency signal content will be simulated by simulator translational acceleration and vice versa. It is expected that increasing ω_{hp} will result in a faster decline of motion simulation after the initial response, e.g. faster transients resulting in smaller simulator displacements. Smaller values of ω_{hp} are expected to spread out the motion simulation, e.g. slower transients resulting in larger simulator displacements. These expectations are confirmed when looking at fig. 2.7.

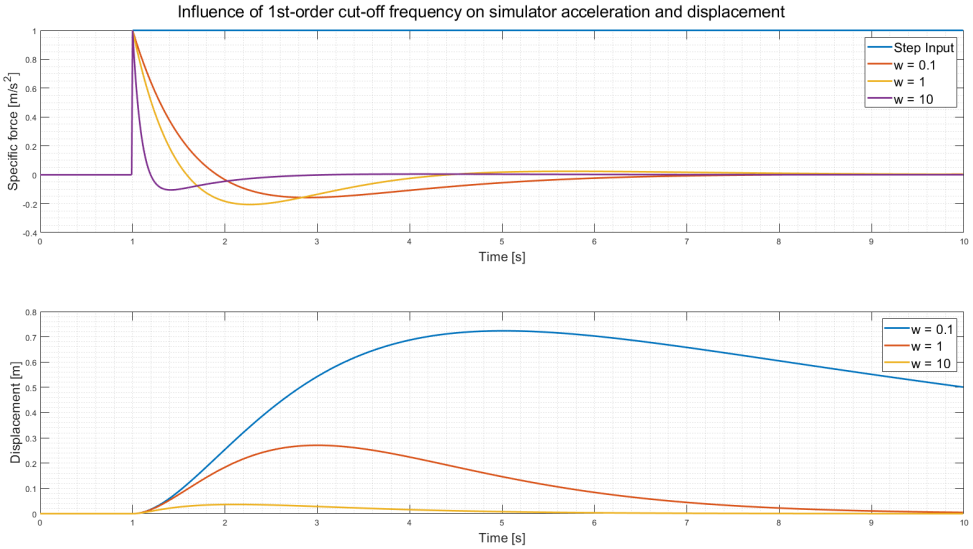


Figure 2.7: Specific force and simulator displacement response for cut-off frequencies $\omega_{hp} = [0.1, 1, 10]\ rad/s$.

The same analogy can be followed when investigating the performance of the 2nd-order washout filter. Three parameters can be altered in this filter: the gain, the cut-off frequency and the damping coefficient. Since the washout filter is of 2nd-order, frequencies not included in the pass-band are being attenuated twice as fast than that of a unity first order high-pass filter. This is highlighted in the Bode plot of a unity 2nd-order (e.g. all parameters equal 1) filter shown in fig. 2.8.

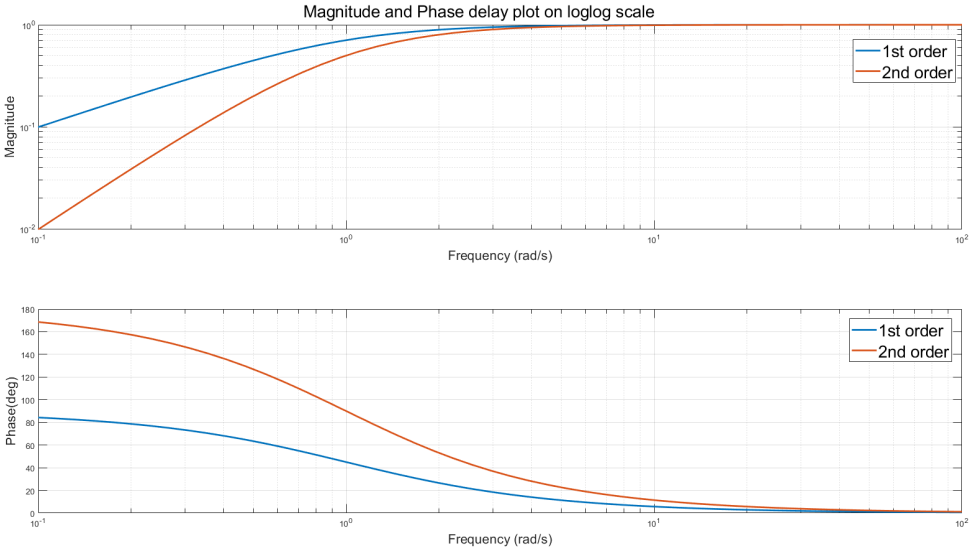


Figure 2.8: Frequency response plots of unity first and second order high-pass filter.

This means that the choice of cut-off frequency will have a stronger effect on the signal when compared to the first order filter, this is visible when comparing the response in fig. 2.7 with fig. 2.9. Because the two filters are placed in series, altering the gain of the 2nd-order filter will produce the same results as discussed previously in the 1st-order case. Changing the frequency and damping coefficient do have different effects though, a higher cut-off frequency results in stronger washout effects resulting in faster and thus stronger perceived transient motion, this could lead to a degradation of motion cue-

ing as false cues are more vividly present. As explained earlier, faster transients correlate with lower simulator displacement, the opposite occurs for lower values of $w_{n_{2nd}}$

If the system is underdamped, $\zeta < 1$, a low frequency sinusoidal washout effect will occur which leads to larger simulator displacement. Overdamping the system will lead to a softer washout response, e.g. a softer transient response which ultimately leads to a smaller displacement than is the case for the critically damped, $\zeta = 1$ or underdamped system. For both parameters the respective responses can be found in fig. 2.9 and fig. 2.10.

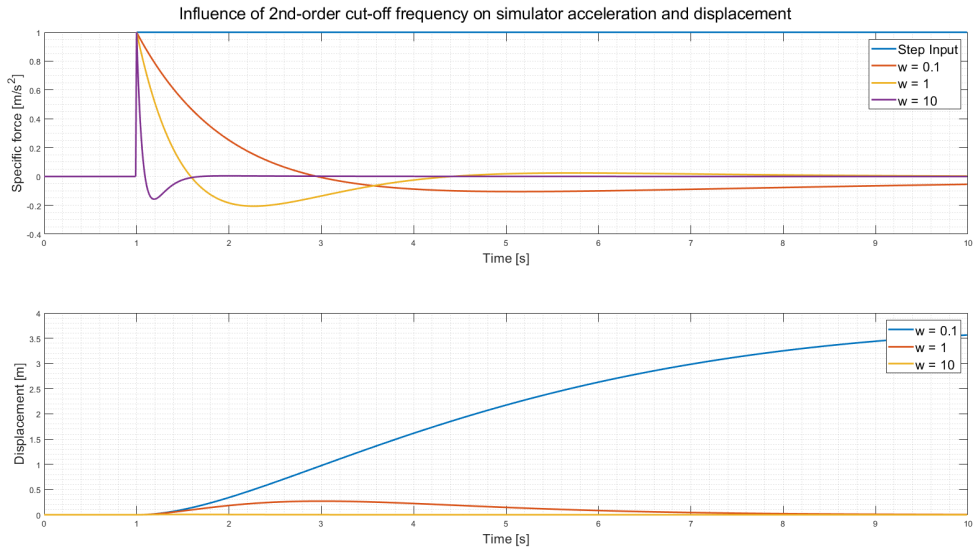


Figure 2.9: Specific force and simulator displacement response for cut-off frequencies $\omega_{hp} = [0.1, 1, 10]\ rad/s$ in the 2nd-order washout filter.

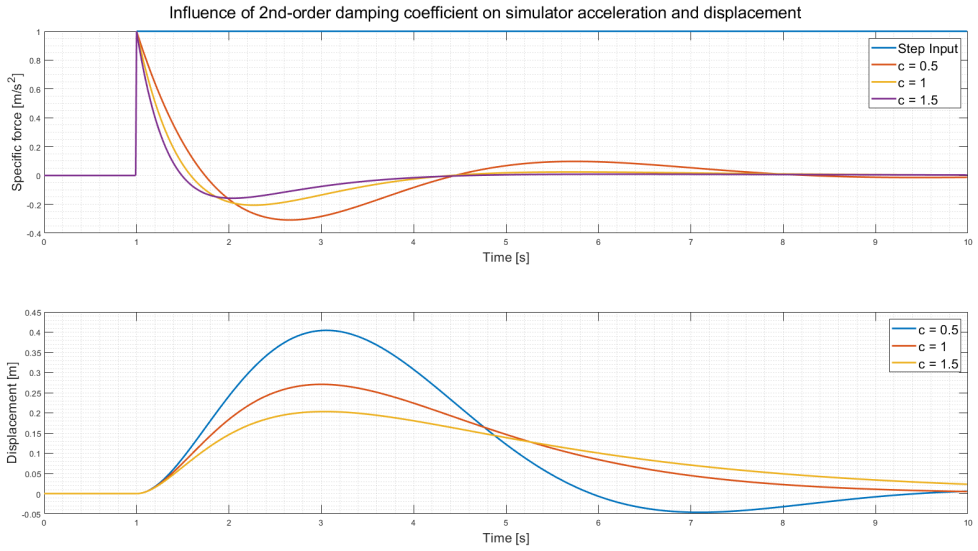


Figure 2.10: Specific force and simulator displacement response for damping coefficients $\zeta_{hp} = [0.5, 1, 1.5]$ in the 2nd-order washout filter.

2.2.2. Key points filter investigation

The goal of an MCA is to, as closely as possible, match the specific force cues one would perceive in the vehicle by simulator motion without reaching the simulator limits. While keeping the amount of

false cues to a minimum. Strong false cues have the ability to destroy the performance of the MCA completely, even if the average specific force error is small. In the previous examples a step input on the acceleration equal to $1m/s^2$ was considered, when trying to simulate an actual vehicle much higher accelerations and decelerations can occur. In order to remain within simulator limits, the filter parameters need to be properly tuned. When one starts to alter the values of these parameters, it is helpful if a guess can be made on how the change will affect the cueing behaviour. Therefore a small summary of the results found in the previous subsection is presented [41].

1. Reducing surge and sway displacement
   1.1. Decrease K_x/K_y
   1.2. Increase $\omega_{hp_x}/\omega_{hp_y}$, this also applies to the 2nd-order washout, however keep in mind that the washout characteristics also change correspondingly.
   1.3. Decrease $\omega_{lp_x}/\omega_{lp_y}$, larger tilt coordination increases surge, sway displacement
   1.4. Increase ζ
2. Reduce angular displacement
   2.1. Increase K for roll, pitch and yaw
   2.2. Decrease ω_{hp} for roll, pitch and yaw
   2.3. Decrease ζ for roll, pitch and yaw

2.2.3. CWA performance real vehicle data

Up until now, analysis of how the CWA works was done by singling out a certain channel, apply an input to it and see how it behaves on the output. However, when simulating an actual vehicle, all channels get used at the same time. Because the channels influence each other the results might not turn out as expected. And since this the following analysis is performed offline without an actual simulator, the assessment of performance can only be done by comparing the specific forces and the angular rates present in vehicle and simulator. This is deemed to be sufficient, a lot can already be said without having a real simulator present. Tuning the filter parameters will be difficult however, usually the parameters are tuned by trial and error based on motion cueing performance scores given by the pilot. Since this is not possible, cut-off frequencies and damping ratios are based on the parameters used by BMW, these are shown in table 2.2.

Table 2.2: Filter parameters used in the classical washout algorithm, values obtained through BMW.

Parameter	Base Value	Unit
$K_{x,y,z}$	*[0.11, 0.11, 0.11]*	[-]
$K_{roll,pitch,yaw}$	*[1.0, 1.0, 0.2]*	[-]
$K_{tilt-coord}$	*[0.35, 0.35, 0.35]*	[-]
ω_{hp}	*[1.50, 2.00, 1.50]*	$[rad/s]$
ω_{lp}	*[1.50, 2.00, 1.50]*	$[rad/s]$
$\omega_{hp_{wash}}$	*[0.30, 0.30, 0.30]*	$[rad/s]$
$\zeta_{hp_{wash}}$	*[1.50, 1.50, 1.50]*	[-]
$\omega_{yaw_{wash}}$	*0.1*	$[rad/s]$
$\zeta_{yaw_{wash}}$	*1.50*	[-]

The values in this table represent the parameter value in x,y,z-direction and value for roll, pitch and yaw respectively. It can also be noted that complementary filters will be used for which ω_{hp} = ω_{lp} and only a 2nd-order washout filter for the yaw rate is used with a very low cut-off frequency, it is reasoned that scaled down reference tracking is more important than high frequency yaw tracking. Because the roll and pitch rate are generally small in cars they do not have to be filtered.
The tuned parameters were utilized for a simulator with dimensions found in table 2.3.

Table 2.3: Simulator specifications used for CWA tuning [28].

Hexapod					
x_H	$\pm 0.28m$	$\dot{x}_H$	$\pm 2m/s$	$\ddot{x}_H$	$\pm 25m/s^2$
y_H	$\pm 0.25m$	$\dot{y}_H$	$\pm 1.7m/s$	$\ddot{y}_H$	$\pm 25m/s^2$
z_H	$\pm 0.22m$	$\dot{z}_H$	$\pm 1.6m/s$	$\ddot{z}_H$	$\pm 25m/s^2$
ϕ_H	$\pm 20°$	$\dot{\phi}_H$	$\pm 135°/s$	$\ddot{\phi}_H$	$\pm 2500°/s^2$
θ_H	$\pm 20°$	$\dot{\theta}_H$	$\pm 130°/s$	$\ddot{\theta}_H$	$\pm 2000°/s^2$
ψ_H	$\pm 20°$	$\dot{\psi}_H$	$\pm 135°/s$	$\ddot{\psi}_H$	$\pm 3000°/s^2$

A vehicle input data set, provided by BMW [70], is used. A snapshot of the translational and rotational input signals are given in fig. 2.11 and fig. 2.12, the complete signal is given in fig. A.1 and fig. A.2.

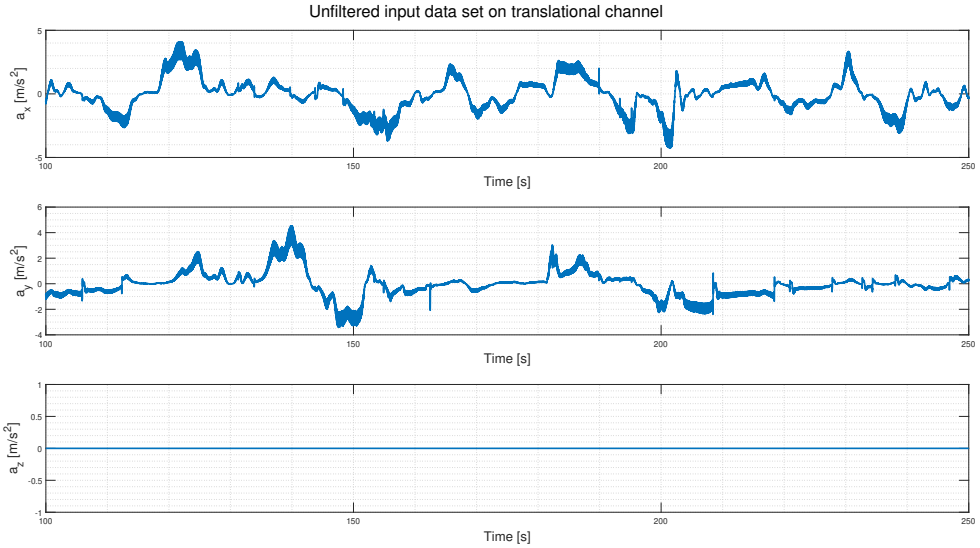


Figure 2.11: Translational vehicle data from BMW test run.

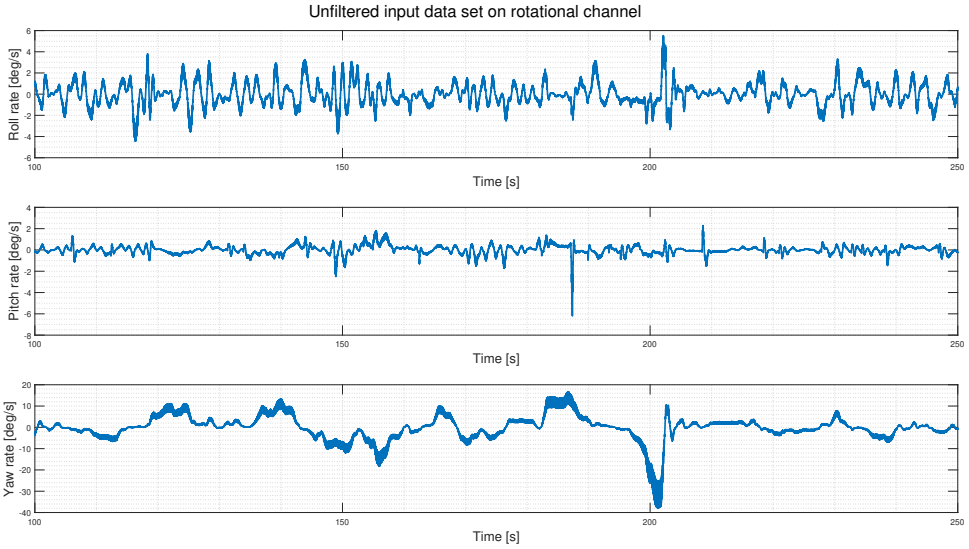


Figure 2.12: Rotational vehicle data from BMW test run.

Because these measurements are noisy (thick lines in the curve), around $50Hz$, they are 1Hz low-pass filtered before they are used as input. One thing that can be noted as well is the lack of acceleration data in z-direction, this occurred due to a logging mistake. A snapshot of the resulting simulator specific forces and rotational rates to this set of inputs is given in fig. 2.13 and fig. 2.14, the full response can be found in fig. A.3 and fig. A.4.

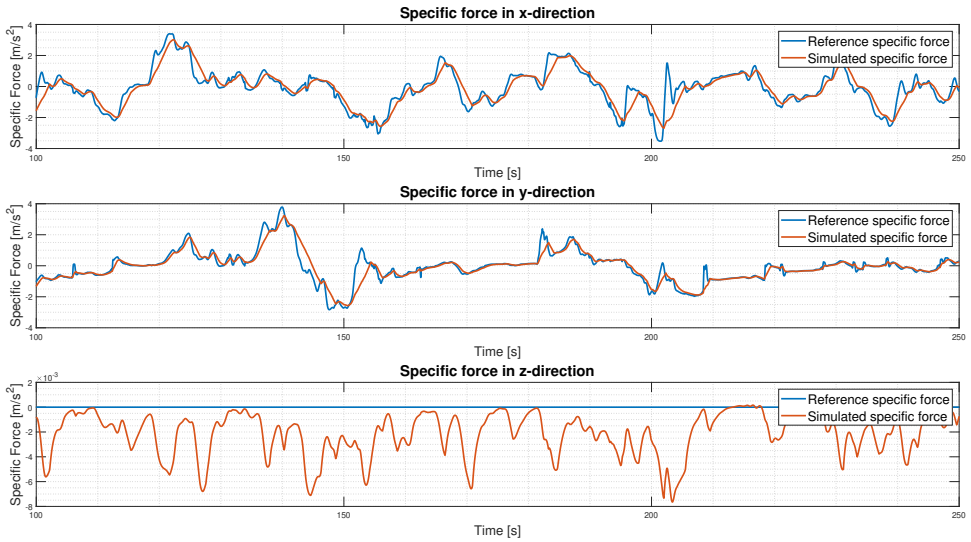


Figure 2.13: Reference vs simulated specific forces in body coordinate reference frame in x, y, and z-direction.

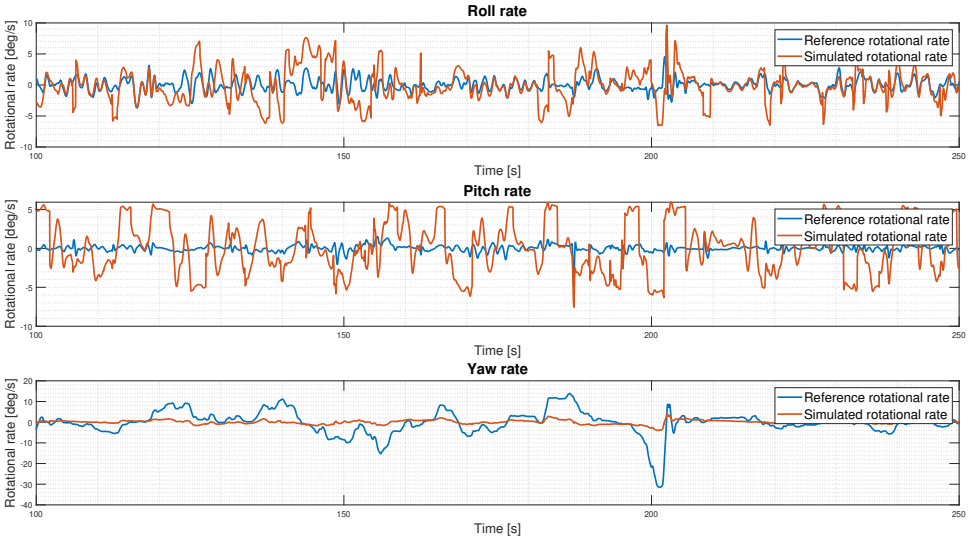


Figure 2.14: Reference vs simulated rotational rates in body coordinate reference frame for roll, pitch, and yaw.

A snapshot of the resulting simulator displacements and rotational simulator angles is given in fig. 2.15 and fig. 2.16, the full translational and rotational displacement is presented in fig. A.5 and fig. A.6.

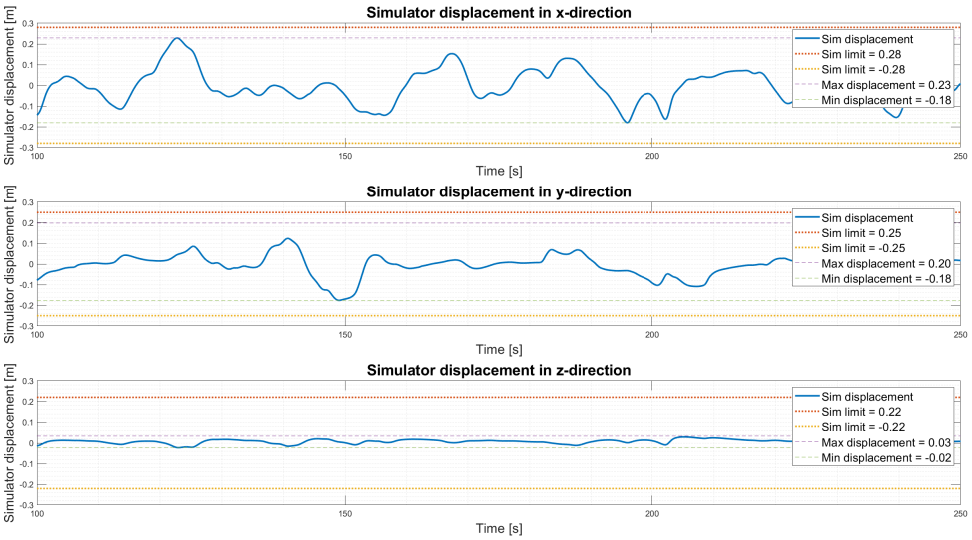


Figure 2.15: Simulator displacement in x, y, and z-direction tracking real vehicle data using CWA.

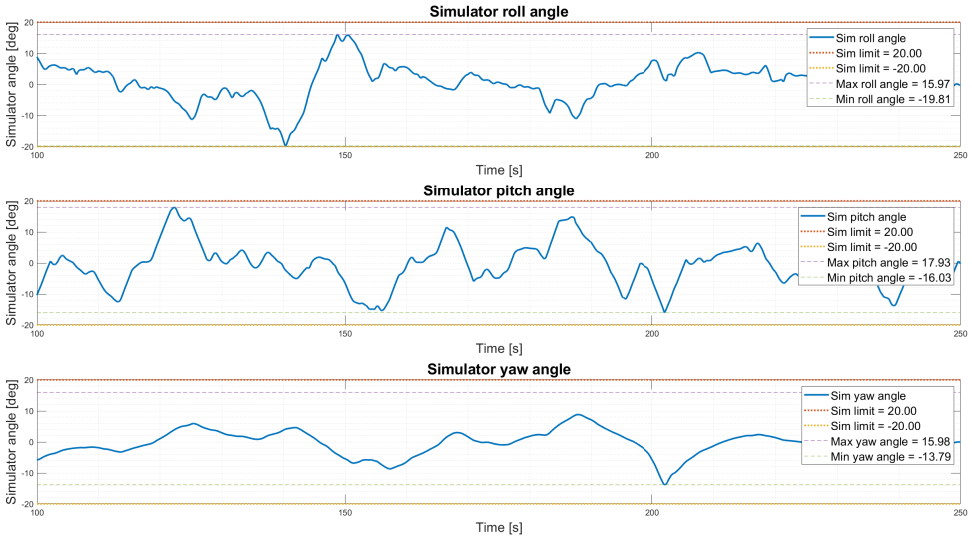


Figure 2.16: Roll, pitch and yaw simulator angles tracking real vehicle data using CWA.

From the displacement figures it can be seen that with the set of parameters presented in table 2.2 the bandwidth of the motion platform is almost maximally utilized. However, these graphs do not say much about the performance of the tuning. Figure 2.13 and fig. 2.14 is what should be investigated. Two things that one can notice right away, at first glance it looks like the simulated specific forces in x- and y- direction are actually quite good and follow the reference quite well, the one in z-direction not so much. However, the scale of the latter one is 10^{-3}, specific forces on this scale are negligible. One should not forget that the acceleration in x- and y-direction is being simulated by the translational acceleration and tilt of the simulator. Due to the limited bandwidth in translational direction, most of the motion is simulated by tilting. This behaviour is shown in fig. 2.17.

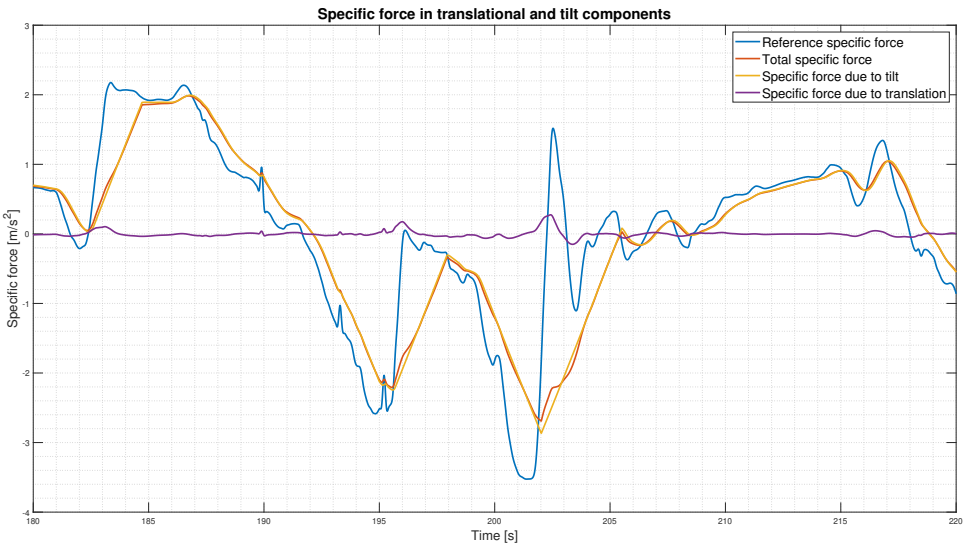


Figure 2.17: Specific force tracking in x-direction using CWA.

This figure gives a little bit more insight into how ”good” the performance of the CWA, in translational direction, actually is. The first thing one can notice is that the majority of the cueing is done by tilting the simulator. This is highlighted when looking at the rotational simulator excitations in fig. 2.16, where rotation angles larger than 10 degrees are common. One can also see that, due to the tilt rate limiter, tracking of the reference specific force is not possible. This results for t = [183s-185s], when the driver start accelerating, in a missing cue. For t = [202s-206s], a lot of false cues are present. This will deteriorate the experienced performance of the motion cueing significantly. It could be argued that in this case removing the tilt-rate limiter, but limiting the amount of tilt the simulator can attain would result in better cueing quality.

Secondly, it seems that the performance of the rotational rate channel is very poor. This has a very good reason. The simulated rotational rates that are presented are a summation from the rotational channel output rates and the rates due to tilt-coordination. Even though in this simulation a rate limiter on the tilt-coordination is present which limits the rate to be below the threshold of perception, the tilt-coordination rate is included completeness. In order to deduce the performance of the rotational channel the motion is split in both counterparts. A snapshot of the roll rate is plotted in fig. 2.18, which confirms that the CWA does what it is supposed to do (the same holds for the pitch rate).

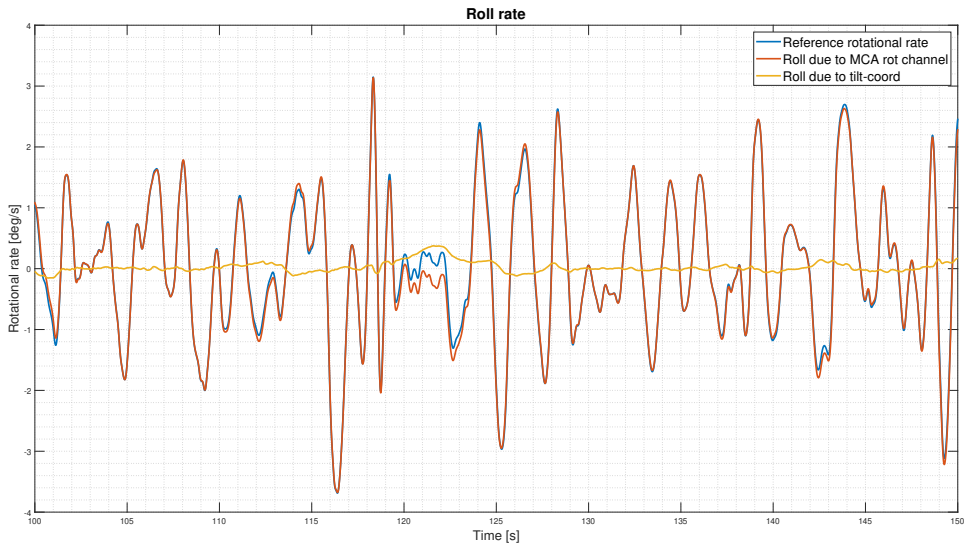


Figure 2.18: Simulator roll rate split-up in its roll motion due to tilt-coordination and the CWA rotational channel component.

The simulated yaw motion is a scaled down version of the reference. This is due to the limited yaw motion a hexapod simulator can provide in comparison to the reference yaw. As explained earlier a solution to this problem would be to install the hexapod on a yaw table. Because this comes at increased cost the added value should be carefully examined.

2.3. Advantages & shortcomings

To conclude this chapter, the main advantages and shortcomings of the CWA are presented. As one could read in the previous sections, the general working of the CWA is easy to understand. When applied on a real simulator, changes in between simulations are easily made with the performance losses/gains easily guessed. The algorithm is also computationally efficient and shows no problem to run real-time at 100Hz [28], making the CWA perfect for running quick simulations. Due to the lack of real-time alternatives and more than 50 years of development[85], CWA became the benchmark algorithm for motion cueing. Quite some disadvantages are present as well. First of all tuning of the algorithm can be quite cumbersome, and since the algorithm is linear it needs to be tuned for the worst case scenario as reaching a simulator limit can be very destructive for motion perception. This has as effect that all other motions are scaled down heavily and can be perceived as numb. Secondly, a constant trade-off has to be made when altering the values of the filters, e.g. when altering the gain one has to make the trade-off between less scaling of input signal, which also increases the proportion of false cues as well as increasing simulator displacement. This effect is even more noticeable when considering very high motion range, redundant DoF simulators utilizing a tripod or x-y table. Next to this, very high transient input signals, e.g. acceleration into abrupt coasting, are simulated by a quick negative translational motion which introduces false cues. Lastly, an untouched extension of the algorithm is called pre-positioning, which increases the effective useable motion motion range significantly. However, even when using pre-positioning the algorithm still needs to be tuned for the worst case scenario.

Even though quite some negative points exist, the fact remains that the CWA is a flexible, simple to build control algorithm that works in all scenarios and has proven to do so for the past 50 years. However, in recent years the car industry has put a lot of effort and research in more advanced, optimization-based algorithms. A real-time version of an approach called ”model predictive control” will be examined in Chapter 3.

3

Model Predictive Control

3.1. MPC Paradigm

In Chapter 2 the CWA was described. It was discussed that the CWA is a robust algorithm that provides predictable motion cueing quality. It was also shown that although the algorithm is fast and easily understood, it can be cumbersome to tune. Phase lead and lag, a phenomenon inherently coupled with filtering, induces missing and false cues. Because no explicit constraints can be included, the inputs need to be scaled down which reduces motion of the simulator, thus motion sensation, and reduces the amount of available workspace that is utilized.
For these reasons, more sophisticated and promising techniques have been researched in the last two decades that try to eliminate these issues. One of these techniques, called model predictive control (MPC) or receding horizon control (RHC), explicitly eliminates some of the problems that occur with the CWA and is therefore seeing a lot of attention by researchers in the industry [5][7][28][51]. Dagdelen et al. [22] were the first to research the use of MPC MCA applications. MPC tries to solve an optimization problem each step. Using a model of the system to predict future outputs over a fixed time horizon, e.g. the prediction horizon N_p, it enables the computation of an optimal input sequence over a fixed time horizon, called the control horizon N_c. The optimal control sequence is calculated by minimizing a cost function that incorporates costs on reference tracking, inputs, changes in inputs and final state. Due to simplification and possible disturbances no guarantee can be given that the predicted system state equals the true system state. Therefore, only the first input of the control sequence is used as the input to the system. The optimization is repeated for the next step, with the time horizon shifted one step, hence the name "receding horizon control" [59] [63]. A schematic of this principle is shown in fig. 3.1.

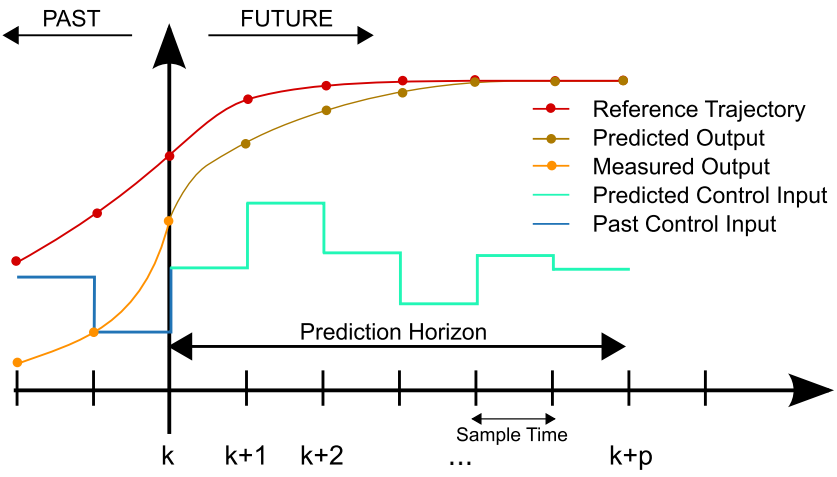


Figure 3.1: Schematic of MPC receding horizon control [8].

A step-by-step plan of the MPC algorithm is given below [76]. The steps and the respective derivations of mathematical equations will be elaborated upon in section 3.2.

MPC Algorithm

1. Establish a kinematic simulator model that links system states, control inputs and system outputs, e.g. 6-DoF kinematic simulator model.
2. Calculate constant evolution and prediction matrices F and S
3. Initialize model parameters: input vector u_0 and initial state x_0
4. Start of **iterative** algorithm:
   (a) Update the reference trajectory for the next prediction horizon, i.e. N_p steps
   (b) Calculate Hessian matrix H and matrix h of the cost function: $\operatorname{argmin} \frac{1}{2} x'Hx + h'x$
   (c) Calculate constraint matrix A and inequality matrix b conform: $A \cdot x \leq b$
   (d) Add slack variables to make output/state constraints soft
   (e) Calculate optimal control sequence u for control horizon N_c
   (f) Only apply u_{k+1}, i.e. input for next time step, as input to the system
   (g) Calculate or measure Δx_{k+1} and y_{k+1}, i.e. the change in state and the total output of the system
   (h) Update initialization parameters u_0 and x_0, total current input and state with applied input and measured/calculated change in state
   (i) Repeat from step (a) until end of simulation

The reason why MPC as MCA has seen much research in the past years is because it can explicitly incorporate constraints on input and state level and use state feedback to improve the prediction on future states in the optimization problem [71]. Also much research is performed to establish faster, more accurate and efficient solvers that can be used in real-time control optimization[**?**][34] [61][88][104]. One of MPC's big advantages, being able to optimize based on the error between predicted output and the reference trajectory over the prediction horizon N_p, is also the main disadvantage of using MPC as MCA. First of all it means that at each step a global optimization problem has to be solved at a frequency of at least $100Hz$ (general running frequency of a driving simulator). Secondly, this means at each step a causal reference of N_p steps for future vehicle motion is required. Vehicle motion is dependent on multiple parameters influenced by the environment and the driver [94]. When an accurate estimate of the reference is required, a driver reference model is required [24]. This dependency is not deterministic but of stochastic nature and is the reason it is hard to predict. Even though this is the case, several studies towards using MPC as MCA have been conducted [26][71][14]. From these studies the effect on cueing quality has been established. Two noteworthy mentions:

1. It has been shown that motion cueing fidelity in a passive drive using oracle logic, i.e. data from a prerecorded drive is available that can be used as perfect reference trajectory, increases significantly when compared to CWA [28]
2. Using oracle logic in real-time simulation is not possible because future driver inputs are not known at each time step. Therefore, a prediction for future motion states as reference is required. Ellensohn et al. heavily simplified this prediction to find out how it affects motion cueing fidelity. In this study it was assumed that the current state of the vehicle is constant over the prediction horizon and is used as constant reference for the next N_p steps. This is a basic strategy, which can be improved significantly. However, even with such a bad prediction, it is experimentally proven that the MPC MCA still outperforms the CWA [26].

From these two points it is assumed that if better prediction for future vehicle motion states exists, cueing fidelity will improve, in the limit converging asymptotically to the oracle fidelity scores. Because

there is a large, yet unexplored region of performance, it is decided that it is important to thoroughly investigate and understand the use of MPC in motion cueing by programming a MPC controller from the ground up using the same reference drive as described in Section 2.2. When successful, this research is anticipated to fundamentally increase the understanding on how MPC works, e.g. show its advantages and limitations. This can be accomplished by comparing the simulation results with the results presented in Chapter 2. Ultimately a working controller that is able to control a simplified model of the system presented in Chapter 2 in table 2.3 in multiple DoF can be used to give an indication on how different prediction models influence motion cueing performance. In other words, it would become a valuable tool that can be used in the future to hypothesize on expected results before conducting real-life experiments.

3.2. MPC-based MCA

In the previous section the working of a general MPC was given. However, in the specific mathematical substantiation several details and proofs are important to gain the knowledge required to understand the implementation of the algorithm. First, two models from two different perspectives are introduced in Subsection 3.2.1. With the models available, the cost function will be presented and the derivation of the general quadratic cost function and its parameters is presented in Subsection 3.2.2. Constraints on inputs, states and changes in inputs need to be defined in function of the decision variable in the cost function, e.g. change in input ΔU, necessary derivations are provided in section 3.2.3. To guarantee stability near the inequality constraint border, soft state/output constraints are compulsory, explanations and derivations regarding the implementation of using slack variables that ensure stability, are presented in Subsection 3.2.4. Combining all these areas defines the complete quadratic optimization problem, in section 3.2.5 the quadratic solver method will be shortly elaborated upon.

3.2.1. Kinematic Simulator Model

As explained before, a sufficiently good kinematic model of the system at hand is paramount to the proper working of the MPC. This kinematic simulator model relates simulator states, applied control inputs and system outputs. In this sense it is important to understand when this kinematic model is described as sufficient. This brings up a trade-off: an accurate kinematic model is required to make accurate causal predictions on future states. However, when considering highly dynamical and high frequency transient systems, one also wants to have a kinematic model that can deliver quick predictions in order to satisfy controller performance and ensure stability. When considering complex systems, higher order complex models are not fast. Therefore a trade-off needs to be made between computational complexity and desired accuracy. Two different kinematic simulator models are mentioned in literature. A simple double integrator model, that describes the motion of the simulator in function of the input on acceleration and rotational acceleration [30][31][32]. As well as a model that combines the double integrator model with the vestibular model presented in section 1.2 [7][22][26][51]. For this research the focus lies on a controller that works in longitudinal and lateral direction, yaw and motion in z-direction are not yet considered at this stage.

Double integrator model

The double integrator model follows directly from the discrete equations of motion of the simulator platform described in Equation (3.1). The equations are given in x-direction, the same logic applies in lateral direction.

$$\begin{aligned} x_{k+1} &= x_k + dt \cdot \dot{x}_k + \frac{1}{2} dt^2 \cdot \ddot{x}_k \\ \dot{x}_{k+1} &= \dot{x}_k + dt \cdot \ddot{x}_k \\ \ddot{x}_{k+1} &= \ddot{x}_k \\ \theta_{k+1} &= \theta_k + dt \cdot \dot{\theta}_k \\ \dot{\theta}_{k+1} &= \dot{\theta}_k \end{aligned} \tag{3.1}$$

As with the CWA, the reference trajectory is given in terms of acceleration/specific force in longitudinal direction and pitch rate. This means the output of the system should also be defined in these

terms. As explained earlier, rotation around the y-axis will produce the sensation of acceleration in x-direction, this has to be incorporated in the model as well. The set of output equations are given in Equation (3.2), incorporating Equation (2.23) using the small angle approximation.

$$\begin{aligned} f_{s_k} &= \ddot{x}_k + g \cdot \theta \\ \omega_{s_k} &= \dot{\theta}_k \end{aligned} \tag{3.2}$$

The system has to be written in state-space, according to Equation (3.3).

$$\begin{aligned} \vec{x}_{k+1} &= A \cdot \vec{x}_k + B \cdot \vec{u}_k \\ \vec{y}_k &= C \cdot \vec{x}_k + D \cdot \vec{u}_k \end{aligned} \tag{3.3}$$

Writing the EOM for acceleration in x-direction and pitch rate in state-space gives Equation (3.4). It should be noted that MPC uses causal information about the system for prediction and control, meaning it is implicitly stated that the output cannot be influenced by the input $\vec{u}_k$ directly, therefore the feedforward matrix D is equal to zero [101].

$$\vec{x}_{k+1} = \underbrace{\begin{bmatrix} 1 & dt & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{A_{I_{dir}}} \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}}_{} \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}}_{} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}}_{A_{I_{rot}}} \cdot \vec{x}_k + \underbrace{\begin{bmatrix} \frac{dt^2}{2} \\ dt \\ 1 \end{bmatrix}}_{B_{I_{dir}}} \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}}_{} \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}}_{} \underbrace{\begin{bmatrix} 0 \\ dt \\ 1 \end{bmatrix}}_{B_{I_{rot}}} \cdot \vec{u}_k$$

$$\vec{y}_k = \begin{bmatrix} 0 & 0 & 1 & g & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \vec{x}_k \tag{3.4}$$

Where $\vec{x}$ and $\vec{u}$ denote the vector of states and inputs. The matrix dimensions are: $A \in \mathcal{R}^{n \times n}$, $B \in \mathcal{R}^{n \times m}$ and $C \in \mathcal{R}^{p \times n}$, "$n$" equals the number of states, "m" the number of inputs and "p" the number of outputs. Including the lateral DoF is done conform Equation (3.5).

$$\begin{aligned} A_{x,y} &= blkdiag(A_{I_{dir}}, A_{I_{dir}}, A_{I_{rot}}, A_{I_{rot}}) \\ B_{x,y} &= blkdiag(B_{I_{dir}}, B_{I_{dir}}, B_{I_{rot}}, B_{I_{rot}}) \end{aligned} \tag{3.5}$$

Equivalently, the C-matrix needs to be rewritten to include the influence of roll angle on lateral specific force and roll angular rate, the new C-matrix is shown in Equation (3.6).

$$C_{x,y} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & g & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -g & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

The state, input and output vector for longitudinal and lateral DoF are equal to Equation (3.7), (3.8) and (3.9) respectively.

$$\vec{x} = \begin{bmatrix} p_x & v_x & a_x & p_y & v_y & a_y & \theta & \dot{\theta} & \phi & \dot{\phi} \end{bmatrix}^T \tag{3.7}$$

$$\vec{u} = \begin{bmatrix} a_x & a_y & \dot{\theta} & \dot{\phi} \end{bmatrix}^T \tag{3.8}$$

$$\vec{y} = \begin{bmatrix} f_{s_x} & f_{s_y} & \omega_{s_y} & \omega_{s_x} \end{bmatrix}^T \tag{3.9}$$

As will be explained in Subsection 3.2.2, the decision variable in the optimization problem is ΔU. Therefore, the state-space representation has to be written in function of ΔU as well. To accomplish

this, the matrices need to be augmented as presented in [101] and [37]. The augmentation process is given in Equations (3.10)-(3.14).

tjeSubtracting $\vec{x}_k$ from $\vec{x}_{k+1}$ gives:

$$\begin{aligned} \vec{x}_{k+1} - \vec{x}_k &= A \cdot (\vec{x}_k - \vec{x}_{k-1}) + B \cdot (\vec{u}_k - \vec{u}_{k-1}) \\ \Leftrightarrow \Delta\vec{x}_{k+1} &= A \cdot \Delta\vec{x}_k + B \cdot \Delta\vec{u}_k \end{aligned} \tag{3.10}$$

The output equation needs to be altered to accommodate for the new state equation. To do this a new, augmented state variable $[\Delta\vec{x}_k^T \ \vec{y}_k^T]$ is created such that the output $\vec{y}_k$ is also represented in the state equation. The difference vector is given by:

$$\begin{aligned} \vec{y}_{k+1} - \vec{y}_k &= C \cdot (\vec{x}_{k+1} - \vec{x}_k) = C \cdot \Delta\vec{x}_{k+1} \\ \Leftrightarrow \vec{y}_{k+1} &= C \cdot \Delta\vec{x}_{k+1} + \vec{y}_k \end{aligned} \tag{3.11}$$

Using Equation (3.10):

$$\Leftrightarrow \vec{y}_{k+1} = C \cdot A \cdot \Delta\vec{x}_k + C \cdot B \cdot \Delta\vec{u}_k + \vec{y}_k \tag{3.12}$$

Combining results in a state-space in function of ΔU:

$$\begin{bmatrix} \Delta\vec{x}_{k+1} \\ \vec{y}_{k+1} \end{bmatrix} = \begin{bmatrix} A & 0_{n\times p} \\ C \cdot A & I_p \end{bmatrix} \cdot \begin{bmatrix} \Delta\vec{x}_k \\ \vec{y}_k \end{bmatrix} + \begin{bmatrix} B \\ C \cdot B \end{bmatrix} \cdot \Delta\vec{u}_k \tag{3.13}$$

$$\vec{y}_k = \begin{bmatrix} 0_{p\times n} & I_p \end{bmatrix} \cdot \begin{bmatrix} \Delta\vec{x}_k \\ \vec{y}_k \end{bmatrix} \tag{3.14}$$

Where $0_{n\times p}$ is a zero matrix of dimensions equal to $n \times p$ and I_p is the identity matrix of dimension p. The augmented state and input matrix are equal to Equation (3.15) and Equation (3.16) respectively.

$$\Delta\vec{x} = \begin{bmatrix} \Delta p_x & \Delta v_x & \Delta a_x & \Delta p_y & \Delta v_y & \Delta a_y & \Delta\theta & \Delta\dot{\theta} & \Delta\phi & \Delta\dot{\phi} & f_{s_x} & f_{s_y} & \omega_{s_y} & \omega_{s_x} \end{bmatrix}^T \tag{3.15}$$

$$\Delta\vec{u} = \begin{bmatrix} \Delta a_x & \Delta a_y & \Delta\dot{\theta} & \Delta\dot{\phi} \end{bmatrix}^T \tag{3.16}$$

$$\vec{y} = \begin{bmatrix} f_{s_x} & f_{s_y} & \omega_{s_y} & \omega_{s_x} \end{bmatrix}^T \tag{3.17}$$

The complete double integrator model is defined by Equations (3.13)-(3.17).

Vestibular model

As explained in Section 1.2, models exist that are able to transform inputs on acceleration and rotational rate to perceived specific force and perceived rotational rate. Incorporating these models explicitly in the MPC has the advantage that only those vestibular motions a human perceives are simulated. In Section 1.2 the transfer functions are described that model both the otolith as well as the semi-circular canal organ. However, these models are continuous time models, to incorporate them into MPC they need to be discretized. Using the MATLAB command *"c2d"* and *"tf2ss"*, it is possible to discretize both transfer functions using a zero-order hold with a sampling time equal to $100Hz$ (sampling time of simulation). The resulting state-space model for the otolith and semi-circular canal in one DoF is given in Equations (3.18)-(3.19) and Equations (3.20)-(3.21) respectively.

$$\vec{x}_{oth_{k+1}} = \begin{bmatrix} 1.533 & -0.5342 \\ 1 & 0 \end{bmatrix} \cdot \vec{x}_{oth_k} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot \vec{u}_k \tag{3.18}$$

$$\vec{y}_{oth_k} = \begin{bmatrix} 0.3716 & -0.3712 \end{bmatrix} \cdot \vec{x}_{oth_k} \tag{3.19}$$

$$\vec{x}_{scc_{k+1}} = \begin{bmatrix} 1.9981 & -0.9981 \\ 1.0000 & 0 \end{bmatrix} \cdot \vec{x}_{scc_k} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \cdot \vec{u}_k \tag{3.20}$$

$$\vec{y}_{scc_k} = \begin{bmatrix} -0.0018685 & 0.0018683 \end{bmatrix} \cdot \vec{x}_{scc_k} \tag{3.21}$$

Similarly to Equation (3.5), the model can be expanded to include the lateral DoF. The complete otolith model that distinguishes between longitudinal and lateral motion is given by:

$$\begin{aligned} A_O &= blkdiag(A_{oth}, A_{oth}) \\ B_O &= blkdiag(B_{oth}, B_{oth}) \\ C_O &= blkdiag(C_{oth}, C_{oth}) \end{aligned} \tag{3.22}$$

With matrix dimensions: $A_O \in \mathcal{R}^{n \times n}$, $B_O \in \mathcal{R}^{n \times m}$ and $C_O \in \mathcal{R}^{p \times n}$. With "$n$" equal to the number of states, "m" the number of inputs and "p" the number of outputs of the otolith model. The input to the system is $u_{oth} = [a_x, a_y]^T$, e.g. the chassis acceleration in x- and y-direction, and the perceived longitudinal forces as output $y_{oth} = [\vec{a}_x, \vec{a}_y]^T$. In similar fashion the complete state-space for the semi-circular canal can be found, resulting in A_S, B_S and C_S, with input $\vec{u}_{scc} = [\omega_y, \omega_x]^T$, denoting the rotational rate around y (pitch rate) and around x (roll rate). The output is described as $\vec{y}_{scc} = [\vec{\omega}_{s_y}\ \vec{\omega}_{s_x}]^T$, i.e. the perceived rotational rate around y and x.
One thing to note is that the transfer function given in Equation (1.1) does not incorporate the effect of tilt-coordination. Incorporating tilt-coordination can be done in the following way [6].

Consider a new, augmented state $\vec{x}_{o\bar{t}h} = [\vec{x}_{oth}\ \beta]^T$, with $\beta = [\theta\ \phi]^T$. Then a new matrix named B_{tilt} can be created according to Equation (3.23).

$$B_{tilt} = B_O \cdot \begin{bmatrix} g & 0 \\ 0 & -g \end{bmatrix} \tag{3.23}$$

Updating the existing state, input and output matrices results in their new, respective expressions shown in Equations (3.24)-(3.26).

$$A_{oth_{aug}} = \begin{bmatrix} A_O & B_{tilt} \\ 0_{q \times n} & I_q \end{bmatrix} \tag{3.24}$$

$$B_{oth_{aug}} = \begin{bmatrix} B_O & 0_{n \times q} \\ 0_{q \times q} & dt \cdot I_q \end{bmatrix} \tag{3.25}$$

$$C_{oth_{aug}} = \begin{bmatrix} C_O & 0_{q \times q} \end{bmatrix} \tag{3.26}$$

With *"q"* equal to the number of elements of β. As will be evaluated in Subsection 3.2.3, imposing constraints on system states becomes easier if they are explicitly defined in the output. To do this, the double integrator model from Equation (3.4), with the exception of the $C - matrix$, can be added to the vestibular model. Explicitly defining states in the output can be done by feeding them to output level by utilizing a identity matrix for C, i.e. I. Using Equation (3.5), the complete system matrices, $\sum_V = A_V, B_V, C_V, D_V$, are shown in Equations (3.27)-(3.29).

$$A_V = \begin{bmatrix} A_{oth_{aug}} & 0 & 0 \\ 0 & A_S & 0 \\ 0 & 0 & A_{x,y} \end{bmatrix} \tag{3.27}$$

$$B_V = \begin{bmatrix} B_{oth_{aug}} \\ 0 \quad B_S \\ B_{x,y} \end{bmatrix} \tag{3.28}$$

$$C_V = \begin{bmatrix} C_{oth_{aug}} & 0 & 0 \\ 0 & C_S & 0 \\ 0 & 0 & I \end{bmatrix} \tag{3.29}$$

Similarly to the integrator model, the vestibular system has to be defined in terms of ΔU. Analogous to Equation (3.13) and Equation (3.14) the system matrices can be augmented. The state, input and output vector for longitudinal and lateral DoF are equal to Equation (3.30), (3.31) and (3.32), respectively.

$$\Delta\vec{x} = \begin{bmatrix} \Delta x_{oth_{x,y}} & \Delta x_{scc_{y,x}} & \Delta\vec{x}_I & \vec{y} \end{bmatrix}^T \tag{3.30}$$

$$\Delta\vec{u} = \begin{bmatrix} \Delta a_x & \Delta a_y & \Delta\dot{\theta} & \Delta\dot{\phi} \end{bmatrix}^T \tag{3.31}$$

$$\vec{y} = \begin{bmatrix} f_{s_x} & f_{s_y} & \omega_{s_y} & \omega_{s_x} & \vec{x}_I \end{bmatrix}^T \tag{3.32}$$

With $\vec{x}_I$ equal to Equation (3.7). Equations (3.27)-(3.32) define the complete vestibular system integrated with the double integrator simulator model.

As mentioned earlier both these models are used to make predictions about future system states. It is possible to build a prediction model using recursion for which the derivation is given in Equations (3.33)-(3.37) [101][76].

Write:

$$\begin{aligned} \Delta\vec{x}_1 &= A \cdot \Delta\vec{x}_0 + B \cdot \Delta\vec{u}_0 \\ \Delta\vec{x}_2 &= A \cdot \Delta\vec{x}_1 + B \cdot \Delta\vec{u}_1 \\ \Leftrightarrow \Delta\vec{x}_2 &= A \cdot (A \cdot \Delta\vec{x}_0 + B \cdot \Delta\vec{u}_0) + B \cdot \Delta\vec{u}_1 \\ \Leftrightarrow \Delta\vec{x}_2 &= A^2 \cdot \Delta\vec{x}_0 + AB \cdot \Delta\vec{u}_0 + B \cdot \Delta\vec{u}_1 \end{aligned} \tag{3.33}$$

Working out for each $\bar{X} = [\vec{x}_1, \vec{x}_2, \cdots, \vec{x}_{N_p}]$ for control horizon N_c, and writing the result in matrix formulation gives the causal prediction model for "$\bar{X}$".

$$\Delta\bar{X} = \underbrace{\begin{bmatrix} A \\ A^2 \\ \vdots \\ A^{N_p} \end{bmatrix}}_{F_x} \cdot \Delta\vec{x}_0 + \underbrace{\begin{bmatrix} B & 0 & \cdots & 0 & 0 \\ AB & B & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ A^{N_p-1}B & A^{N_p-2}B & \cdots & \cdots & A^{N_p-N_c}B \end{bmatrix}}_{S_x} \cdot \Delta\vec{U} \tag{3.34}$$

$$\Delta\bar{X} = F_x \cdot \Delta\vec{x}_0 + S_x \cdot \Delta\vec{U} \tag{3.35}$$

Matrices F_x and S_x have dimension $F_x \in \mathcal{R}^{(N_p \cdot n) \times n}$ and $S_u \in \mathcal{R}^{(N_p \cdot n) \times (N_c \cdot m)}$. $\vec{x}_0$ is the current state of the system and $\Delta\vec{U}$ represents the vector of decision variables: $\Delta\vec{U} = [\Delta\vec{u}_0, \Delta\vec{u}_1, \cdots, \Delta\vec{u}_{N_c}]^T$. F_x denotes the evolution and S_x the prediction matrix. From $\vec{y_k} = C \cdot \vec{x}_k$, a prediction model for the output simply follows by multiplying $\bar{X}$ with "C".

$$\bar{Y} = \underbrace{\begin{bmatrix} CA \\ CA^2 \\ \vdots \\ CA^{N_p} \end{bmatrix}}_{F_y} \cdot \vec{x}_0 + \underbrace{\begin{bmatrix} CB & 0 & \cdots & 0 & 0 \\ CAB & CB & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ CA^{N_p-1}B & A^{N_p-2}B & \cdots & \cdots & CA^{N_p-N_c}B \end{bmatrix}}_{S_y} \cdot \Delta\vec{U} \tag{3.36}$$

$$\bar{Y} = F_y \cdot \vec{x}_0 + S_y \cdot \Delta\vec{U} \tag{3.37}$$

Matrices F_y and S_y have dimension $F_y \in \mathcal{R}^{(N_p \cdot p) \times n}$ and $S_y \in \mathcal{R}^{(N_p \cdot p) \times (N_c \cdot m)}$

3.2.2. Cost Function

The general cost function of the MPC involves two terms, the stage cost and the terminal cost. The latter one is required to ensure asymptotic stability [76]. The complete cost function is presented in Equation (3.38).

$$J(t) = \min \sum_{j=1}^{N_p} \|y(t+j) - r(t+j)\|_{Q_y}^2 + \sum_{j=0}^{N_c-1} (u(t+j))_{Qu}^2 + \sum_{j=0}^{N_c-1} (\Delta u(t+j))_{Qdu}^2 + X_{N_{Q_N}}^2 \tag{3.38}$$

$$s.t.: \begin{cases} \vec{x}_{k+1} = A \cdot \Delta\vec{x}_k + B \cdot \Delta\vec{u}_k \\ \vec{y}_k = C \cdot \Delta\vec{x}_k \\ -\vec{y}_{lim} \leq \vec{y} \leq \vec{y}_{lim} \\ -\vec{u}_{lim} \leq \vec{u} \leq \vec{u}_{lim} \\ -\Delta\vec{u}_{lim} \leq \Delta\vec{u} \leq \Delta\vec{u}_{lim} \end{cases} \tag{3.39}$$

With Q_y, Q_u, Q_{du}, and Q_N being weighting matrices.

- Q_y represents the weight of cost of the reference tracking error between the computed state output and the reference signal.
- Q_u denotes weight on the cost which tries to keep the total input as small as possible, larger inputs can stress the system and can be expensive, reference tracking while keeping them as small as possible is desirable.
- Q_{du} is the weight on cost of the incremental variation of the inputs. High deviations/oscillations in input result in large input derivatives which can lead to higher stresses in the system, keeping them as low as possible is desirable.
- Q_N is the weight on the terminal cost, which is a cost based on the state of the system. This cost is introduced to effectively impose stability on the MPC by making sure the state cannot runaway to very large values.

The first three terms represent the stage cost, the latter term the terminal cost. N_p and N_c denote the prediction and control horizon, with $N_c \leq N_p$. The values that correspond to the parameters in Equation (3.39) are mentioned in section 1.4, the simulator workspace limits. In order to use this cost function in the MPC framework, it needs to be rewritten to a generic Quadratic Programming (QP) problem that minimizes ΔU, Equation (3.40) shows the QP formulation, Equation (3.41) represents the constraint formulation [35].

$$J_{cost} = \operatorname{argmin} \frac{1}{2} \Delta U^T \cdot H \cdot \Delta U + h^T \cdot \Delta U \tag{3.40}$$

$$s.t.\ A_c \cdot \Delta U \leq b \tag{3.41}$$

The full derivation of the Hessian matrix H and matrix h is given in Equations (3.42)-(3.53).

Reference tracking error

Starting with the reference tracking error, using Equation (3.36) and Equation (3.37) the term $\sum_{j=1}^{N_p} \|y(t+j) - r(t+j)\|^2_{Q_y}$ in matrix notation can be written, with $R = [\vec{r}_1, \vec{r}_2, \cdots, \vec{r}_{N_p}]$ being the reference trajectory vector for N_p future steps, as follows:

$$\begin{aligned}
\sum_{j=1}^{N_p} \|y(t+j) - r(t+j)\|^2_{Q_y} &= (\bar{Y} - R)^T Q_y (\bar{Y} - R) \\
&= (F_y \vec{x}_0 + S_y \Delta\vec{U} - R)^T Q_y (F_y \vec{x}_0 + S_y \Delta\vec{U} - R) \\
&= \vec{x}_0^T F_y^T Q_y F_y \vec{x}_0 + \vec{x}_0^T F_y^T Q_y S_y \Delta\vec{U} - \vec{x}_0^T F_y^T Q_y R \\
&\quad + \Delta\vec{U}^T S_y^T Q_y F_y \vec{x}_0 + \Delta\vec{U}^T S_y^T Q_y S_y \Delta\vec{U} - \Delta\vec{U}^T S_y^T Q_y R \\
&\quad - R^T Q_y F_y \vec{x}_0 - R^T Q_y S_y \Delta\vec{U} + R^T Q_y R \\
&= \Delta\vec{U}^T S_y^T Q_y S_y \Delta\vec{U} + 2\vec{x}_0^T F_y^T Q_y S_y \Delta\vec{U} - 2R^T Q_y S_y \Delta\vec{U} \\
&= \Delta\vec{U}^T \cdot H_y \cdot \Delta\vec{U} + h_y^T \cdot \Delta\vec{U}
\end{aligned} \tag{3.42}$$

Which indeed resembles Equation (3.40) with:

$$\begin{cases} H_y &= S_y^T \cdot Q_y \cdot S_y \\ h_y &= 2 \cdot (\vec{x}_0^T \cdot F_y^T - R^T) \cdot Q_y \cdot S_y \end{cases} \tag{3.43}$$

Input & change in input

First the total input u needs to be written in function of $\Delta\vec{u}$ before $\sum_{j=0}^{N_c-1} (u(t+j))^2_{Qu}$ can be written in terms of H and h. Similarly to Equations (3.33)-(3.37), using the recursion formula the following expression can be found.

$$\begin{aligned}
\vec{u}_1 &= \vec{u}_0 + \Delta\vec{u}_1 \\
\vec{u}_2 &= \vec{u}_1 + \Delta\vec{u}_2 \\
\Leftrightarrow \vec{u}_2 &= \vec{u}_0 + \Delta\vec{u}_1 + \Delta\vec{u}_2
\end{aligned} \tag{3.44}$$

Where $\vec{u}_0$ denotes the total previous input. Writing in matrix formulation gives:

$$\bar{U} = \underbrace{\begin{bmatrix} I_m \\ I_m \\ \vdots \\ I_m \end{bmatrix}}_{F_u} \cdot \vec{u}_0 + \underbrace{\begin{bmatrix} I_m & 0 & \cdots & 0 & 0 \\ I_m & I_m & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & 0 \\ I_m & I_m & \cdots & \cdots & I_m \end{bmatrix}}_{S_u} \cdot \Delta\vec{U} \tag{3.45}$$

$$\bar{U} = F_u \cdot \vec{u}_0 + S_u \cdot \Delta\vec{U} \tag{3.46}$$

Matrices F_u and S_u have dimension $F_u \in \mathcal{R}^{(N_c \cdot m) \times m}$ and $S_u \in \mathcal{R}^{(N_c \cdot m) \times (N_c \cdot m)}$. $\Delta\vec{U} = [\Delta\vec{u}_0, \Delta\vec{u}_1, \cdots, \Delta\vec{u}_{N_c-1}]^T$, and I_m is an identity matrix of dimension m, equal to the number of inputs to the system. Knowing Equation (3.45), analogous to Equations (3.42)-(3.43), $\sum_{j=0}^{N_c-1} (u(t+j))^2_{Qu}$ can be written in terms of H and h, resulting in:

$$\begin{cases} H_u &= S_u^T \cdot Q_y \cdot S_u \\ h_u &= 2 \cdot \vec{u}_0^T \cdot F_u^T \cdot Q_u \cdot S_u \end{cases} \tag{3.47}$$

Writing the change in input term, $\sum_{j=0}^{N_c-1} (\Delta u(t+j))^2_{Qdu}$, straightforwardly follows from:

$$\sum_{j=0}^{N_c-1} (\Delta u(t+j))^2_{Qdu} = \Delta \vec{U}^T \cdot Q_{du} \cdot \Delta \vec{U} \tag{3.48}$$

$$\Leftrightarrow H_{du} = Q_{du} \tag{3.49}$$

Terminal cost

The last term that needs to be written in function of H and h is the cost on the terminal state. To be able to do this, the final state of the system at step N_p should be written in terms of $\Delta \vec{U}$. Realising that the terminal state is already represented in Equation (3.61) gives the following equality:

$$\vec{x}_{N_p} = A^{N_p} \cdot \vec{x}_0 + S_x(lr) \cdot \Delta \vec{U} \tag{3.50}$$

Where $S_x(lr)$ denotes the last row of the S_x matrix. Rewriting $X^2_{N_{Q_N}}$ gives:

$$\begin{aligned} X^2_{N_{Q_N}} &= X_N^T Q_N X_N^T \\ &= (A^{N_p} \vec{x}_0 + S_x(lr) \Delta \vec{U})^T Q_N A^{N_p} \vec{x}_0 + S_x(lr) \Delta \vec{U} \\ &= \Delta \vec{U}^T S_x(lr)^T Q_N S_x(lr) \Delta \vec{U} + \vec{x}_0^T A_{N_p}^T Q_N S_x(lr) \Delta \vec{U} \end{aligned} \tag{3.51}$$

Where:

$$\begin{cases} H_t &= S_x(lr)^T \cdot Q_N \cdot S_x(lr) \\ h_t &= \vec{x}_0^T \cdot A_{N_p}^T \cdot Q_N \cdot S_x(lr) \end{cases} \tag{3.52}$$

Combining Equation (3.43), (3.47), (3.49), and (3.52) defines the full Hessian matrix H and h matrix as found in Equation (3.40).

$$\begin{cases} H &= S_y^T \cdot Q_y \cdot S_y + S_u^T \cdot Q_y \cdot S_u + Q_{du} + S_x(lr)^T \cdot Q_N \cdot S_x(lr) \\ h &= 2 \cdot (\vec{x}_0^T \cdot F_y^T - R^T) \cdot Q_y \cdot S_y + 2 \cdot \vec{u}_0^T \cdot F_u^T \cdot Q_u \cdot S_u + \vec{x}_0^T \cdot A_{N_p}^T \cdot Q_N \cdot S_x(lr) \end{cases} \tag{3.53}$$

3.2.3. Constraints

The constraint equation that needs to be incorporated by the optimization is shown in Equation (3.41). From Equation (3.39) three different constraints can be recognized: constraints on the change in input, on the total input and on the state of the system. As can be seen in Equation (3.39) a lower as well as an upper limit needs to be enforced by the optimization process. However the notation, Equation (3.40), requires only a *less or equal sign*, e.g. an upper limit, to avoid possible issues the constraints can be written in the form described in [101].

$$\begin{cases} -\Delta \vec{U} &\leq -\Delta \vec{U}_{min} \\ \Delta \vec{U} &\leq \Delta \vec{U}_{max} \end{cases} \tag{3.54}$$

Constraints on total input

The constraint on total input means that for the next N_c steps, the total applied input cannot exceed the limit value. The F_u and S_u matrices, defined in Equation (3.45), can be used to this extent. These matrices describe the future N_c inputs in function of the decision variable $\Delta \vec{U}$. Taking into account Equation (3.54), the constraint equation for the total input is described by:

$$\begin{bmatrix} -S_u \\ S_u \end{bmatrix} \cdot \Delta\vec{U} \leq \begin{bmatrix} -\vec{U}_{min} + F_u \cdot \vec{u}_0 \\ \vec{U}_{max} - F_u \cdot \vec{u}_0 \end{bmatrix} \tag{3.55}$$

Where $\vec{u}_0$ denotes the previous total applied input.

Constraints on change in input

The input vector equals $\Delta\vec{u} = [\Delta a_x, \Delta a_y, \Delta\dot{\theta}, \Delta\dot{\phi}]^T$. Having constraints on their derivative implies that jerk in x- and y-direction and the rotational acceleration around x and y should be limited. Since the constraints are defined for platform motion and not for the actuators driving the platform, no values are defined that constrain platform jerk. Therefore only the constraints on the rotational acceleration are considered. Therefore two selectors are required, one for the pitch rate and one for the roll rate. Both selectors are given in Equation (3.56).

$$\begin{cases} S_{\dot{\theta}} &= \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \\ S_{\dot{\phi}} &= \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \end{cases} \tag{3.56}$$

Using the selectors, matrix M can be defined as follows.

$$M = \begin{bmatrix} S_{\dot{\theta}} & 0 & \cdots & \cdots & 0 \\ 0 & S_{\dot{\theta}} & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & S_{\dot{\theta}} \\ S_{\dot{\phi}} & 0 & \cdots & \cdots & 0 \\ 0 & S_{\dot{\phi}} & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & \cdots & S_{\dot{\phi}} \end{bmatrix} \tag{3.57}$$

With M having dimension $M \in \mathcal{R}^{(N_c \cdot \frac{m}{2}) \times N_c \cdot m}$. Knowing M the constraint equation for the incremental variation can be computed and is given in Equation (3.58).

$$\begin{bmatrix} -M \\ M \end{bmatrix} \cdot \Delta\vec{U} \leq \begin{bmatrix} -\Delta\vec{U}_{min} \\ \Delta\vec{U}_{max} \end{bmatrix} \tag{3.58}$$

Constraint on state

Contrary to the vestibular model, platform states are not explicitly defined in the output of the integrator model. The difference in constraint equation computation will be evaluated.
In case of the integrator model the future states need to be solely defined in terms of $\Delta\vec{U}$. This can be done by using the recursion formulation on equation Equation (3.59).

$$\bar{x}_{k+1} = \bar{x}_k + \vec{x}_{k+1} \tag{3.59}$$

Where $\bar{x}_k$ denotes the state at step k of the system and $\vec{x}_{k+1}$ denotes the augmented state variable $[\Delta\vec{x}_{k+1}, y_{k+1}]^T$. For which it is known that: $\Delta\vec{x}_{k+1} = A \cdot \Delta\vec{x}_k + B \cdot \Delta\vec{u}_k$. Substituting gives:

$$\begin{aligned} &\bar{x}_{k+1} = \bar{x}_k + A \cdot \Delta\vec{x}_k + B \cdot \Delta\vec{u}_k \\ \Leftrightarrow\ &\bar{x}_1 = \bar{x}_0 + A \cdot \Delta\vec{x}_0 + B \cdot \Delta\vec{u}_0 \qquad \text{for k=0} \\ &\bar{x}_2 = \bar{x}_1 + \Delta\vec{x}_2 \\ \Leftrightarrow\ &\bar{x}_2 = \bar{x}_0 + A \cdot \Delta\vec{x}_0 + B \cdot \Delta\vec{u}_0 + A \cdot \Delta\vec{x}_1 + B \cdot \Delta\vec{u}_1 \\ \Leftrightarrow\ &\bar{x}_2 = \bar{x}_0 + A \cdot \Delta\vec{x}_0 + B \cdot \Delta\vec{u}_0 + A \cdot (A \cdot \Delta\vec{x}_0 + B \cdot \Delta\vec{u}_0) + B \cdot \Delta\vec{u}_1 \\ \Leftrightarrow\ &\bar{x}_2 = \bar{x}_0 + (A + A^2) \cdot \Delta\vec{x}_0 + B \cdot \Delta\vec{u}_0 + A \cdot B \cdot \Delta\vec{u}_0 + B \cdot \Delta\vec{u}_1 \\ \Leftrightarrow\ &\bar{x}_2 = \bar{x}_0 + (A + A^2) \cdot \Delta\vec{x}_0 + (A \cdot B + B) \cdot \Delta\vec{u}_0 + B \cdot \Delta\vec{u}_1 \end{aligned} \tag{3.60}$$

Working out until N_p gives the following matrix:

$$\bar{X}_{tot} = \underbrace{\begin{bmatrix} I_n \\ I_n \\ \vdots \\ I_n \end{bmatrix}}_{I_c} \bar{x}_0 + \underbrace{\begin{bmatrix} A \\ A + A^2 \\ \vdots \\ A + \cdots + A^{N_p} \end{bmatrix}}_{F_c} \Delta\vec{x}_0 + \underbrace{\begin{bmatrix} B & 0 & \cdots & 0 \\ AB + B & B & \cdots & 0 \\ \vdots & \ddots & \ddots & 0 \\ A^{N_p-1}B + \cdots + B & \cdots & \cdots & A^{N_p-N_c}B + \cdots + B \end{bmatrix}}_{S_c} \Delta\vec{U} \tag{3.61}$$

$$\bar{X}_{tot} = I_c \cdot \bar{x}_0 + F_c \cdot \Delta\vec{x}_0 + S_c \cdot \Delta\vec{U} \tag{3.62}$$

It should be noted that in this equation only the states are considered on which constraints are imposed, those states are $[x, v_x, y, v_y, \theta, \phi]$. A selector matrix K_c is required that satisfies this condition.

$$K_c = I_{N_p} \otimes \begin{bmatrix} 1 & 0 & 0_{1\times11} \\ 0 & 1 & 0_{1\times12} \\ 0_{1\times3} & 1 & 0_{1\times10} \\ 0_{1\times4} & 1 & 0_{1\times9} \\ 0_{1\times6} & 1 & 0_{1\times7} \\ 0_{1\times8} & 1 & 0_{1\times5} \end{bmatrix} \tag{3.63}$$

Where "$\otimes$" is the Kronecker product and $0_{1\times t}$ a row matrix with "t" elements of zero. This results in the following constraint equation for implicit state definition.

$$\begin{bmatrix} -(K_c \cdot S_c) \\ K_c \cdot S_c \end{bmatrix} \Delta\vec{U} \leq \begin{bmatrix} -\bar{X}_{min} + K_c \cdot I_c \cdot \bar{x}_0 + K_c \cdot F_c \cdot \Delta\vec{x}_0 \\ \bar{X}_{max} - K_c \cdot I_c \cdot \bar{x}_0 - K_c \cdot F_c \cdot \Delta\vec{x}_0 \end{bmatrix} \tag{3.64}$$

The total constraint matrix for the optimization problem for the MPC using the double integrator model equals:

$$\underbrace{\begin{bmatrix} -S_u \\ S_u \\ -M \\ M \\ -(K_c \cdot S_c) \\ K_c \cdot S_c \end{bmatrix}}_{A_{constraint}} \Delta\vec{U} \leq \underbrace{\begin{bmatrix} -\vec{U}_{min} + F_u \cdot \vec{u}_0 \\ \vec{U}_{max} - F_u \cdot \vec{u}_0 \\ -\Delta\vec{U}_{min} \\ \Delta\vec{U}_{max} \\ -\bar{X}_{min} + K_c \cdot I_c \cdot \bar{x}_0 + K_c \cdot F_c \cdot \Delta\vec{x}_0 \\ \bar{X}_{max} - K_c \cdot I_c \cdot \bar{x}_0 - K_c \cdot F_c \cdot \Delta\vec{x}_0 \end{bmatrix}}_{b_{constraint}} \tag{3.65}$$

As one can see, implicitly defining state constraints can become convoluted. The difference in imposing constraints on an explicit system is significant. For the vestibular model a selector matrix K_c is also required. Given the imposed constraints on output $[x, v_x, y, v_y, \theta, \phi]$, K_c looks like:

$$K_c = I_{N_p} \otimes \begin{bmatrix} 0_{1\times4} & 1 & 0_{1\times9} \\ 0_{1\times5} & 1 & 0_{1\times8} \\ 0_{1\times7} & 1 & 0_{1\times6} \\ 0_{1\times8} & 1 & 0_{1\times5} \\ 0_{1\times10} & 1 & 0_{1\times3} \\ 0_{1\times12} & 1 & 0_{1\times1} \end{bmatrix} \tag{3.66}$$

Using Equation (3.37):

$$Y_{min} \leq K_c \cdot \bar{Y} \leq Y_{max} \tag{3.67}$$

$$\Leftrightarrow Y_{min} \leq K_c \cdot (F_y \cdot \vec{x}_0 + S_y \cdot \Delta\vec{U}) \leq Y_{max} \tag{3.68}$$

$$\Leftrightarrow \begin{cases} -K_c \cdot S_y \cdot \Delta\vec{U} \leq -Y_{min} + K_c \cdot F_y \cdot \vec{x}_0 \\ K_c \cdot S_y \cdot \Delta\vec{U} \leq Y_{max} - K_c \cdot F_y \cdot \vec{x}_0 \end{cases} \tag{3.69}$$

Which leads to the constraint matrix for explicit defined states:

$$\begin{bmatrix} -(K_c \cdot S_y) \\ K_c \cdot S_y \end{bmatrix} \Delta\vec{U} \leq \begin{bmatrix} -Y_{min} + K_c \cdot F_y \cdot \vec{x}_0 \\ Y_{max} - K_c \cdot F_y \cdot \vec{x}_0 \end{bmatrix} \tag{3.70}$$

The total constraint matrix for the optimization problem for the MPC using the vestibular model equals:

$$\underbrace{\begin{bmatrix} -S_u \\ S_u \\ -M \\ M \\ -(K_c \cdot S_y) \\ K_c \cdot S_y \end{bmatrix}}_{A_{constraint}} \Delta\vec{U} \leq \underbrace{\begin{bmatrix} -\vec{U}_{min} + F_u \cdot \vec{u}_0 \\ \vec{U}_{max} - F_u \cdot \vec{u}_0 \\ -\Delta\vec{U}_{min} \\ \Delta\vec{U}_{max} \\ -Y_{min} + K_c \cdot F_y \cdot \vec{x}_0 \\ Y_{max} - K_c \cdot F_y \cdot \vec{x}_0 \end{bmatrix}}_{b_{constraint}} \tag{3.71}$$

3.2.4. Slack variables

Slack variables are introduced for output/state constraints to make them ”soft”. The reason why this is necessary is because if the output/state constraints are enforced, this can lead to high amplitude and high frequency changes in both control and incremental variation in control input. This could lead to the violation of input constraints which can lead to constraint conflicts [101]. Appending the slack variable vector $\vec{y}_s$ with the decision variable vector, $\Delta\vec{U}$, alters the constraint equation such that:

$$A \cdot \Delta\vec{U} - y_s \leq b \tag{3.72}$$

Where $\vec{y}_s$ should only affect the output/state constraints. The specific matrix that only affects constraints on the maximum values of $[x, v_x, y, v_y, \theta, \phi]$ looks as follows:

$$A_{Sv} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{3.73}$$

The matrix that affects the minimum values is simply $-A_{Sv}$. The full appendable matrix A_s is defined by:

$$A_s = \begin{bmatrix} 0_{size(-S_u;Su)} \\ 0_{size(-M;M)} \\ -A_{Sv} \\ A_{Sv} \end{bmatrix} \tag{3.74}$$

$$A_{new} = \begin{bmatrix} A_{constraint} & A_s \end{bmatrix} \tag{3.75}$$

Where S_U and M denote the matrices from Equation (3.65), and A_{new} is the new complete constraint matrix. The input vector equals $\Delta\vec{U}_{aug} = [\Delta\vec{U}\ \vec{y}_s]^T$.
Applying no cost to using slack variables would not make sense as this would make the constraints

completely redundant. Therefore a cost should be added to using slack variables to satisfy the output/state constraints. This can be easily done by adding a cost matrix to the quadratic cost function from Equation (3.40).

$$J_{cost} = \text{argmin}\, \frac{1}{2}\Delta\vec{U}_{aug}^{T} \cdot H_N \cdot \Delta\vec{U}_{aug} + h^T \cdot \Delta\vec{U}_{aug} \tag{3.76}$$

$$H_N = H + R_{slack} \tag{3.77}$$

Where R_{slack} denotes the weight on the cost of using the slack variables in y_s. Costs on using slack variables for the displacement or angle do not necessarily have to be equal to each other and should be tuned accordingly. The new Hessian, H and h matrix can simply be constructed as follows:

$$\begin{cases} H_N &= blkdiag(H, R_{slack}) \\ h &= \begin{bmatrix} h & 0_{1\times(N_p \cdot dim(Sv))} \end{bmatrix} \end{cases} \tag{3.78}$$

Where $dim(Sv)$ equals the amount of slack variables.

3.2.5. Quadratic Solver

The last step of the MPC is to find the set of inputs that minimize the cost function presented in Equation (3.40) while also satisfying the constraints imposed on the system. Many different solvers exist that are able to solve a constrained quadratic programming (QP) problem. These can be categorized in two main domains: explicit or off-line methods and online/real-time methods. In the former method the solution to the QP problem is explicitly computed in function of the initial state, the control action can be applied in real-time by use of a resulting look-up table [**?**] [30]. The problem with this method is that the look-up table can grow exponentially with increasing prediction/control horizon and state and input dimensions [69][102]. Due to the inherent time constraints imposed on the MPC MCA problem [22], in the scope of this research only the latter, online algorithms are considered. The online methods can be subdivided into

- Interior point (IP): A technique that utilizes the convex nature of the quadratic problem. IP has polynomial complexity [74]. Although IP is efficient in solving large optimization problems, due to the lack of *hot-start strategies* [104], e.g. using the solution of the previous calculated optimal control action to reduce computational load of the new optimal control action, its application is limited to processes with lower real-time requirements. Some studies were performed to try and come-up with hot start strategies which were successful, however no generic solution has yet been found [102][88].
- Active Set (AS): The active set method is inspired by the explicit method where it expects that the active set does not change much from one QP to the next. In this critical region, the QP solution depends affinely on the current state of the system [34]. This means that the active set method does utilize a form of hot-start strategy. Unlike the IP method, the AS method cannot guarantee polynomial complexity, in the worst case exponential in the size of the problem.

Several solvers exist that utilize one of the two described methods. In MATLAB the ready to use function "*quadprog*" uses the interior point method [96]. Another solver that is used often for MPC in driving simulation [7], is named QPOases which uses the active set method [33]. For simplicity sake, *quadprog* is used to solve the QP problem. If performance becomes more stringent, more time will be invested into this matter.

3.3. Performance analysis

In the previous section the complete algorithm and the defining mathematics have been fully explained. In this section simulation results for both models will be presented and analysed. First it will be explained how the parametric weight values were tuned and their final values will be presented. Secondly, the effect of introducing slack variables will be explored. An analysis on the effect of changing the prediction and control horizon will be elaborated thereafter. After these investigations, the simulation parameters

are all set and simulation results with real vehicle data can be analysed and compared to the results found in section 2.2.

3.3.1. Weight values

Four weight matrices are present in the cost function of Equation (3.40). The reference error tracking weight, the weight on total input and its incremental variation, the weight on the terminal state and the weight on using the slack variables. If tuned independently for the 4DoF simulation this implies tuning *30* variables. The weighting values can be tuned by trial-and-error with a more simple, 2DoF, double integrator model. It is assumed that the dynamics of each DoF are independent [22], as such the weights can be tuned by only considering motion in that specific DoF. In order to tune the weights on the acceleration input a step reference trajectory on specific force is considered, the weights for the other DoF are set to zero. The weights belonging to the rotational motion where tuned in similar manner, but following a sinusoidal reference trajectory, in both cases the constraints are disabled. Plotting values of the Hessian matrix H and h gives more insight on the weighting parameter influence on MPC performance and what gets priority. Based on this information, more objective decisions about changing certain values can be made while it also can be used to understand why the MPC behaves in certain ways. One thing to note is that a decision needs to be made on what part of the matrices need to be shown. The Hessian matrix is a symmetric matrix, of size $[N_c \cdot \text{no deg inputs}]$, when working out the multiplication in Equation (3.79) the entries on the diagonal represent the auto-correlated weights: $[W_{1,1}, W_{2,2}, \cdots]$, the off-diagonal represent the inter-correlated input weights: $[W_{1,2}, W_{2,1}, \cdots]$.

$$\begin{bmatrix} \Delta u_{a,1} \\ \Delta u_{\omega,1} \\ \Delta u_{a,2} \\ \Delta u_{\omega,2} \\ \vdots \\ \Delta u_{a,N_c} \\ \Delta u_{\omega,N_c} \end{bmatrix}^T \cdot \begin{bmatrix} W_{a_{1,1}} & W_{\omega_{1,2}} & W_{a_{1,3}} & W_{\omega_{1,4}} & \cdots & W_{a_{1,N_c-1}} & W_{\omega_{1,N_c}} \\ W_{a_{2,1}} & W_{\omega_{2,2}} & W_{a_{2,3}} & W_{\omega_{2,4}} & \cdots & W_{a_{2,N_c-1}} & W_{\omega_{2,N_c}} \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ W_{a_{N_c,1}} & W_{\omega_{N_c,2}} & W_{a_{1,3}} & W_{\omega_{N_c,4}} & \cdots & W_{a_{N_c,N_c-1}} & W_{\omega_{N_c,N_c}} \end{bmatrix} \cdot \begin{bmatrix} \Delta u_{a,1} \\ \Delta u_{\omega,1} \\ \Delta u_{a,2} \\ \Delta u_{\omega,2} \\ \vdots \\ \Delta u_{a,N_c} \\ \Delta u_{\omega,N_c} \end{bmatrix} \tag{3.79}$$

$$= W_{a_{1,1}} \cdot \Delta u_{a,1}^2 + W_{\omega_{1,2}} \cdot \Delta u_{a,1} \Delta u_{\omega,1} + W_{a_{1,3}} \cdot \Delta u_{a,1} \Delta u_{a,2} + \cdots + W_{\omega_{N_c,N_c}} \cdot \Delta u_{\omega,N_c}^2 \tag{3.80}$$

Since the interest is in the ratio in weight, the parameter which defines importance/priority, the assumption is made that only the weights on the input at the next instance are investigated, e.g. $W_{a_{1,1}}$ and $W_{\omega_{2,2}}$.
An example can be given for the tuning of the weighting matrix Q_{du}. If the value of Q_{du} is set to a value of 1, the behaviour from fig. 3.2 is found.

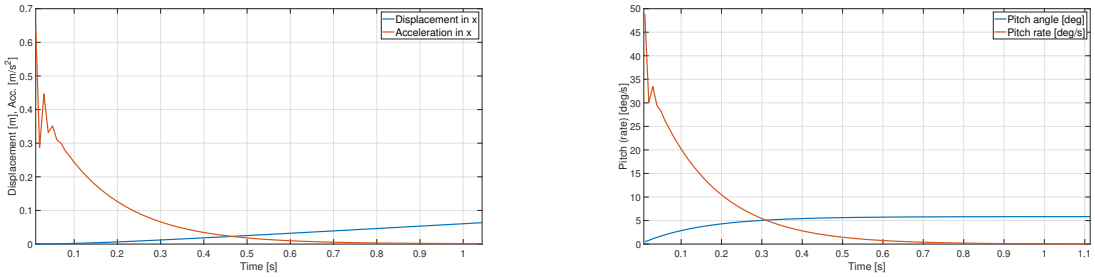


Figure 3.2: **Left**: displacement & acceleration in x-direction for unity step acceleration input with $Q_{du} = 1$. **Right**: pitch angle & pitch rate for unity step acceleration input with $Q_{du} = 1$.

Only the first second of the simulation is shown, as one can clearly see, oscillatory inputs are present. These are not desirable as they have a direct impact on the perceived specific force. Figure 3.3 shows the weight values corrected for the applied input $\Delta\vec{U}$. From this figure it can be deducted that indeed the weight Q_{du} on a_x and ω_y is low compared to the other values, reducing other parameters is therefore more efficient for the optimizer in order to find a minimum.

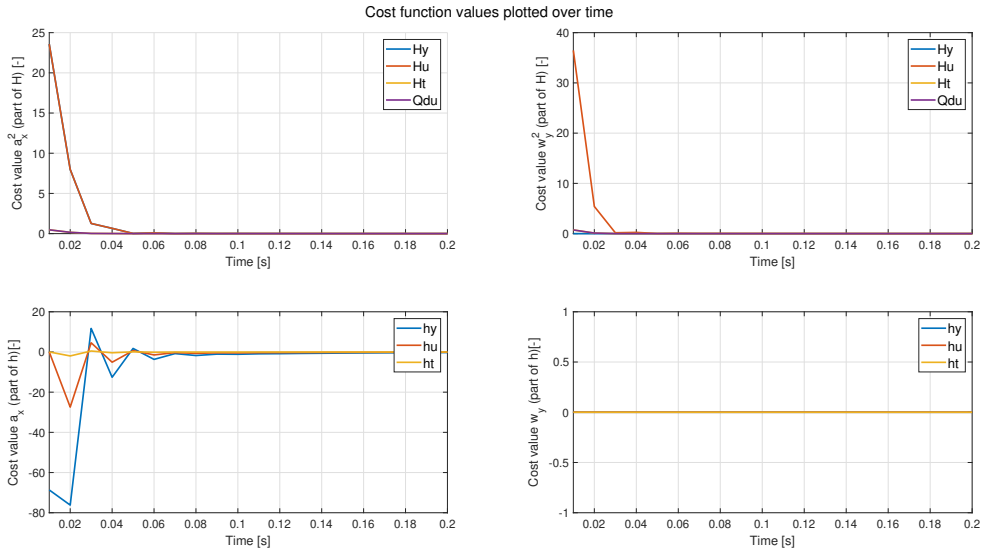


Figure 3.3: MPC cost parameter values over time for longitudinal acceleration unity step input, $Q_{du} = 1$.

Increasing Q_{du} by a factor of 20 should significantly improve the performance and smooth everything out. The result is shown in fig. 3.4 with the total specific force shown in fig. 3.5.

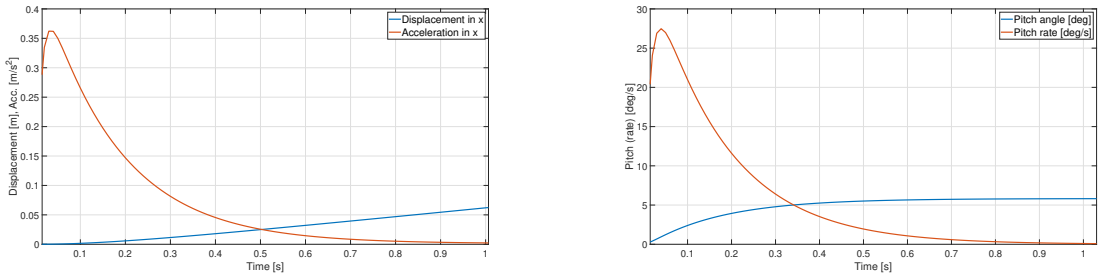


Figure 3.4: **Left**: displacement & acceleration in x-direction for unity step acceleration input with $Q_{du} = 20$. **Right**: pitch angle & pitch rate for unity step acceleration input with $Q_{du} = 20$.

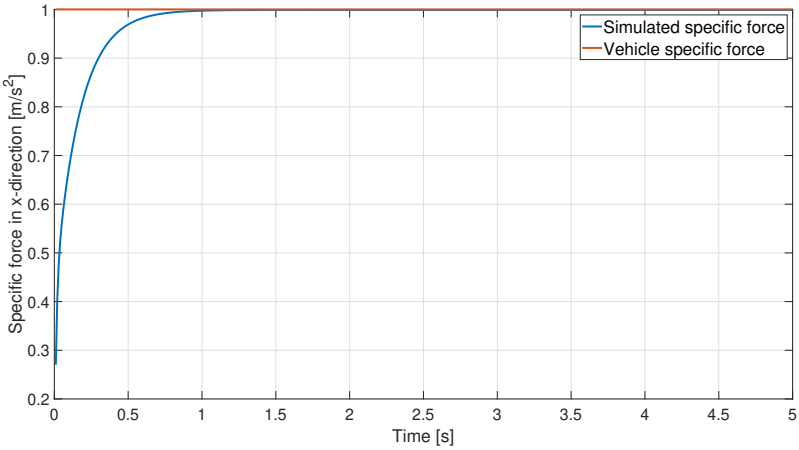


Figure 3.5: Reference vs simulated specific forces in x-direction for unity step acceleration input.

With the new cost values in the simulation equal to fig. 3.6. It can be noted that by increasing the

weight of Q_{du} the cost of total input U also decreases. The reason is simple: lower high amplitude variations in input lead to lower total inputs, and therefore decreases cost.

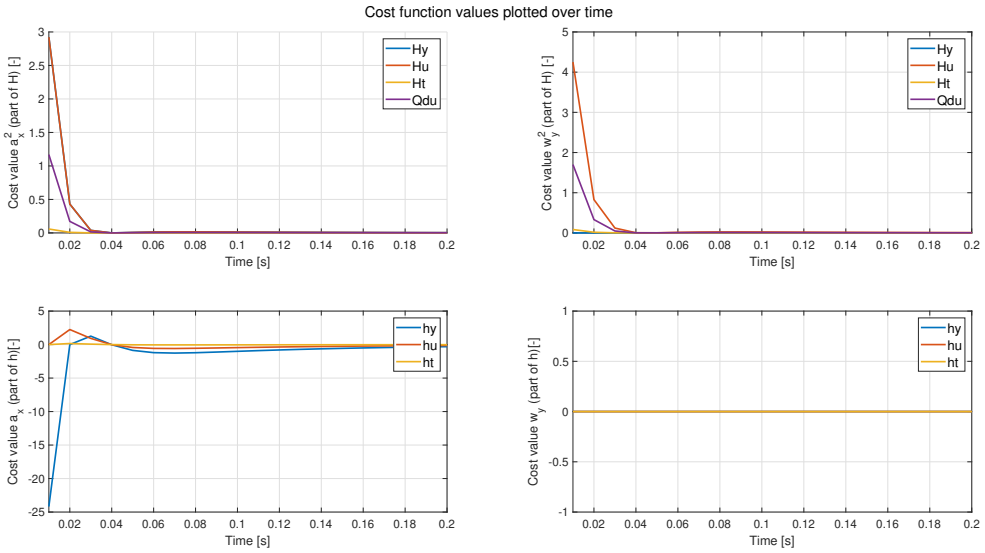


Figure 3.6: MPC cost parameter values over time for longitudinal acceleration unity step input, $Q_{du} = 20$.

In similar fashion the rest of the weighting parameter values was found. The full set of parameters for the 4DoF system is given in Table 3.1. It should be noted that in further research more effort into tuning of these parameters needs to happen. For example Katliar established the error weighting values, Q_y, by an approximation of the variance of the specific force and rotational rates [50].

Table 3.1: MPC MCA weighting parameter values.

Parameter	Value
$Q_y(a_x, a_y, \omega_y, \omega_x)$	*[1, 1, 1, 1]*
$Q_{du}(a_x, a_y, \omega_y, \omega_x)$	*[20, 20, 20, 20]*
$Q_u(a_x, a_y, \omega_y, \omega_x)$	*[1, 1, 1, 1]*
Q_t	*[10]*
$Q_{slack}(x, y, \theta, \phi)$	*[10^4, 10^4, 10^2, 10^2]*

3.3.2. Slack variables

As explained earlier, the slack variables have the function of realising low frequency input signals near the constraint boundary. At a cost a slack variable can be used to exceed an otherwise ”hard” constraint. Since constraints are not enforced, it is important to implement a margin on the constraint. After some simulations, it was found that with the cost values in table 3.1 a margin of ≈10% is sufficient, e.g. set max x-displacement to $0.25m$ instead of $0.28m$. In order to investigate the effect of including a slack variable, the original situation is elaborated upon: a step input on specific force with a hard output constraint on x-displacement of $0.28m$. The resulting specific force can be seen in fig. 3.7, the displacement in fig. 3.8.

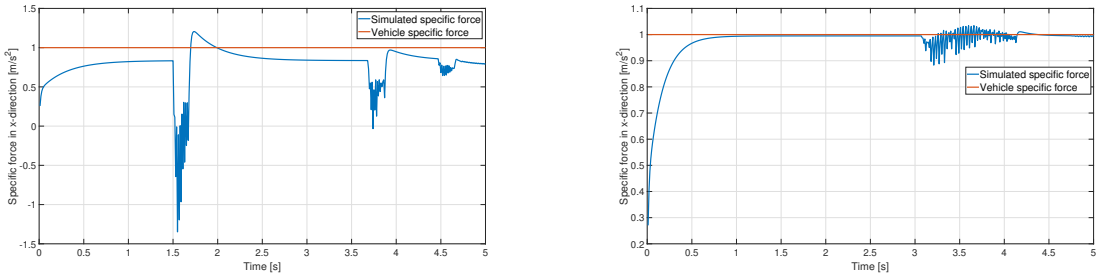


Figure 3.7: **Left**: Specific force in x-direction for unity step input on acceleration with $N_p = 10$ without slack variables. **Right**: Specific force in x-direction for unity step input on acceleration with $N_p = 50$ without slack variables.

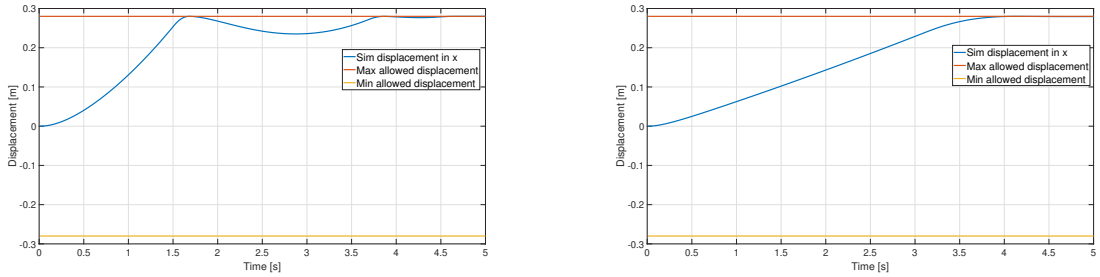


Figure 3.8: **Left**: Displacement in x-direction for unity step input on acceleration with $N_p = 10$ without slack variables. **Right**: Displacement in x-direction for unity step input on acceleration with $N_p = 50$ without slack variables.

The high amplitude/frequency input change near the displacement constraint can clearly be seen. With a larger prediction horizon the effect is still present, albeit less pronounced. This makes sense since the optimization program can anticipate further into the future, even though it is less bad this behaviour is still not desired. When adding the slack variables the resulting specific force is shown in fig. 3.9 and the displacement shown in fig. 3.10

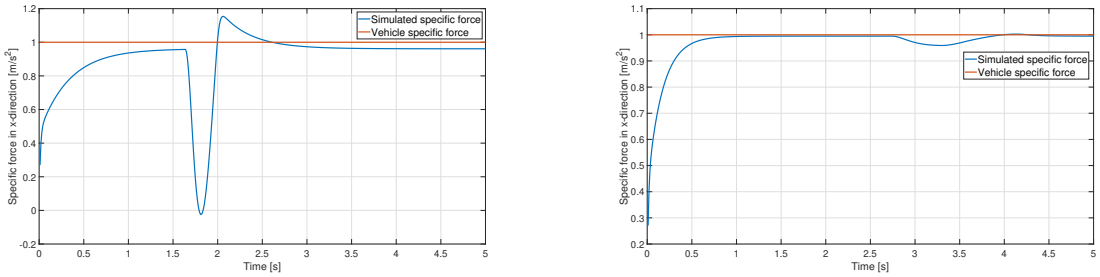


Figure 3.9: **Left**: Specific force in x-direction for unity step input on acceleration with $N_p = 10$ with slack variables. **Right**: Specific force in x-direction for unity step input on acceleration with $N_p = 50$ with slack variables.

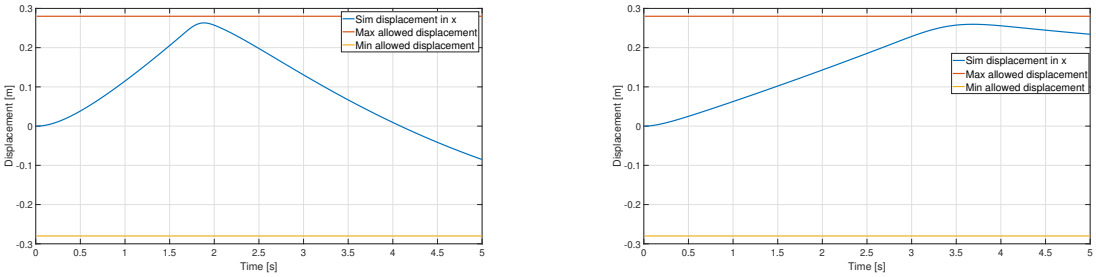


Figure 3.10: **Left**: Displacement in x-direction for unity step input on acceleration with $N_p = 10$ with slack variables. **Right**: Displacement in x-direction for unity step input on acceleration with $N_p = 50$ with slack variables.

The first thing that should be noted is the input signal which becomes smooth, no high frequent signal are present near the boundary of the constraint. Another thing that becomes apparent is the effect of a small prediction horizon in combination with adding slack variables. When the prediction horizon is too small, the simulated specific force shows high frequent transient behavior due to reaching the workspace limits. This can be seen in the left part of fig. 3.9 and fig. 3.10, between 1.5s and 2s. This characteristic disappears when increasing the prediction horizon N_p from 10 to 50.

3.3.3. Prediction and control horizon

Altering the prediction and control horizon has a great effect on the performance of the MPC. Increasing both horizons, makes the MPC more accurate and due to the extended prediction the constraints can be met without applying large inputs near the constraint boundary. On a more practical note looking at motion cueing, a large enough prediction horizon reduces the need to add washout to the algorithm. The algorithm can anticipate motion and use its motion workspace optimally. However, due to computational limits the prediction/control horizon cannot increase indefinitely. This means in real-time applications a trade-off is necessary between accuracy and the maximum frequency at which the simulation can be run. For this research, this trade-off is not necessary since the MPC runs off-line.
In this study the prediction and control horizon are equal to each other, based on the results in fig. 3.10, it was decided to perform simulations with a prediction horizon equal to 50 steps, for full test drive MPC calculations this prediction horizon safeguards reasonable computational time while also giving solid results.

3.3.4. Simulation results real vehicle data

Using a prediction horizon and control horizon of 50 steps and simulation parameters found in table 3.1, simulations using the reference drive found in fig. A.1 and fig. A.2 can be performed. The performance of both models will be investigated. For both models two different prediction methods will be used for comparison, both are found in [22] and [26]. One will be a constant prediction where the current vehicle state is used to create a N_p long constant reference trajectory for all 4 DoF. The other will be an oracle prediction, where the next N_p prerecorded samples in all 4 DoF will be used as reference trajectory. Next to an MPC side-by-side comparison using different prediction strategies, the oracle MPC will be compared to the CWA for the situation presented in fig. 2.17.

Double integrator model

The simulated specific force in x- and y-direction with respect to the vehicle specific force for a 50s simulation using *oracle logic* is shown in fig. 3.11. The perceived rotational rate with respect to the vehicle rotational rate is shown in fig. 3.12. Figure 3.13 shows that all workspace limits are satisfied. The results for a 500s simulation can be found in fig. A.7, fig. A.8 and fig. A.9.

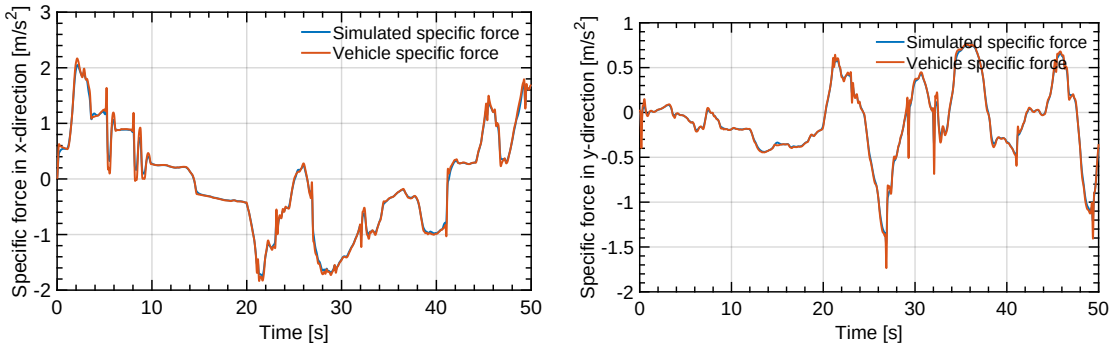


Figure 3.11: **Left**: 50 second snapshot of simulated and reference specific force in x-direction using MPC with double integrator model. **Right**: 50 second snapshot of simulated and reference specific force in y-direction using MPC with double integrator model.

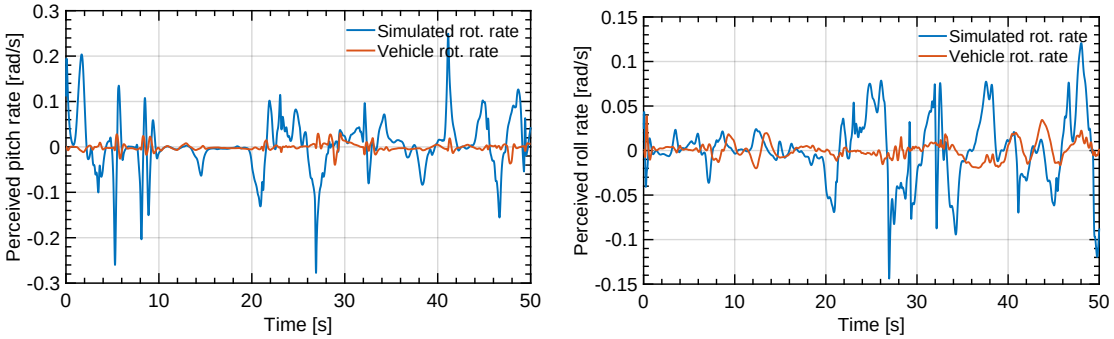
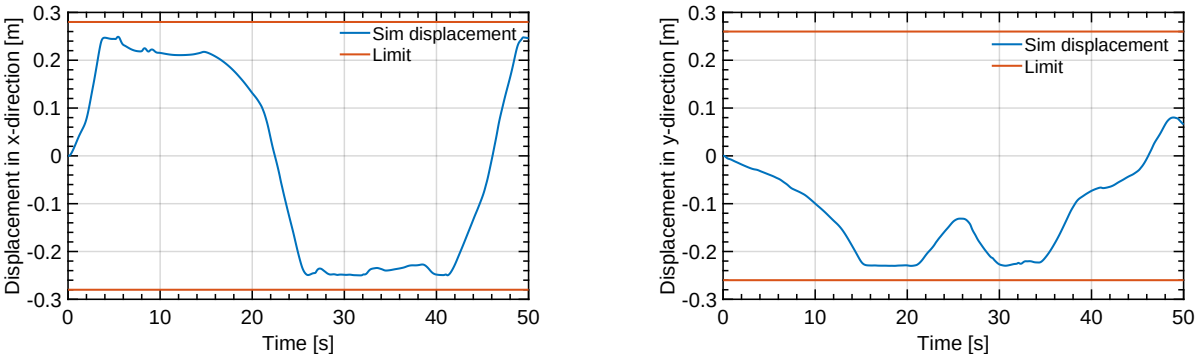


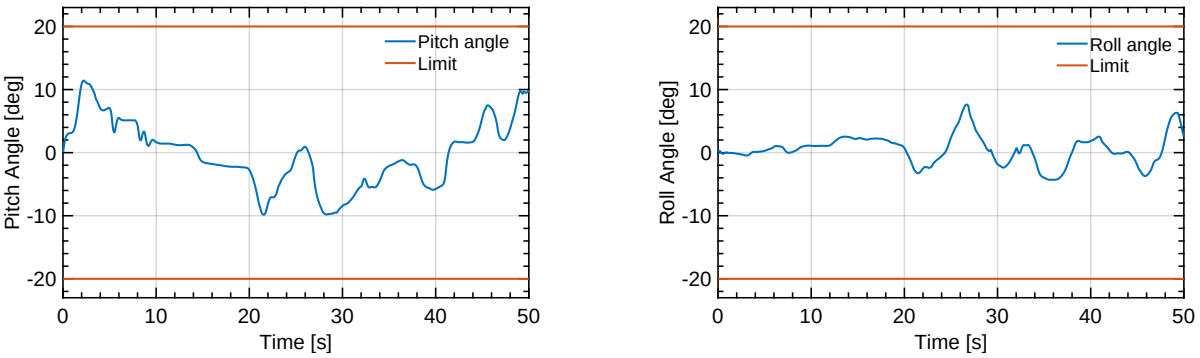
Figure 3.12: **Left**: 50 second snapshot of simulated and reference pitch rate using MPC with double integrator model. **Right**: 50 second snapshot of simulated and reference roll rate using MPC with double integrator model.

From these figures the benefit of using the MPC paradigm with a perfect reference trajectory becomes clear. Looking at the specific force graphs less false cues or missing cues than CWA are present, resulting in higher motion cueing fidelity in longitudinal and lateral direction. However, just as with the CWA, due to a limited workspace in longitudinal/lateral direction, the majority of specific force is simulated by tilt coordination. This results in a lot of missing and false cues, especially since the perceptional threshold for rotation is not considered. From simulation it is difficult to say how severe this rotational incongruency affects the overall motion cueing fidelity. From experiments done by [71], it is clear that the best motion cueing ratings are given if a compromise is made between accurate longitudinal/lateral simulation of specific forces and limiting false rotational rates. This can be done by pre-tuning the weights using a motion cueing predictor [71], and then fine tuning them in simulator experiments.



(a) Displacement in X-direction.

(b) Displacement in Y-direction.



(c) Pitch angle displacement.

(d) Roll angle displacement

Figure 3.13: 50 second snapshot of simulator workspace displacement in both x- and y-direction as well as pitch and roll using MPC with double integrator model.

It is expected that the performance deteriorates when the prediction quality for the reference declines. When using a constant reference trajectory the simulated specific force, both in longitudinal and lateral direction shows more missing and false cues due to lag, this can be seen in fig. 3.14. The difference is, however, less pronounced than presented by [26]. This could have several reasons, to make the real-time implementation possible, usually the control horizon is limited (50) and not equal to the prediction horizon $(100-150)$. Larger control horizons limit the applied input in future steps as the control energy is distributed over more control steps [101], this could result in improved performance when featuring a bad reference. Because strict real-time deadlines are required and cannot be guaranteed, in case the algorithm did not solve the optimization process, the previous input is applied to the system. One thing to also note is that in this situation the control is ”perfect”, e.g. it controls the same model that it also uses to predict evolution of future states. In a real-life simulation large incongruencies may exist between the model and the actual system.

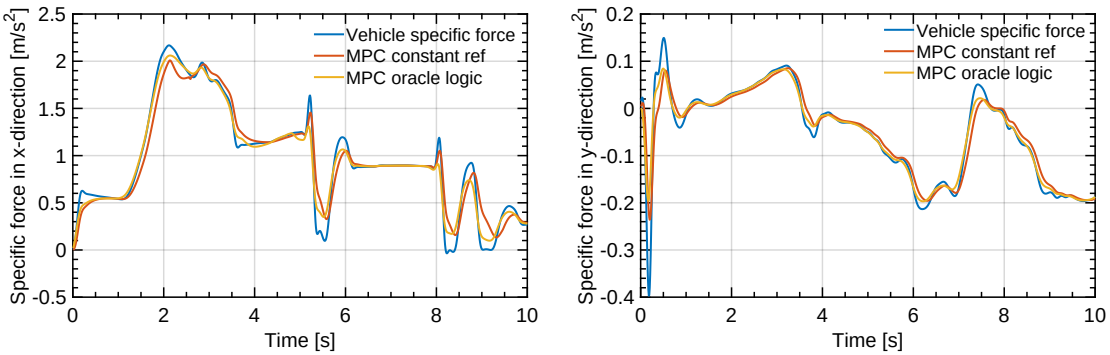


Figure 3.14: **Left**: 10 second snapshot of specific force in x-direction using MPC with double integrator model. **Right**: 10 second snapshot of specific force in y-direction using MPC with double integrator model.

Vestibular model

The results for the MPC for a 50s simulation using *oracle logic* can be found in fig. 3.15 and fig. 3.16 regarding the specific force and rotational rate respectively. The same logic as with the double integrator model can be followed. Because a perfect reference is available, utilizing the advantages of MPC: a prediction model and explicit constraints, the simulation of longitudinal and lateral specific forces presents less false and missing cues while remaining within workspace limits, see fig. 3.17. Due to the inclusion of the vestibular model, the simulated specific force are a scaled version of the reference signal. Comparing the simulated rotational rates with the vehicle rotational rates, false and missing cues are abundant. As well as with the double integrator model, the weights can be tuned to increase the priority of following rotational rates. The results for a 500s simulation can be found in fig. A.10, fig. A.11 and fig. A.12.

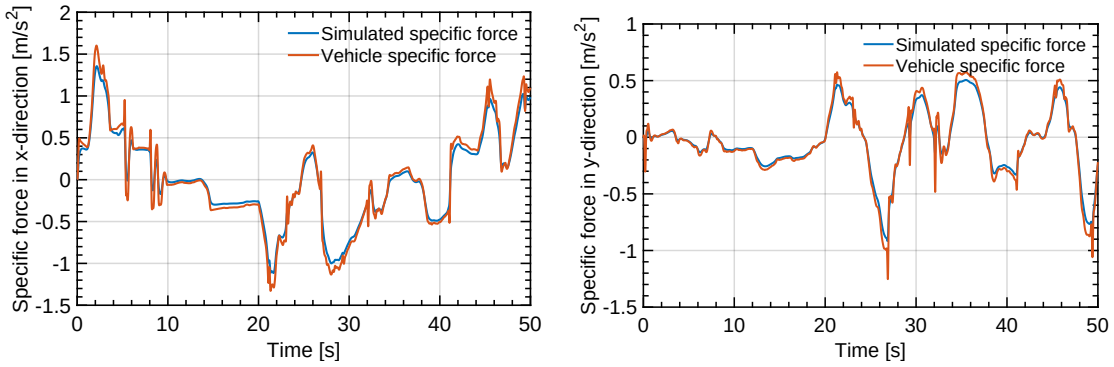


Figure 3.15: **Left**: 50 second snapshot of specific force in x-direction using MPC with vestibular model. **Right**: 50 second snapshot of specific force in y-direction using MPC with vestibular model.

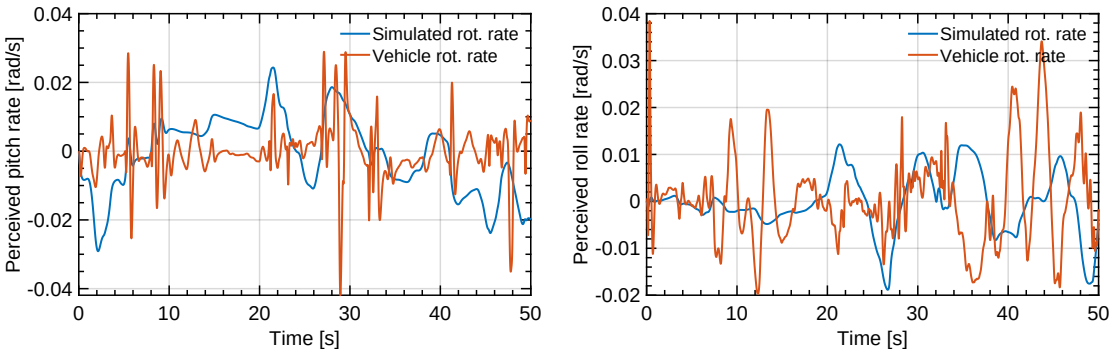
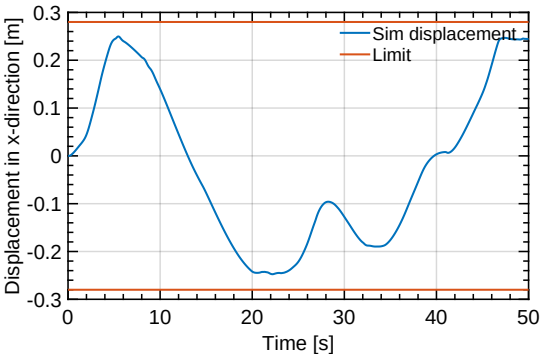
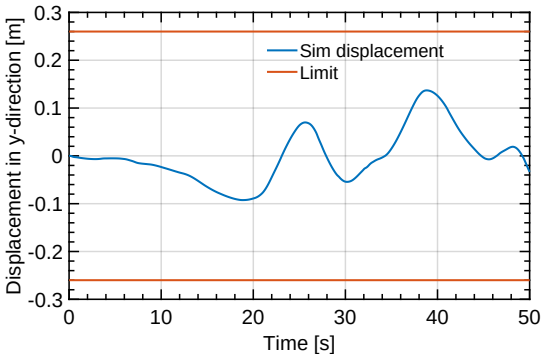


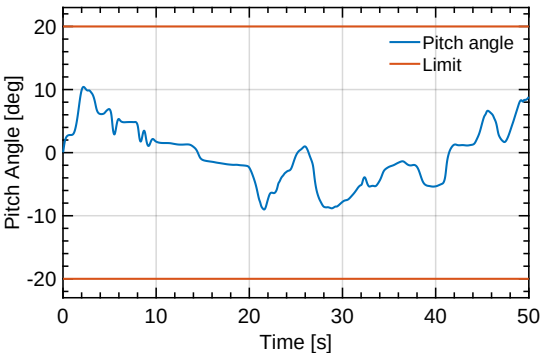
Figure 3.16: **Left**: 50 second snapshot of perceived pitch rate using MPC with vestibular model. **Right**: 50 second snapshot of perceived roll rate using MPC with vestibular model.



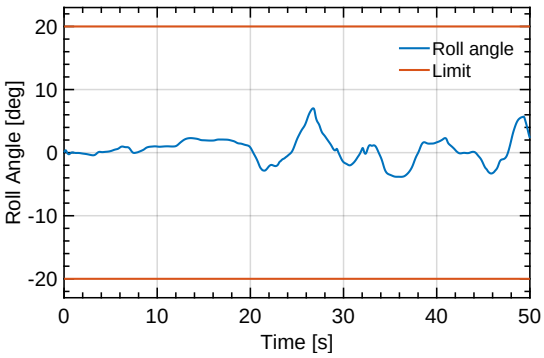
(a) Displacement in X-direction.



(b) Displacement in Y-direction.



(c) Pitch angle displacement.



(d) Roll angle displacement

Figure 3.17: 50 second snapshot of simulator workspace displacement in both x- and y-direction as well as pitch and roll using MPC with vestibular model.

It is expected that the performance deteriorates when the prediction quality for the reference declines. When using a constant reference trajectory the simulated specific force, both in longitudinal and lateral direction shows more missing and false cues due to lag, this can be seen in fig. 3.18. The reason for this to happen is because the algorithm is not able to anticipate on the future reference trajectories as well compared to oracle logic. However, the difference is not as distinct as expected. Next to the possible theories given for fig. 3.14, two other reasons might exist. Because the prediction horizon is limited to $N_p = 50$, in both cases anticipating on the reference is limited as well. In this case, being able to elongate the prediction horizon would result in different motion. Another aspect that adds to this behavior is the missing of an explicit state term in the cost function in Equation (3.38) as can be found in [50], as well as equal error weighting for specific force and rotational rate. The added cost term enables an explicit trade-off between using tilt-coordination and translational acceleration of the platform. In the presented case the majority of the accelerations are performed by tilt. Adding this together with penalizing the error on rotational rate more heavily, as done by [50], would reduce the amount of tilt. It is expected when the overall simulated specific force decreases, the difference between constant and oracle prediction becomes more apparent. The second theory is that because MPC only uses the first computed control input that even when using a perfect prediction the overall difference is not that much. Both of these theories need more investigation to give a conclusive statement.

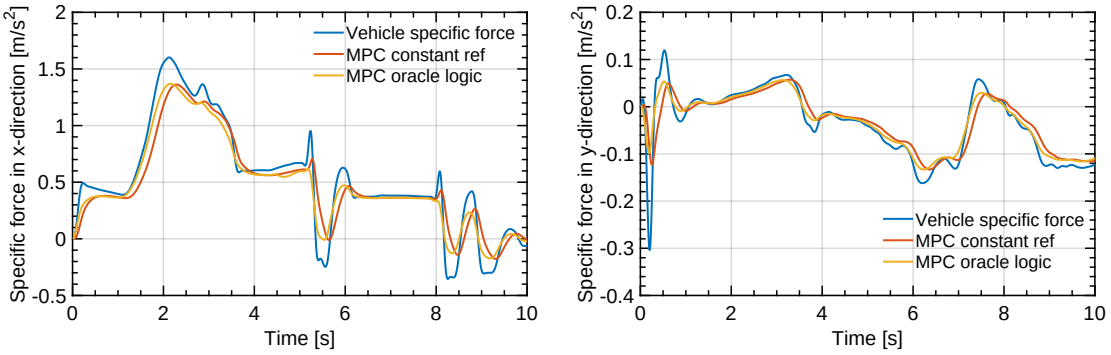


Figure 3.18: **Left**: 10 second snapshot of specific force in x-direction using MPC with vestibular model. **Right**: 10 second snapshot of specific force in y-direction using MPC with vestibular model.

3.3.5. Comparison CWA and MPCMCA

Performance of the CWA has been thoroughly analysed in section 2.2, the performance of the MPC in previous sections. Since the vestibular model was omitted from the CWA, only the double integrator MPC will be compared. In fig. 3.19 and fig. 3.20 the difference between simulated specific force in x- and y-direction is shown.

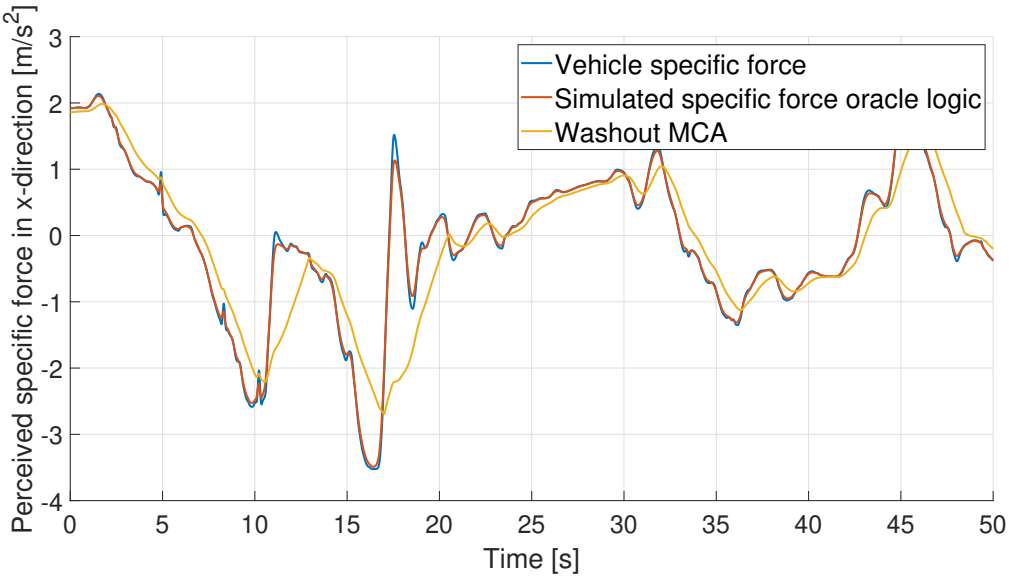


Figure 3.19: Comparison of simulated specific forces in x-direction between oracle MPC using double integrator model and CWA.

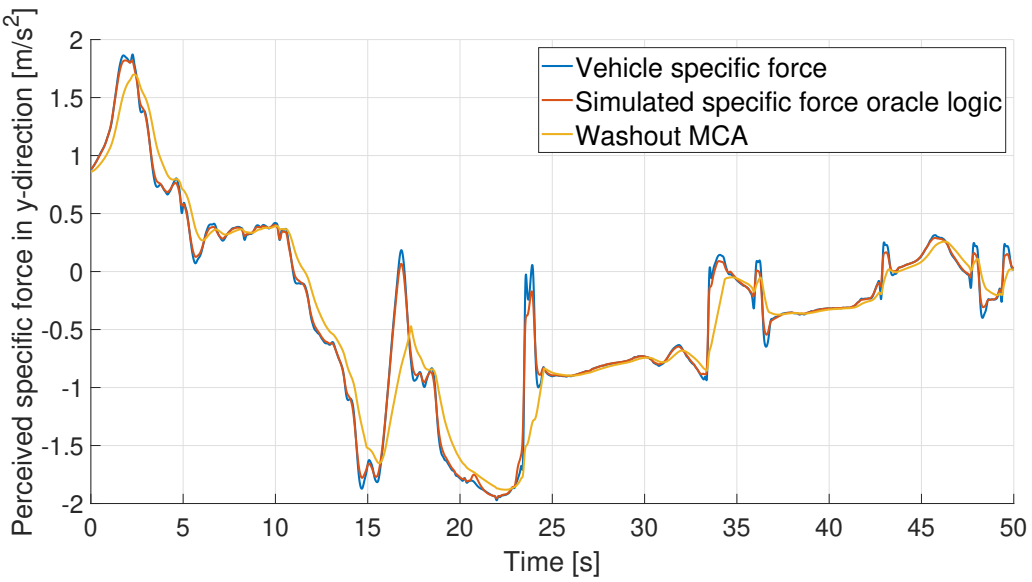


Figure 3.20: Comparison of simulated specific forces in y-direction between oracle MPC using double integrator model and CWA.

The figures above show the differences between the two methods. The MPC based strategy is able to simulate the high frequent specific forces whereas CWA is not able to do this. Also CWA shows signs of providing lagged cues to the driver whereas MPC does not. In further investigation more concrete metrics should be used for comparison, for example using objective motion cueing RMSE scores as presented in [28].

3.4. Discussion and conclusions

Using model predictive control in motion cueing in driving simulation is quite new. The main reason can be found in more efficient solvers that are able to solve complex optimization problems fast enough to guarantee real-time simulation. It was determined that in order to fully understand the use of MPC in motion cueing, a MPC algorithm should be built from the ground up, tested and analysed. All these aspects are elaborated upon in previous sections. The full mathematical substantiation was given, after which several tests were run and its performance analysed and compared to that of the CWA.
As expected the MPC is mathematically more complex than the equivalent CWA. However, its advantages are clearly present. Because simulator limitations can be explicitly incorporated in the optimization process, tuning is only required to change the actual performance of the algorithm instead of limiting outputs. When comparing the CWA and MPC output, it was found that differences in simulator motion exist and based on the perception traces MPC showed to perform better. These findings were close to what was found in literature [28][50]. However, this could differ from the control performance of an actual simulator. Many assumptions are made that can be different for a real-life simulation. First of all perfect simulation is assumed, meaning the model to predict and control are one and the same. Secondly, the decision was made to impose constraints on platform motion, whereas in literature it can be found that usually constraints are imposed on actuator level [26]. Also real-time performance is neglected for the offline simulations, this can have large implications on the performance as the solver is able to use many iterations. Finally, in literature an input scaling factor is often present [31][22][28], in the study presented above this scaling factor was omitted.
Contrary to expectation, the main difference between the oracle reference and constant reference trajectory was the introduction of lagging simulator motion when using the constant reference. It was argued that several factors may have an influence: a larger control horizon could result in a different picture, also obtaining real-time performance was not important which meant that in both scenarios the algorithm is given enough time to find the optimal solution.
To conclude this chapter several recommendations are given for future investigations.

1. The MPC algorithm should be extended to 6 DoF.

2. The cost function should be extended to include an explicit cost on the states, this can be found in literature as well [50][19]. Adding this term would enable making an explicitly defined trade-off between simulation by tilt and by translation. It would also add a neutral push to the simulator.
3. One of the findings of [26] is that elongated prediction horizons and an improvement in prediction quality would improve motion cueing quality. To this extent, the MPC algorithm should be adapted such that it can deal with longer prediction horizons without adding too much computational complexity. One technique that was often used in literature [71][14] is a technique called move blocking, this technique ensures mathematical stability for large prediction horizons while ensuring real-time capabilities [16].
4. One point that has not yet been discussed is the lack of coordinate reference transformations. The MPC algorithm presented in this chapter does not feature these transformations. This should be investigated further, and literature should be consulted on how to include these non-linear transformations into a linear optimization framework.
5. MPC should be validated by comparing the output of this MPC with the MPC algorithm provided by BMW.
6. Quantitative comparison frameworks should be set-up based on metrics used in literature. One possibility for using an objective motion cueing metric could be the RMSE as used by Ellensohn [28].

4

Supervised Data-Driven Modelling Approach

In Chapter 3, the model predictive control paradigm was presented. One of the main advantages of using model predictive control as motion cueing algorithm over the standard CWA is the explicit inclusion of future motion reference while also enabling the use of explicit workspace constraints. This advantage is also one of the main challenges for an MPC MCA, realising an accurate reference trajectory for each time step for the N_p next prediction steps. This chapter commits itself to explaining a supervised data-driven modelling approach to tackle this challenge. The chapter is constructed in the following way. Section 4.1 presents alternatives, found in literature, to the constant reference trajectory elaborated upon in Chapter 3. In Section 4.2 the problem statement as well as different modelling approaches are described. Section 4.3 presents the data sets and possible features that are available to use in training, testing and validating. This section will also present the feature selection. Section 4.3 will be superseded with Section 4.4 where the options regarding data preprocessing are elaborated upon, these include data scaling and prefiltering of data. Because this thesis concerns itself with a supervised data-driven approach, model training requires input-output mappings, the process of generating these mappings will be discussed in Section 4.7. Several different network structures consisting of a multi-layer perceptron model, a recurrent neural network, a deep recurrent neural networks and an encoder-decoder recurrent neural network will be set-up in Section 4.5. Neural network models feature many hyperparameters that influence model performance, such as parameters that define the network structure and affect the learning process. The process of tuning these parameters will be examined in Section 4.8. Section 4.9 elaborates on model performance analysis on an independent test set, both on temporal as well as metric level. Section 4.10 finalizes the chapter with a discussion on the results as well as some recommendations for future work.

4.1. Driver Behavior Modelling Preliminaries

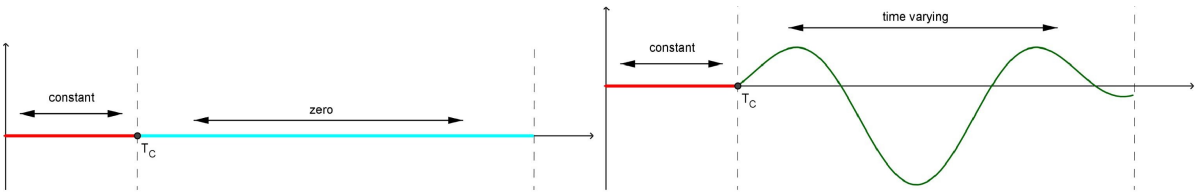
Several other studies suggest different techniques in order to incorporate driver models in real-time MPC to generate a future reference signal. Two studies will be examined.

4.1.1. Switching prediction from non-look-ahead to look-ahead reference

Bruschetta et al. [15] suggest a strategy that is able to switch between a non-look-ahead (NLA) to a look-ahead (LA) prediction in real-time [15]. In order to do so a data-set is built-up by letting a virtual driver drive around a track following a reference trajectory minimizing laptime. This is done for different reference trajectories and different track grip levels ranging from 100% (realistic grip) to 40% grip (heavily reduced grip due to e.g. rain or ice). Several combinations are simulated and data is gathered from all scenarios. Four driver skill level (DSL) categories are devised and the previously recorded data is categorized according to one of the four DSL's. For each data-set the most challenging lap is recognised. This is the lap with the highest peak accelerations in longitudinal and lateral acceleration, i.e. the most difficult to simulate. This data-set is then used as reference. The reference trajectory is built-up in the following way:

1. The first T_c seconds of the future reference is held constant, i.e. the current system outputs.
2. For $t > T_c$ the prediction equals a scaled version of the data from the most challenging lap, i.e. $K_{LA} * r_{MCL}$.

When relevant unexpected behaviour is detected which would significantly impact the MPC applied input (dependent on DSL), the gain K_{LA} is smoothly reduced to 0, effectively reducing the future reference trajectory to a NLA prediction strategy using constant reference. The strategy is shown in fig. 4.1.



(a) LA case used as a NLA strategy (b) LA case

Figure 4.1: Future reference trajectory strategy using a switching NLA and LA prediction [15].

Validation with human-in-the loop experiments show that the driver can be accurately placed within one of four categories. It is shown that utilizing the LA prediction strategy enables an improvement in longitudinal/pitch DoF ranging from 60% to 215% in the value of the gain K.

4.1.2. Driver model based on yaw, lateral error, velocity and gear changing

In the paper written by Drop et al. [24] an online MPC algorithm used as MCA is researched. Another strategy than the one developed by [15] is utilized. The online prediction method simulates a car, by means of a simplified kinematic bicycle model, on a 2-dimensional representation of the actual road. It is argued that the simulation time of the car cannot exceed 1ms due to constraints on the MPCMCA calculation, therefore simplified models are required [24].
As mentioned, the environmental model is a simple 2D representation of the centre line of the actual road. Curvatures are modelled by a clothoid at the start, a section with constant curvature in the middle and a clothoid at the end. Velocities on this 2D path were assumed to be $100km/h$ on rural sections, during the lane change $70km/h$ and $55km/h$ elsewhere. In corners a desired lateral acceleration of $3m/s^2$ is assumed. It should be noted that the driver-in-the-loop may exceed all of above stated values.
The car is modelled according to a simplified kinematic bicycle model incorporating yaw dynamics based on the equations of motion and longitudinal acceleration which is a function of engine force, brake force, drag and rolling resistance. As input it requires a front axle steering angle and longitudinal acceleration in body frame.
The driver model outputs are used as input to the car model. The driver model encompasses a yaw and lateral error controller, a PID velocity controller and a gear shifting controller. The yaw controller is a combination of a first-order feedforward and feedback controller whose output is required for the determination of steering wheel angle. The velocity PID controller acts on the error between the target velocity, based on road position and current car velocity, trying to mimic an average acceleration profile. The gear controller changes gear based on two parameters: current engine RPM and target speed of the car.
Performing experiments with driver-in-the-loop both in an online as well as offline manner resulted in an overall positive experience regarding motion cueing. It was shown that small improvements to the prediction method could result in considerable improvements in motion cueing. It was also found that a non-zero state-error weight in the cost function is regarded as a reasonable method for preventing motion incongruencies due to prediction errors. This state term is defined as the second term in Equation (4.1) [24].

$$J(t) = \frac{1}{N}\sum_{k=0}^{N-1} \|y(x_k, u_k) - \hat{y}_k\|^2_{W_y} + \sum_{k=0}^{N-1} \|x_k - \hat{x}_k\|^2_{W_x} + \sum_{k=0}^{N-1} \|u_k - \hat{u}_k\|^2_{W_u} \tag{4.1}$$

4.2. Problem and Method Description

Up to this point, three different models for reference trajectory generation were investigated.

1. Constant reference trajectory.
2. NLA-LA reference strategy: a hybrid method including a short-term constant reference as well as a long-term component based on reference drives performed by other drivers.
3. An artificial driver model consisting of a simple kinematic bicycle model, using different controllers following the centre line of the road and adhering to the set speed limits.

These three options present their own advantages and disadvantages. Utilizing a constant reference trajectory is feasible in each scenario as it is only dependent on the current state. However, only in rare cases does the state remain constant over the horizon N_p, so more often than not this prediction does not provide high accuracy. Although the second option proved to give good motion cueing results, it requires a prerecorded reference drive which largely compromises its usability. Any deviation/stochasticity in the driving scenario would make this option unusable. The last option has the advantage of both previous options, it is very flexible and can be used in almost every scenario. However, when the general driving direction is unknown, for example in an urban scenario where the intent of the driver is unknown, e.g. which exit on the roundabout will the driver take, the inclusion of some heuristic/rule-based logic (e.g. based on turn signal indicator) which predicts future driving behavior is required. However, even in the cases where it can be used it should be noted that this controller is tuned to match an average acceleration profile and is therefore not able to catch driver specific behavior [24].
The upcoming sections propose the pursuit of another way of generating a reference trajectory: namely a supervised data-driven neural network model. The term ”supervised” denotes the act of training the model in offline fashion. Effectively this means that training data is fed to the model offline, after which the trained model can be used to make online predictions given certain (sequences of) input features. The reason to investigate such models is because modelling driver behavior is a stochastic matter. Therefore, it is difficult/impossible to give an analytical description of the governing equations that model driver behavior. Given enough data and training time a neural network serves as a ”black box” model that is able to learn dependencies in data to make predictions about future states. In Chapter 3 a 4 DoF MPC algorithm was presented, this MPC algorithm requires a reference trajectory for both longitudinal and lateral acceleration and pitch and roll rate. However, as will be explained in Chapter 5, validation will be done on a multi-DoF simulator requiring a reference trajectory in all 6 driving DoFs. This means the model needs to be able to predict the trajectory in all 6 DoFs independently. Because MPC requires a discretized temporal, i.e. time and state dependent, reference trajectory it is required that for each DoF a prediction needs to be made for each sample in the ”N_p” long prediction horizon. These two requirements lead to the following problem statement:

For accelerations in all translational directions as well as for rotational rates around each axis a supervised data-driven model must give a prediction for each sample within the prediction horizon of length ”N_p”.

In order to comply with the problem statement a methodology was set-up to understand what actions need to be undertaken. This methodology follows the ”engineering design process” [29], normally subdivided into 6 steps: Idea → Concept → Planning → Design → Development → Launch. However, to tailor the process to the problem at hand, three changes are implemented. Firstly, the ”concept” phase is replaced by data review. Based on the data review and the problem statement, design concepts can be realised. Next to this, the ”planning” phase is omitted all together. Thirdly, ”Design” and ”Development” are combined into one phase ”Design and Optimization”. And lastly, the final step ”launch” is replaced by ”Integration and Validation”, since the thesis concerns itself with software integration and not product development.

1. Idea: The first thing to do is to come-up with an idea, a problem that needs to be solved. This problem is defined in the statement given above. The substantiation for pursuing a solution to this problem is given in all preceding chapters.
2. Data review: It is already stated that a data-driven model will be used, therefore the concept phase entails a review of the data. This includes an investigation on the available dataset, e.g.

what were scenario specific elements and in which environment was the data recorded. It also includes an investigation on which signals are present in the dataset. Normally part of the concept development is also the conceptualization of models and their structure, however this will be part of step 3: model design and optimization.

3. Design and Optimization: Based on the data review in the previous step and the problem statement, several different model structures are set-up. Part of this process includes the data preparation which entails data preprocessing as well as input-output mapping creation.

4. Integration and Validation: In this phase the results of the prediction models will be integrated with the MPC algorithm and preliminary results will be analysed. Based on the preliminary simulation results, research hypotheses and an experiment design tailored to the research question will be established. The experiment serves as validation.

4.3. Data Review

The data-set contains 43 independent drives obtained during a dynamic driving experiment in a rural environment [70]. The road consisted of two opposing single lanes, one going in North-to-South (NS) and one in South-to-North (SN) direction. Participants were asked to drive in one of two direction, and drive as they normally drive every day, without any time pressure, adhering to the traffic regulations that are in place. A figure depicting the road lay-out is given in fig. 4.2 . The participants conducted the experiment driving a model of a 2018 BMW 530i. The total of 43 drives can be split-up in 13 drives from NS, 24 drives from SN and 6 drives that were abandoned prematurely, e.g. due to motion sickness. The latter 6 drives are omitted from the data-set. For each experiment 62 different variables were logged, ranging from simulator specific variables to kinematic vehicle model states.

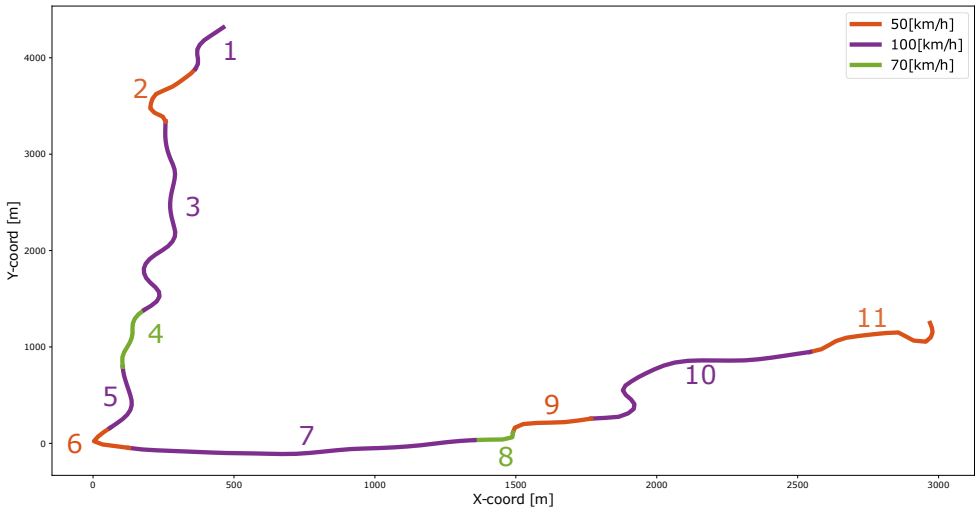


Figure 4.2: Road layout segmented based on local speed limit.

A description of the rural road segmentation is given in table 4.1.

Table 4.1: Explanation on segmentation rural road on which experimental data was gathered.

Road Segment	***Situation***
1 → 2	Rural road entering village
2 → 3	Leaving village to rural road
3 → 4	Sharp hilltop, reduced speed limit rural road
4 → 5	Sharp hilltop, increased speed limit rural road
5 → 6	Approaching roundabout, reduced speed limit
6 → 7	Leaving roundabout, increased speed limit rural road
7 → 8	Sharp turn, reduced speed limit
8 → 9	Approaching village, reduced speed limit
9 → 10	Leaving village, increased speed limit
10 → 11	Entering village, reduced speedlimit

The reason why data is reviewed has multiple reasons, the first one is to make sure logged data has no offset errors or different/low signal-to-noise ratio when comparing different participants to each other. Finding such behavior and correcting them before using them to train models can have a large impact on the quality of the model. The other is to find out how signals differ between participants, i.e. if large variations between drivers can be found and where exactly these deviations occur.

4.3.1. Speed profile

The first signal that is analyzed is the velocity profile of different drivers to check whether participants adhere to imposed speed limits. A figure of the velocity profiles for both NS and SN direction can be found in fig. 4.3. One can see that there is a noticeable difference between the two drives, in case of the NS drive more people seem to underestimate or overestimate their own speed causing them to drive quicker or slower than the imposed speed limit. Whereas in SN direction less people seem to underestimate their speed, causing the majority of participants to adhere to the speed limit. However, there is one participant in the SN direction who severely underestimates his own speed at sections 2,3, 7 and 8 which causes him to drive too fast at each of these sections. Although, significantly different this drive is not omitted from the data set to keep variation in the data set during training. Another thing to be noted is that in section 10 participants seem to drive significantly slower than the imposed speed limit of $100 km/h$, this could have two reasons. The first option is that participants felt enforced to drive slower than the imposed $100 km/h$ because the road is quite swirly at segment 10. The other option is that a there is a mismatch between the road sign a participant sees and the speed limit definition found in the software. Whatever the reason, it does not matter for this application because of the following. Since these signals will be used as input feature to the neural networks to predict driver behavior, in case of option 1 the model will learn that accelerative behavior is not only dependent on the difference with the speed limit but also on the road layout (this is also part of the feature set as discussed in a later section). In case of option 2, one should note that this thesis research serves as an exploratory study to understand the effectiveness of using neural networks to do driver prediction in a certain environment. If the results are positive, further research should be performed to understand to which capacity such a network can be used in a real-time scenario, dealing with definition mismatches is part of that investigation.

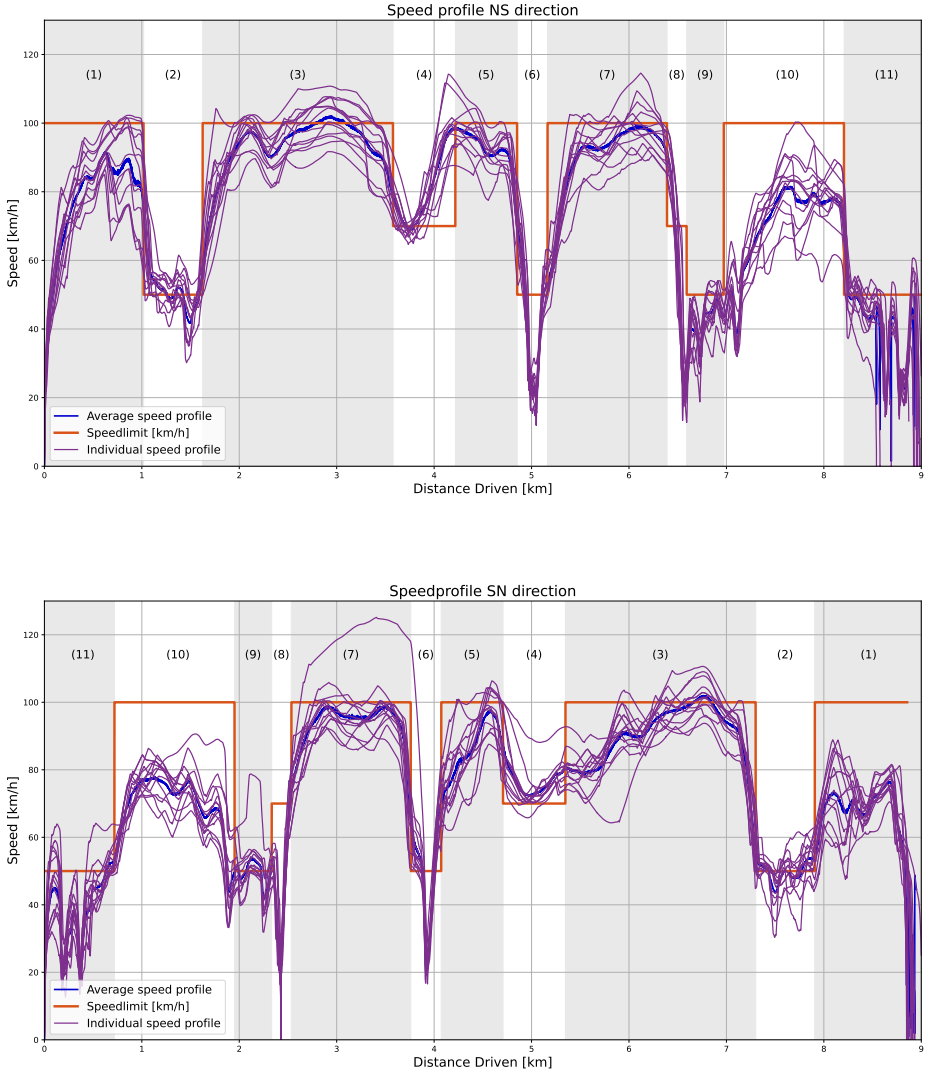


Figure 4.3: Velocity profiles featuring all participants in both North-to-South as well as South-to-North direction.

4.3.2. Road curvature and steering wheel angle

When investigating the road curvature it was found that because the road curvature is created based off splines, it is wrongly defined at certain locations and high peaks exist between samples. To counteract this effect, the road curvature is low-pass filtered with a $0.2Hz$ 2nd-order butterworth filter [87] using Python's Scipy *butter* and *filtfilt* functionality [100]. As for the steering wheel angle it was found that an offset was included in the data, this offset is removed by subtracting the original signal with the mean steering wheel angle for each participant independently. Both results can be found in fig. 4.4.

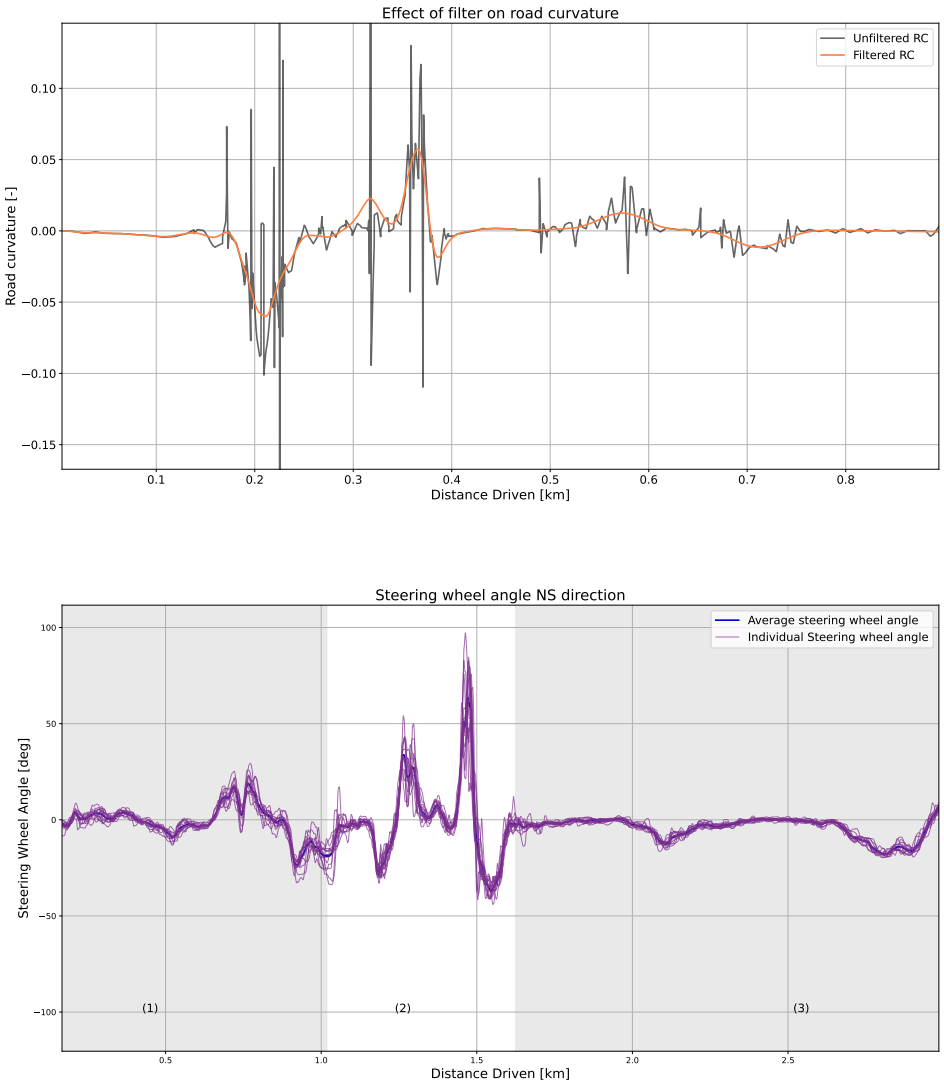


Figure 4.4: **Top**: Effect of 0.1Hz 2nd-order low-pass butterworth filter on road curvature.
Bottom: Steering wheel angle profile without offset error.

4.3.3. Acceleration profile

The last signals that are analyzed are the longitudinal and lateral acceleration profile of both drives from NS and SN. A figure for both the longitudinal and lateral acceleration in NS and SN direction can be found in fig. 4.5 and fig. 4.6, respectively.

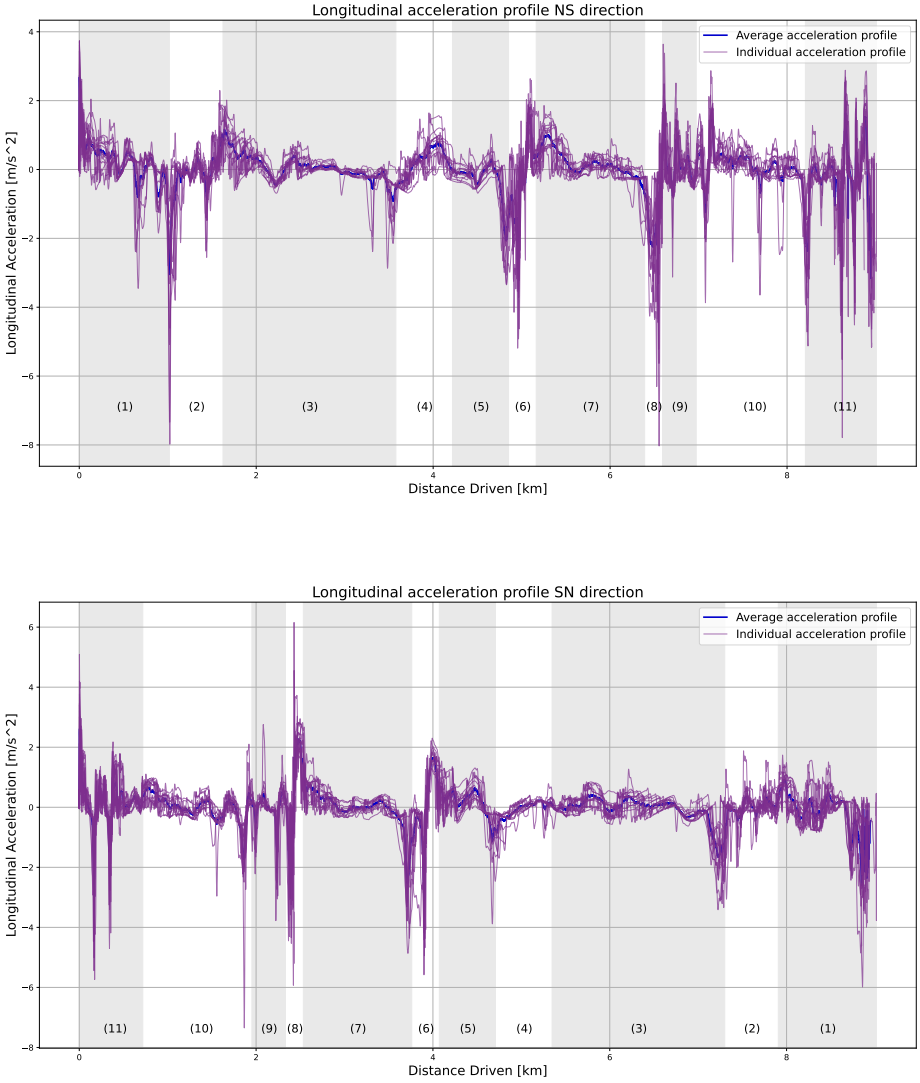


Figure 4.5: Longitudinal acceleration profiles featuring all participants in both North-to-South as well as South-to-North direction.

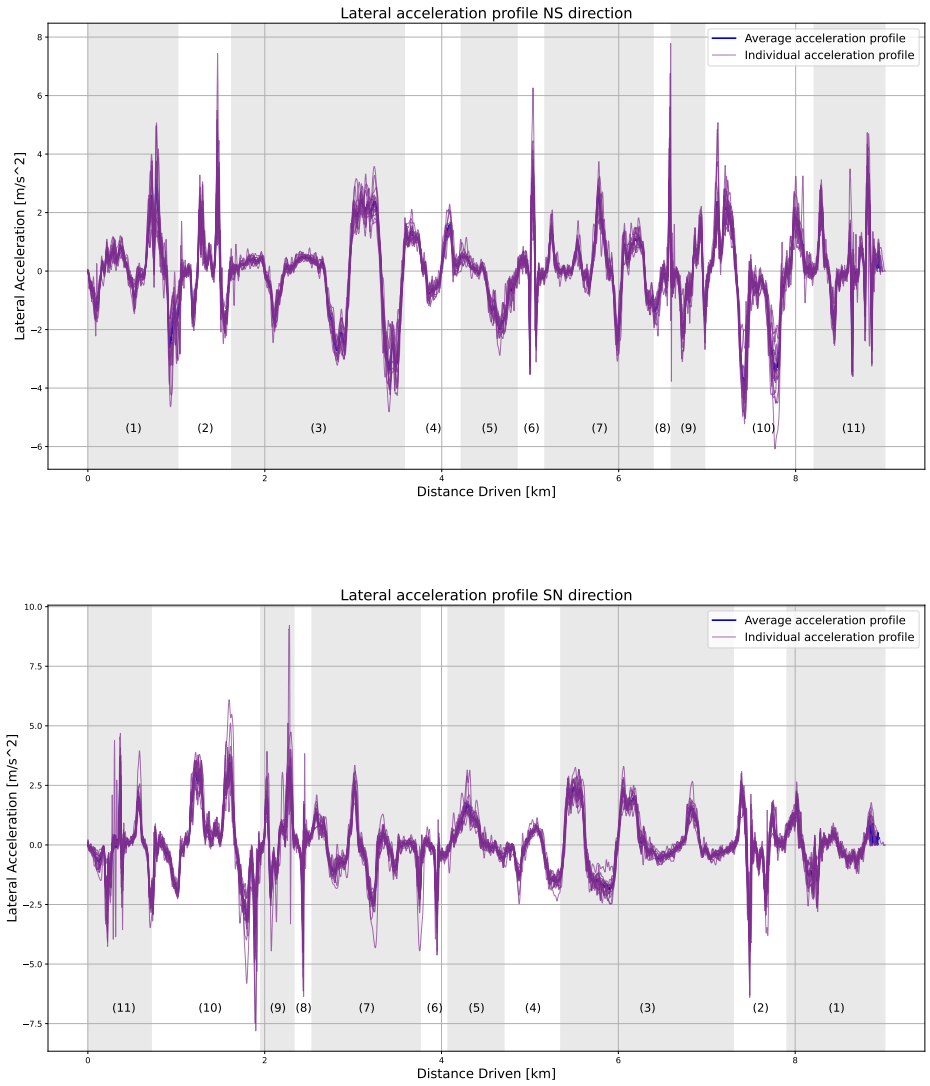


Figure 4.6: Lateral acceleration profiles featuring all participants in both North-to-South as well as South-to-North direction.

It should be noted that the acceleration signal suffered from high frequency spikes due to gear shifting. Since it is not desired for the model to predict high frequency accelerations due to gear shifts, the acceleration signals are low-pass filtered with a $1Hz$ 5th-order butterworth filter, the resulting signal can be found in fig. A.13.
From the figures it seems as if the spread of data around the mean is larger for longitudinal acceleration than for lateral acceleration. This observation agrees with what can be found in literature as well [28]. The hypothesis is that the model will find it more difficult to predict longitudinal than lateral acceleration.

4.4. Feature Analysis and Data Preprocessing

This section provides a discussion on the input and output feature selection as well as data preprocessing.

4.4.1. Input and output features

Selecting the output features that the model needs to predict is in this case straightforward, they are defined by the reference the MPC MCA tries to optimize towards. The output features are defined in table 4.2.

Table 4.2: Output features determined by MPC MCA reference trajectory requirement.

Output feature	Unit
Longitudinal acceleration	m/s^2
Lateral acceleration	m/s^2
Vertical acceleration	m/s^2
Pitch rate	rad/s
Roll rate	rad/s
Yaw rate	rad/s

The selection of input features is not as straightforward, because in this case it is not fixed by an external requirement. The easiest option is to use all available signals as input features to the model and let the model learn the dependencies by itself. However, this does come at the cost that model complexity increases with the amount of input features being used [58]. Also performance of the model might decrease by adding too many input features. This can occur because the average feature quality, i.e. feature richness, will go down and the model might overfit on features that have no correlation to the predicted output feature [58]. This makes training more difficult. Selecting only very rich features is the goal, but a difficult task, the process of selecting features is called ”feature selection”. Several feature selection techniques exist, three of them are highlighted below.

1. **Filter method**: Several filter approaches exist for selecting relevant input features. One of the more simple methods is calculating the correlation between input and output features. The output of the algorithm is the subset of input features that show the highest correlation between individual input and output features and that show low input feature cross-correlations. This way redundant features, e.g. two input features having large correlations, are omitted from the input feature space [92][47].

2. **Wrapper method**: This is a model specific approach, where wrapper methods select the subset of features that give the model the best performance. Wrapper algorithms are computationally expensive because the performance of the model needs to be evaluated multiple times, each time with a different combination of selected features. An example is the ”exhaustive search heuristic”, which evaluates all possible combinations of input features and selects the combination with the best model performance. Another method often used is forward selection, where the algorithm starts with zero features and adds one at each iteration and stops when a certain threshold in performance and number of features is obtained [57][12].

3. **Embedded methods**: Feature selection is embedded within the machine learning algorithm. One of the methods commonly used to reduce the input feature space is ”L1-regularization”. L1-regularization adds a penalty term to the network’s cost function equal to the sum of the absolute values of the parameters in the model. Applying L1-regularization to an algorithm often results in a sparse feature set, causing some features to equal zero. This effectively reduces the input feature space [65].

Based on the specific data-set, the amount of possible input features that are present within the data-set, and method specific advantages and disadvantages, one of the three specified methods can be selected. The goal of this study is to provide a framework with which one can predict the output features presented in table 4.2. Therefore, advanced feature selection techniques are not considered at this stage.
However, feature selection is based on intuition and examples of driver behavior modelling found in literature [2][49], that present some features used for prediction. In order to select the features, they are categorized in two classes: causal (past-time) features and non-causal (look-ahead/future) environment features. The reason why this division is made will be apparent in section 4.7. These classes can be sub-categorized into environmental, driver specific and vehicle features. The full input feature list and their class and sub-class categories are given in table 4.3.

Table 4.3: Input features used for driver prediction, categorized into causal and non-causal (<X [m]>look-ahead) features.

Input feature	Class	Subclass	Link to output feature
Absolute road curvature	Causal	Environmental	- Longitudinal/Lateral acceleration - Roll, pitch, yaw rate
Sign road curvature [-1; 1]	Causal	Environmental	- sign longitudinal/lateral acceleration - sign roll-, yaw rate
Road width	Causal	Environmental	- Longitudinal/Lateral acceleration - Roll, yaw rate
Speed limit	Causal	Environmental	- Longitudinal acceleration - Pitch rate
Road elevation	Causal	Environmental	- Vertical acceleration - Pitch rate
Absolute road curvature	Non-Causal <X [m]> look-ahead	Environmental	- Longitudinal/Lateral acceleration - Roll, pitch, yaw rate
Sign road curvature [-1; 1]	Non-Causal <X [m]> look-ahead	Environmental	- sign longitudinal/lateral acceleration - sign roll, yaw rate
Road width	Non-Causal <X [m]> look-ahead	Environmental	- Longitudinal/Lateral acceleration - Roll, yaw rate
Speed limit	Non-Causal <X [m]> look-ahead	Environmental	- Longitudinal acceleration - Pitch rate
Road elevation	Non-Causal <X [m]> look-ahead	Environmental	- Vertical acceleration - Pitch rate
Steering wheel angle	Causal	Driver specific	- Lateral acceleration - Roll, yaw rate - Vehicle dynamics
Throttle [0-1]	Causal	Driver specific	- Longitudinal acceleration - Pitch rate - Vehicle dynamics
Brake [0-1]	Causal	Driver specific	- Longitudinal acceleration - Pitch rate - Vehicle dynamics
Longitudinal acceleration	Causal	Driver specific	- Longitudinal acceleration - Pitch rate - Vehicle dynamics
Lateral acceleration	Causal	Driver specific	- Lateral acceleration - Roll rate - Vehicle dynamics
Speed	Causal	Driver specific	- Longitudinal acceleration - Pitch rate
Yaw rate	Causal	Driver specific	- Lateral acceleration - Roll, yaw rate - Vehicle dynamics
Roll rate	Causal	Vehicle	- Lateral acceleration - Roll rate - Vehicle dynamics
Pitch rate	Causal	Vehicle	- Longitudinal acceleration - Pitch rate - Vehicle dynamics

Not all of the features listed above were part of the attributes in the data-set. The absolute road curvature, the sign of the road curvature (left corner is positive, right corner negative), the road elevation and acceleration in Z both causal and non-causal had to be calculated and added to the data-set. It is worth noting that the reason why the non-causal signals are distance and not time-based is to make sure the amount of information the model sees at each time step is independent of other attributes such as speed and input sample length (e.g. $x = v \cdot t$). It also makes more sense as a person, regardless of the amount of input samples and speed they are driving, is able to anticipate road conditions a certain fixed distance in the future, assuming the road is not visually obstructed. The reasoning behind the majority of the presented signals is the direct link with the to-be-predicted output. For example, it is argued that the magnitude of the road curvature influences the braking behavior, as such longitudinal

acceleration, before the turn but also influences the lateral acceleration of the vehicle in the turn. It is of course hard to say if the network will learn these exact dependencies between features or if it also learns more high dimensional dependencies between features. The reason why the attributes that are linked to vehicle dynamics are included is that the network can learn a representation of the dynamics/inertia of the simulated vehicle as well, e.g. given a certain speed and throttle input, what kind of acceleration is to be expected?

4.4.2. Data preprocessing

Data preprocessing is a very important step before setting up and training the model. Data preprocessing constitutes several techniques, of which a couple are given below [55].

- **Removing Null/Nan values:** It often occurs that certain samples within the dataset are missing, for these samples the Null value is written. A NN model cannot cope with Null values within features, therefore these have to be replaced by a different value. Depending on the data type and structure, different techniques exist to replace Null values [40][55].
- **Outlier detection:** Sometimes measurements could be wrong, and large outliers could be present in the data. These are often removed/replaced [40][55].
- **Categorical data:** Machine learning models consist of many mathematical equations, as such categorical data (e.g. city names) cannot be used in their raw form but need to be encoded/transformed to a numerical data type [55].
- **Noise reduction**: When signals feature too much noise, signals are often low-pass filtered.
- **Scaling:** Using the raw attribute values in the dataset is considered bad practice and will decrease the performance/generalization of the model [55][89]. The reason is because the value ranges (often because of the unit) between input features can be vastly different within or between scenarios (city vs highway driving). For example, the value range of throttle input is [0-1], of speed is [0-130] km/h, while longitudinal acceleration is within the range [-9, 9]m/s^2. This could result in the model putting more weight and thus more importance on those features with higher values, which is not desirable. Next to this, when using for example sigmoid activation functions in the model, where an input is mapped to a value between 0 and 1, large discrepancies between value ranges could lead to values being squashed or saturated which makes training difficult/impossible. To solve this, all features are scaled to have the same value range. Scaling is often done with either min-max scaling or z-score normalization [55][89].
- **Train-Test-Validate**: The performance on the training data is not important as good results do not automatically cross-over to good performance on unseen data. Therefore the data-set needs to be split-up into a training, validation and test set. The validation set is required to keep track of under or overfitting during training. Ultimately, it is the performance on the unseen test-set that is important [55].
- **Re-sampling**: Depending on the desired sampling rate of the output, and the amount of time steps the data is often re-sampled (in most cases down-sampled) [55].

In this case Nan values are checked and if necessary, removed. Outliers are checked for, data is scaled, and separated in the equivalent training, validation and test set and the data is down-sampled. Since the data was gathered in a virtual environment, the overall data quality is high: it was found that no Nan values were present in the dataset. When determining if there were faulty outliers in the dataset, it was also found that no faulty outliers were present. As for splitting up the data into the equivalent sets, it was decided to take roughly (rounded to integer amount of drives) 80% of SN and NS drives as training set (20 SN drives and 9 NS drives), and 10% (equals two SN and two NS drives) for both test and validation set. The frequency at which all attributes are recorded is $100Hz$, this means that for a prediction horizon of 10s the model should predict 1000 samples. This leads to model with increased complexity, reducing computational efficiency. To artificially reduce model complexity, the signals are down-sampled to obtain 50 prediction samples, which results in a down-sampling frequency of $5Hz$.

Scaling

Many different scaling techniques exist, but only two of them are considered in this case, min-max scaling and z-score normalization. The formula for both are given in eq. (4.2) [55].

$$\begin{aligned} min-max: X' &= \frac{X - X_{min}}{X_{max} - X_{min}} \cdot (F_{max} - F_{min}) + F_{min} \\ z-score: z &= \frac{X - \mu}{\sigma} \end{aligned} \tag{4.2}$$

Where X' and z denote the scaled value of the attribute, X denotes the current value, X_{min} and X_{max} denote the minimum and maximum value present in the data. F_{min} and F_{max} denote the feature range to which the signal should be transformed, where $F_{min} = 0$ and $F_{max} = 1$, μ and σ denote the mean and standard deviation of the signal respectively. It is very important that the attribute specific values, X_{min}, X_{max}, μ and σ, are independent from the test and validation set, i.e. the test and validation set should be excluded when determining these values. This is to make sure no information leaks from the unseen data to the training set, this could lead to the model implicitly developing a bias/overfit on unseen data which is not desirable. A plot showing the resulting distribution of the input features scaled with both min-max and z-score normalization scaling can be found in fig. 4.7.

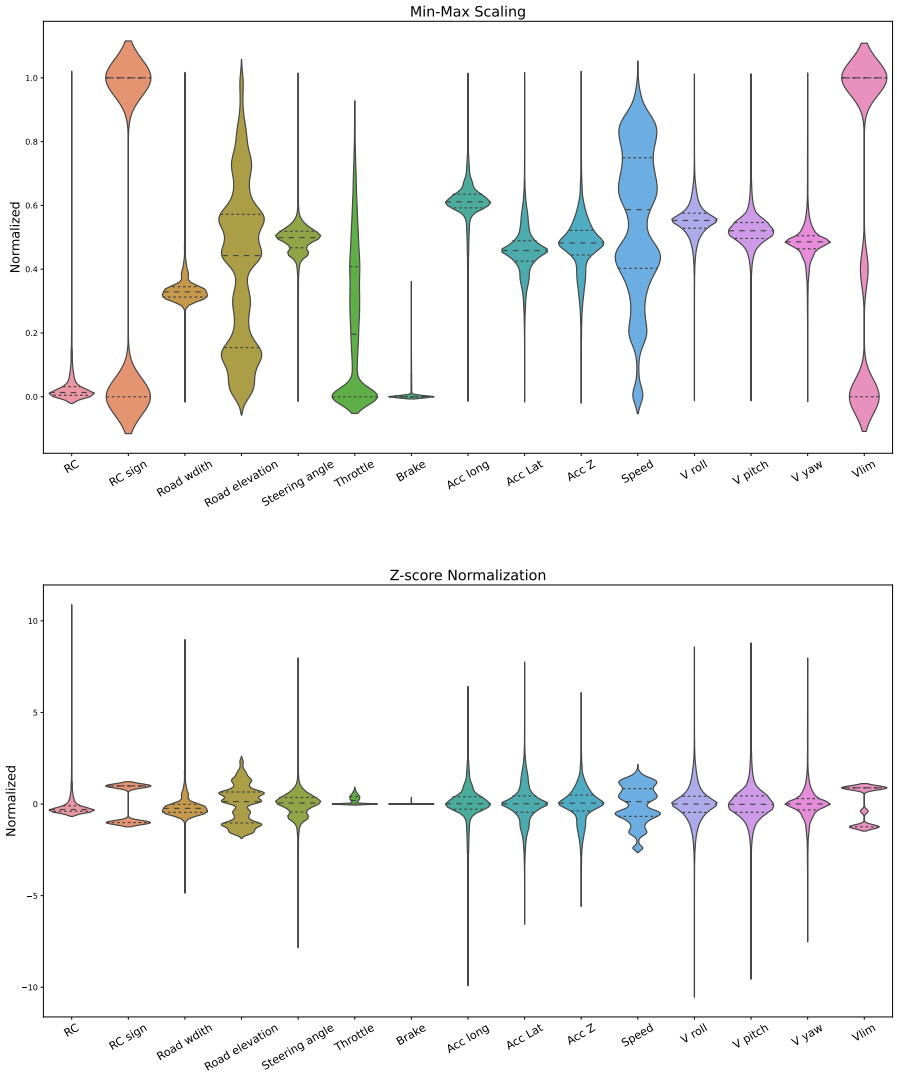


Figure 4.7: Violin distribution plots of input features scaled with a min-max **(top)** and z-score normalization scaling **(bottom)**.

When looking at the individual distributions, one can distinguish a couple of noteworthy features. First of all one can see that features that possess a near symmetric distribution with a mean of zero after z-score normalization are symmetric around 0.5 after min-max scaling, which makes sense since the feature range is between 0 and 1. Next to this, looking at the speed distribution three blobs can be distinguished, corresponding to the three different speedlimits. Something else that catches the eye is the distribution of both the road curvature as well as the brake distribution, in both cases one can see that the majority of density lies around 0. This makes sense as the amount of high radius corners is limited, as well as the amount of heavy braking points.
When comparing both distributions with each other one can see that the distribution in case of the min-max scaling is enforced to lie between 0 and 1 whereas in the case of z-score normalization the value range is defined by the signal distribution parameters giving rise to inter-attribute value range differences. As discussed before, large variations in value ranges between features could lead to bad model performance. For this reason min-max scaling is chosen. To get the most performance out of the models, the effect of using different scaling should be investigated at a later stage.

4.5. Network Model Structures

This section presents three different model types that are utilized to perform temporal driver dynamics regression. The first model discussed is a vanilla feed-forward neural network, the second one a deep-learning recurrent neural network and the last one a deep-learning encoder-decoder network. In order to build, train and test the network, the Keras API which uses Tensorflow as backend was used [17].

4.5.1. Multi-layer Perceptron Model

A multi-layer perceptron model, in short MLP, is often described as the classic/vanilla form of a neural network. It is a fully connected, feed-forward neural network featuring at least three layers, an input, a hidden and an output layer. The term ”fully connected” indicates that all nodes between two adjacent layers are connected with each other. The term, feed-forward indicates that information flows from the input to the output without information flowing back through the network [83]. Although different neural network structures explicitly exist to perform temporal sequence prediction (such as recurrent neural networks), a vanilla MLP is also able to perform the same task, in some cases showing competitive results [93][13]. Therefore it is chosen to use the simpler MLP as a prediction quality benchmark. A schematic overview of a three layer MLP featuring three input nodes, five nodes in the hidden layer and two output nodes is given in fig. 4.8.

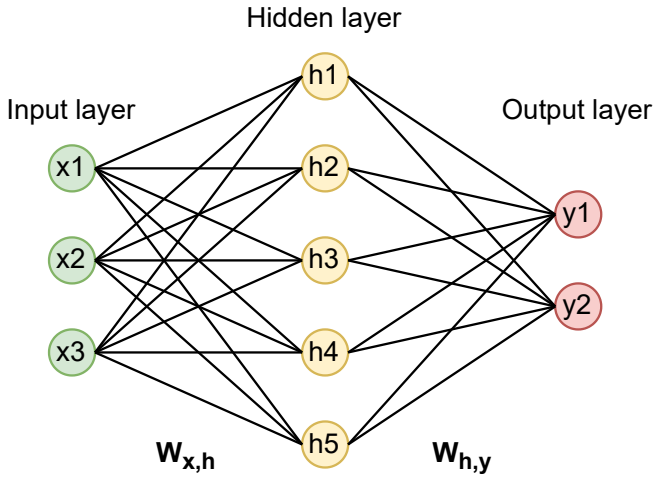


Figure 4.8: Schematic of one hidden-layer vanilla multi-layer perceptron network.

In this figure all the circles depict ”neurons” and the lines that connect them are called ”(synaptic) weights”. Each neuron is a processing unit that transforms a certain input to a certain output, e.g. a hidden layer neuron can be described as $h_{out} = \rho(h_{in} + b)$, where the function ρ is called the ”activation

function" for hidden layer neuron *h* and *b* a bias term that is added that enables shifting the activation function to the left or right, see fig. 4.9. A negative bias shifts the activation function to the right, a positive bias shifts it to the left. Many different activation functions exist, but currently for MLP's, the nonlinear rectified linear unit (its function is given in Equation (4.3), a plot in fig. 4.9), or Relu, is the benchmark activation function [40].

$$\rho_{Relu}(x) = max(0, x) \tag{4.3}$$

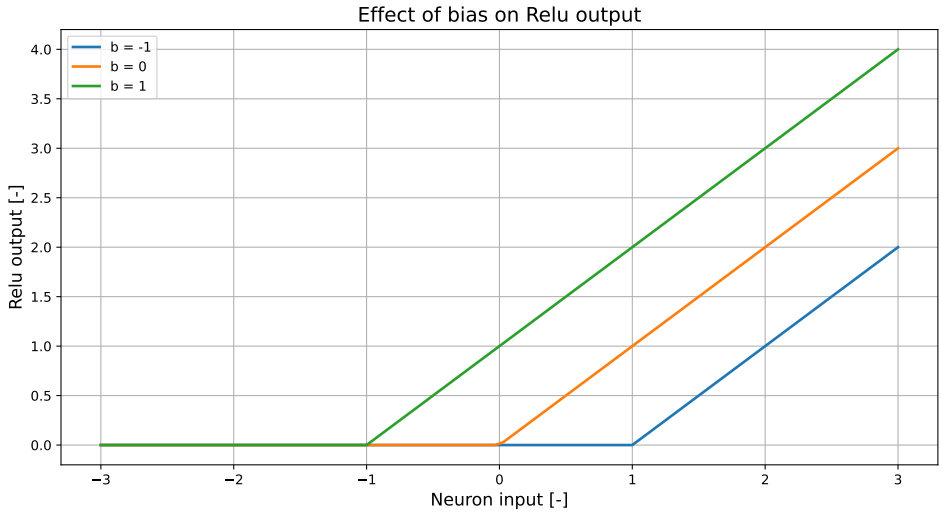


Figure 4.9: Influence of a varying bias term on the output of a Relu neuron.

The reason for the Relu's popularity is because, contrary to a squashing sigmoid (*sig*) or hyperbolic tangent (*tanh*) activation function which transforms an input to a value range of [0, -1], and [-1,1], respectively, it eliminates the risk of vanishing gradients [40]. This risk occurs in deep networks with many hidden layers, where the input to the activation function is the weighted sum of outputs from previous neurons. These neuron outputs are a function of the previous weight value times the value of the neuron itself. When working out the mathematics, it can be found that recursively writing the output of a neural network gives a function of many weight multiplications with the input neurons flowing through layers of activation functions. The equations for the hidden layer, and output layer neurons are given in eq. (4.4) [84]. In a model, featuring many hidden layers, it can occur that the outputs of the neurons quickly diminish, i.e. converge to zero. This occurs due to recursive multiplication of many values that lie between -1 and 1. As an effect information flow stops at a certain point making it impossible for a network to train [9]. The problem of vanishing gradients becomes more clear when discussing how a network is trained using gradient descent approaches.

$$\begin{aligned} h_{out} &= \rho(\vec{W}_{x,h} \times \vec{x} + b_x) \\ y_{out} &= \rho(\vec{W}_{h,y} \times \vec{h}_{out} + b_h) \\ &= \rho(\vec{W}_{h,y} \times \rho(\vec{W}_{x,h} \times \vec{x} + b_x) + b_h) \end{aligned} \tag{4.4}$$

Although Relu is stated to be the benchmark, it does have one big disadvantage. Relu activation functions do suffer from a phenomenon called "dead Relu's" which occurs when the total input is negative, effectively mapping the input to zero [40]. Looking at Equation (4.4), this occurs when $\vec{W}_{x,h} \times \vec{x} + b_x < 0$, which happens when both terms are negative or when one is more negative than the other. Dead Relu's negatively impact training performance, when many dead Relu's are present in the network the gradients will saturate to zero which stops gradient flow through the network which means the network stops learning. When a network features many neurons, and it stops learning, it could mean that the gradient updates are too large pushing all biases to large negative values eliminating

the possibility for the Relu to recover. This can be resolved by lowering the learning rate or by utilizing a variation on Relu called "Scaled Exponential Linear Unit" or Selu for which the adapted equation is given in Equation (4.5) [53].

$$\text{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } \alpha \leqslant 0 \end{cases} \tag{4.5}$$

Training

Training neural networks effectively means altering the weights such that the error between the network's predicted output and the labelled (true) output is as small as possible. This means dealing with an optimization task which, for a vanilla MLP, can be formulated as follows.

$$Minimize \;\; E = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, \vec{W}), y_i) + \lambda \cdot R(\vec{W}) \tag{4.6}$$

Where "N" depicts the amount of training input-output mappings that are used, L_i is defined as the loss function (will be further elaborated upon in "metrics") which is a function of both $f(x_i, \vec{W})$, i.e. the output of the network (function of input x_i and weights $\vec{W}$), and the true output y_i. The added term $\lambda \cdot R(\vec{W})$ is a regularization penalty, where λ is a weighting term that is used to make a trade-off between model performance on training data and obtaining a simpler neural network [82][56]. Regularization if discussed further in Section 4.6. Often the simpler model has a better fit on unseen data than a complex model which tends to overfit [75].
In this thesis only backpropagation optimization algorithms are considered, where the weight optimization relies on taking the gradient of the error with respect to the output, by applying the chain rule it is then possible to derive a formulation for the change in error with respect to each input and each individual weight [81]. The most basic version of a backpropagation algorithm using gradients is called "gradient descent" and works in the following way [81]. First, one needs to calculate the gradient $\delta E/\delta y$, i.e. the partial derivative of the error with respect to each output. By applying the chain rule one can define $\delta E/\delta w$, i.e. the partial derivative of the error with respect to each individual weight in the network. When all the gradients are defined, a weight update occurs in negative direction w.r.t. the calculated gradient, for gradient descent the update rule is defined as found in Equation (4.7).

$$\Delta \vec{W} = -\epsilon \cdot \frac{\delta E}{\delta w} \tag{4.7}$$

Where ϵ is called the learning rate, it defines by how much the gradients change each training iteration. More elaborate algorithms that take more parameters, to make a weight update, into account exist. Two common ones are:

1. Gradient descent with momentum: Similar to the vanilla gradient descent update rule, however a term is added in the update rule: $\Delta \vec{W} = -\epsilon \cdot v_{t+1}$, where $v_{t+1} = \frac{\delta E}{\delta w} + \alpha v_t$. Momentum does not update weights based on the gradient but based on the velocity which has the gradients incorporated, with α being an exponential decay factor, often called "friction coefficient". Adding this term increases rate of convergence by pushing the weights towards the optimum as well as averaging out oscillations that occur in normal gradient descent when near or at saddle points [81][73].

2. Adaptive momentum estimation (or Adam): This is an algorithm often used for optimizing machine learning networks due to its efficient solving capabilities [52]. Adam combines two algorithms, namely Adagrad [25] which accelerates optimization in dimensions with small gradients, holds back optimization in dimensions with large gradients by introducing an implicit learning rate decay factor (with bad parameterization this can lead to an optimization stop). The other one being RMSprop [43] which is similar to Adagrad but adds in an explicit decay factor which reduces the implicit learning rate decay over time. The result is an algorithm which includes ideas from momentum (building up a velocity term) as well as ideas from RMSprop (acceleration in dimension with small gradients and vice versa without strong learning rate decay). Because Adam

requires null initialization which leads to a large first update step, decaying bias terms are added to minimize this effect, the full mathematical background can be found in [**?**].

Due to the efficient nature and widespread use of the Adam optimizer, Adam is used to train the networks proposed in this thesis.
Before the training process can start, all the weights in the network need to be initialized to retrieve an initial gradient value. For all models, the Glorot uniform initialization was used [39]. Where for each neuron a value is generated by sampling from a uniform distribution with limits dependent on the amount of inputs and outputs of that weight tensor.

Metrics

Deciding on how to calculate the error is not a trivial task and many different metrics exist, depending on the task at hand some might make more sense to use than other. For regression, equations for three well known metrics are given below.

- Mean absolute error:

$$MAE = \sum_{i=1}^{n} \frac{|f(x_i, \vec{W}) - y_i|}{n} \tag{4.8}$$

- Mean squared error:

$$MSE = \sum_{i=1}^{n} \frac{(f(x_i, \vec{W}) - y_i)^2}{n} \tag{4.9}$$

- Root mean squared error:

$$RMSE = \sum_{i=1}^{n} \frac{\sqrt{(f(x_i, \vec{W}) - y_i)^2}}{n} \tag{4.10}$$

Where each metric can be interpreted in a certain way. The mean absolute error imposes an equal weight on the spread of data, i.e. error between true and predicted output. The mean squared error puts more emphasis on extremities in the errors, they are weighted more. However, the unit of the error is also squared, for example when the unit of the actual values is m the unit of error becomes m^2, which can lead to loss of interpretability. MSE could be interpreted as a metric of variance of the error. Taking the square root of the MSE results in the RMSE, which enforces the error unit to be equal to the unit of the true and predicted values. RMSE can be interpreted as the standard deviation of the error.

4.5.2. Recurrent Neural Network

Unlike the vanilla neural network described before, a recurrent neural network (RNN) cycles information back into itself, i.e. it is possible for the network to include previous states at different times. For a high level representation of the structure see fig. 4.10. This makes RNNs suitable to detect patterns in data and perform time series forecasting [84]. For this reason RNNs are deemed a suitable candidate to do driver prediction.

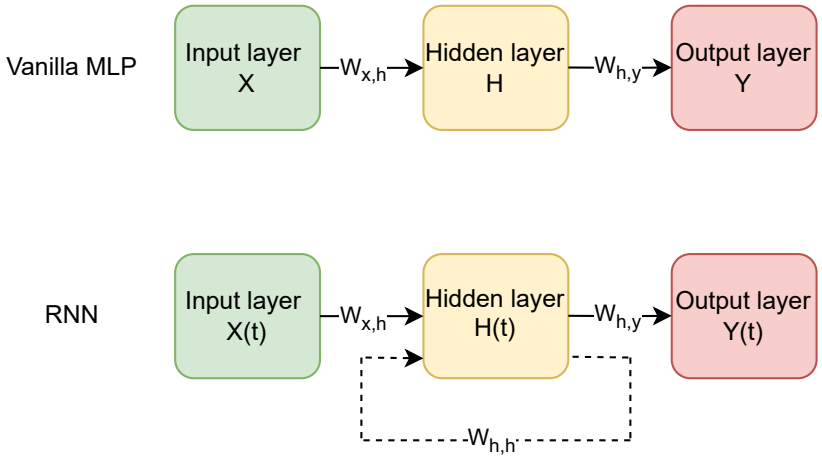


Figure 4.10: Difference of information flow between a vanilla MLP and RNN [84].

In this schematic, the hidden states are summarized into one block, as is the case for a vanilla MLP this means that a RNN can feature multiple hidden layer cells. The hidden layer equation is similar to the one presented in Equation (4.4) but with an added recurrent term [84].

$$\begin{aligned} h_t &= \rho_h(\vec{W}_{x,h} \times \vec{x}_t + \vec{W}_{h,h} \times h_{t-1} + b_x) \\ y_t &= \rho_y(\vec{W}_{h,y} \times h_t + b_y) \end{aligned} \tag{4.11}$$

With ρ denoting the activation function, often a tanh, sigmoid or Relu. The schematic of a single vanilla RNN hidden state cell is given in fig. 4.11.

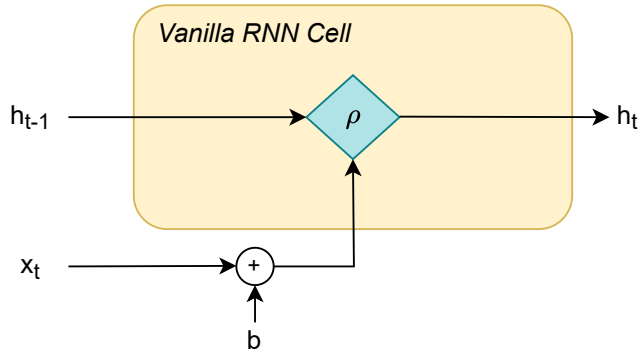


Figure 4.11: Schematic of a vanilla RNN hidden cell [84].

Since h_t is a function of h_{t-1} which in turn is a function of its preceding hidden state, one can see that each hidden state is a function of all preceding hidden states. In this way a RNN is able to ”remember” information from the past and incorporate it for making predictions in the future, a feat that a vanilla MLP does not possess. For visualizing purposes it is possible to ”unroll” the RNN presented in fig. 4.10 to get fig. 4.12.

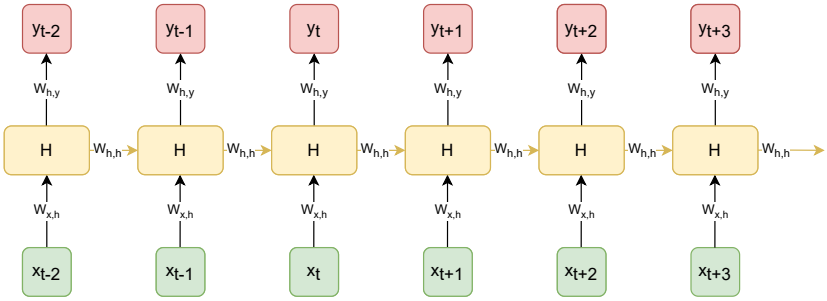


Figure 4.12: Schematic of an ”unrolled” recurrent neural network [42].

When unrolling the RNN it becomes visible that the hidden layer actually consists of multiple RNN cells, in this case denoted with ”H”, that provide information to the next cell. Essentially, unrolling a RNN means copying the cells for each time step in the input sequence, i.e. each hidden cell features the same topology and weighting matrix.
Many different structures of vanilla RNNs exist, each one being very application specific. A short summary of the different structures are given below [4].

1. **One-to-One:** A traditional neural network where only one set of inputs at time *t* is used to predict a set of outputs y at time *t*.

2. **One-to-Many:** A RNN where one input is used to generate a sequence of output. An example for which such structure could be used is image captioning, where the input is a single image and the output is a sentence description of what is going on in the picture, i.e. a list of multiple words.

3. **Many-to-One:** A RNN where a sequence of inputs is used to predict one single output. An example is sentiment classification where the input might be a sentence, i.e. list of words, and the algorithm should classify which sentiment belongs to the sentence, e.g. happy/mad/sad...

4. **Many-to-Many:** A RNN where a sequence of inputs is used to predict a sequence of outputs. Two variants of this structure exist, one where the timescale of both input and output sequence is the same (as shown in fig. 4.12), i.e. predicted output y_{t-2} is only dependent on hidden state with input x_{t-2} and previous hidden states. The other one where the sequence of outputs is shifted in time, e.g. predicted output y_{t-2} is dependent on hidden state with input x_{t+3} and previous hidden states. An example for which the first structure can be used is to do text recognition, where the network needs to classify the type of each word in a sentence, e.g. ”a beautiful mind” as input leads to ”article-adjective-noun” as output. The second structure is often used for text translation, where a full sentence is required before it can be translated.

Training

It is clear that the structure of a RNN is different than the structure of a vanilla MLP network, where the predicted output for each time step, in the prediction sequence, is a function of a sequence of its hidden state which incorporates information from sequences of past hidden states (and thus inputs). Although the structure is different, the idea of training a RNN follows the same logic as presented for a vanilla MLP. First a loss function needs to be determined, in the case of a RNN the loss function can be written as a sum of all individual loss terms, i.e. sum of each loss for each output.

$$Minimize \;\; E = \sum_{t=1}^{T} L_i(Y_t, \hat{Y}_t) \tag{4.12}$$

Where ”T” is the total amount of outputs in the output sequence, Y_t the true values at time ”t” and $\hat{Y}_t$ the predicted values at time ”t”. Having the loss function defined, it is possible to calculate the gradient of the loss with respect to the output. Then by applying the chain rule one and recursively calculating all the gradients that affect the error can be determined, in the example given in fig. 4.12 gradients with respect to three weight matrices, $W_{x,h}$, $W_{h,h}$ and $W_{h,y}$, need to be calculated. After which the weights can be updated with any of the gradient-based update rules previously discussed. Because

each of the individual loss functions is a function dependent on time, RNN optimization is usually called ”backpropagation through time (BPTT)” [84].

Long Short-Term Memory Cells

One of the main issues present with vanilla RNNs is the problem of vanishing or exploding gradients. When formulating the gradient matrix equations for each weight matrix, many weight matrix multiplications occur. If small or large values are present in the weight matrices (< 1 or > 1) the gradients can either vanish or explode when calculating the gradients backwards through the network. Either one of the results will have the effect of hampering/stopping backpropagated gradient flow through the network, making it difficult for the network to use information from multiple time steps in the past [84][48]. This topology artefact limits vanilla RNNs to be used for long time sequences.
To counteract this topology problem, the RNN cell as described in eq. (4.11) has been extended. One of these extensions is called ”long short-term memory cells”, or LSTM in short [48]. An LSTM cell has the goal of being able to learn time dependencies that go back hundreds of time steps. To achieve this goal an LSTM cell consists of a cell/memory state, a hidden state and three ”gates” that manipulate which information is passed on to the cell state. The cell state flows through all the LSTM cells present in the network. It is updated by addition and subtraction, which reduces the effect of vanishing or exploding gradients by limiting the amount of multiplications in BPTT, resulting in a structure which can memorize information from 1000 previous time steps [48]. The relevant LSTM equations are given in Equations 4.13.

$$
\begin{aligned}
f_t &= \sigma(W_{x,f} \times x_t + W_{h,f} \times h_{t-1} + b_f) \\
i_t &= \sigma(W_{x,i} \times x_t + W_{h,i} \times h_{t-1} + b_i) \\
o_t &= \sigma(W_{x,o} \times x_t + W_{h,o} \times h_{t-1} + b_o) \\
c'_t &= tanh(W_{x,c} \times x_t + W_{h,c} \times h_{t-1} + b_c) \\
c_t &= f_t \cdot c_{t-1} + i_t \cdot c'_t \\
h_t &= o_t \cdot tanh(c_t)
\end{aligned}
\tag{4.13}
$$

A schematic of an LSTM cell is given in fig. 4.13.

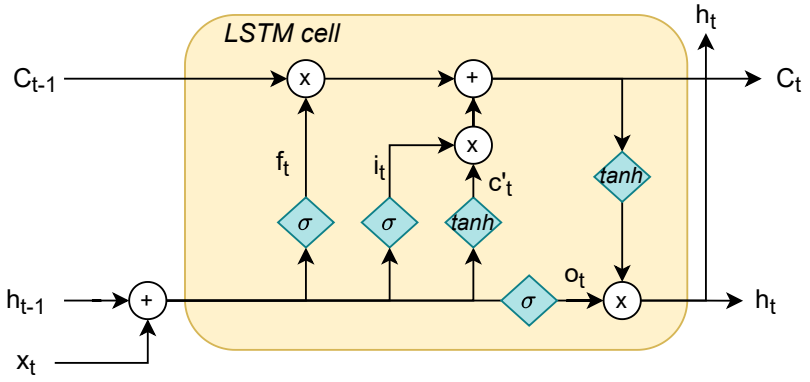


Figure 4.13: Schematic of a long short-term memory RNN cell [48].

Where σ and $tanh$ denote the sigmoid and hyperbolic tangent activation functions, and the addition and multiplication operations denote pointwise operations. In this schematic the three aforementioned gates are represented: a forget gate denoted by f_t, an input gate denoted by i_t and an output gate denoted by o_t. Each of the gates can be interpreted in the following manner.

- **Forget gate**: This gate manipulates which information from the previous cell state c_{t-1} should be maintained/forgotten. It does so by looking at the previous hidden state h_{t-1} and the input x_t, and transforming the input to a [0-1] value range with a sigmoid activation function. A ”0” implying the information should be forgotten, and a ”1” implying the information should be remembered. The output of the forget gate applies to the cell state through pointwise multiplication.

- **Input gate**: The input gate manipulates to which extent new information should be added to the cell state. It does so in a similar fashion as the forget gate, except that it consists out of two different steps. The first step is to understand which information should be added, to this extent a sigmoid is utilized which transforms the input to a [0-1] value range. A ”0” implying the information should not be added, a ”1” implying information should be added. The output is pointwise multiplied by a matrix of new candidate values c_t', which can be added to the cell state, represented by a $tanh$ transformation of the previous hidden and input state.
- **Output gate**: The output gate decides which information from the cell state is outputted by the LSTM cell. It does so by, firstly, transforming the cell state values to a value range [-1, 1] by applying a tanh activation function. Then the output gate o_t, sigmoid of h_{t-1} and x_t, decide which parts are outputted by pointwise multiplication.

Different alterations on the LSTM cell exist, such as the gated recurrent unit, which is similar in performance but computationally more efficient [18]. However, for this thesis the standard LSTM cells are utilized.

4.5.3. Encoder-Decoder Network

Encoder-decoder networks follow the same logic as the sequence-to-sequence (many-to-many) RNN model described earlier. With encoder-decoder networks, two different RNNs are required. The first RNN transforms the input sequence of possibly arbitrary length, to a fixed-dimensional vector. This network is called the encoder network. The second RNN is then used to map the fixed-dimensional vector description to the target sequence [91]. As described in [91], using vanilla RNN cells in the encoder and/or decoder works in theory, however due to temporal learning limitations (vanishing/exploding) gradients, an LSTM RNN works better for capturing long time dependencies. A representation of the encoder-decoder RNN is given in fig. 4.14.

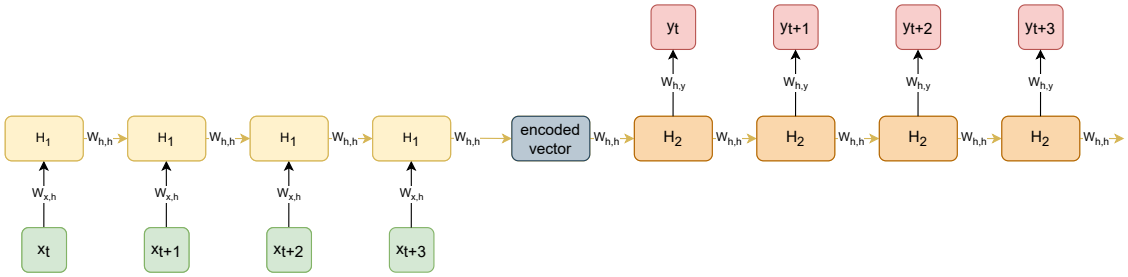


Figure 4.14: Schematic of an encoder-decoder RNN [48][91][4].

Where H_1 denotes the hidden cell of the encoder RNN, and H_2 denotes the hidden cell of the decoder RNN, both featuring LSTM cells. As with normal LSTM RNNs, the encoder and decoder network are not limited to shallow RNN networks, also deep networks can be utilized and in some cases these deep encoder-decoder networks outperform their shallow counterparts [91].

4.6. Generalization

As discussed previously, the dataset is split-up in a training, validation and test set. When a model shows very good performance on the training data, but bad performance on the test-set, it is said that the model has bad generalization. One of the reasons why this can happen is because the network tries to capture and model higher dimensional relations in the data that are due to for example noise. This is what is called overfitting. The opposite can also be true, when the model oversimplifies data dimensionality, this is what is called an underfit. Overfitting is a result of utilizing a model too complex for the problem at hand, whereas underfitting generally occurs when a too simple model is utilized.
To this extent several generalization techniques exist, these techniques make it possible to construct a complex model while reducing the risk of overfitting. One method was briefly described when discussing feature selection, namely L1-regularization, which adds a penalty to the absolute sum of weight values in the model, meaning a simpler model with less parameters/lower weights is prioritized. L2-regularization, another type of regularization, adds a penalty equal to the square of the weights [65].

A more novel approach, used in this thesis, is the regularization technique called ”dropout” [90]. When dropout is applied to layers of a network (designers choice) neurons are randomly dropped during training. This means the neurons are deactivated, i.e. weights of these neurons are set to zero, at that training iteration. Effectively this means that for each iteration a network with different structure is being trained, i.e. dropout is an efficient form of ensemble optimization. At test time the fully connected network is utilized (no dropout) with its neuron weights equal to factored-down trained weights, the factor equals the probability of dropping neurons during training. An example of dropout applied to a single layer MLP with dropout probability equal to ”p” is pictured in fig. 4.15. Due to dropout, both hidden neurons h_2 and h_3 have been deactivated during a specific training iteration.

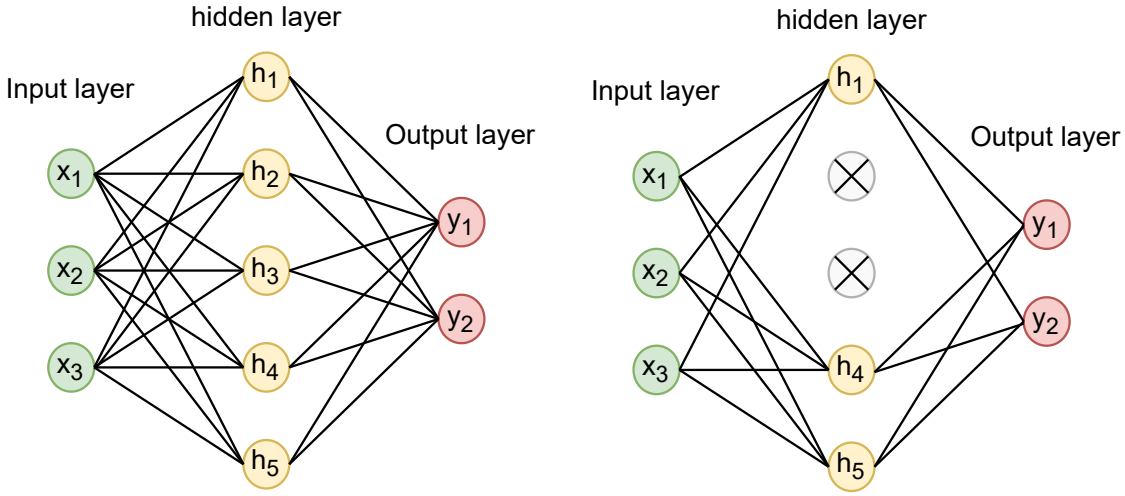


Figure 4.15: Example of dropout regularization applied to only the hidden layer of a single layer MLP [90].

4.7. Temporal Sequence Input-Output Mapping

As discussed previously the goal of the network is to predict a time sequence of all 6 vehicle DoFs for a N_p prediction horizon. To make such prediction the model requires a causal time sequence of a to-be-determined length featuring a set of input attributes. These inputs need to be fed to the three different models discussed in the previous section. It is important to understand how the data should be arranged to make sure the models work as intended. For all three models the same data structure can be used, but the implementation for the MLP network differs slightly, this difference will be explained in section 4.8 when discussing the precise network structures.
To create the input and output data structure, input and output batches need to be created. A figure of the data creation structure is presented in fig. 4.16.

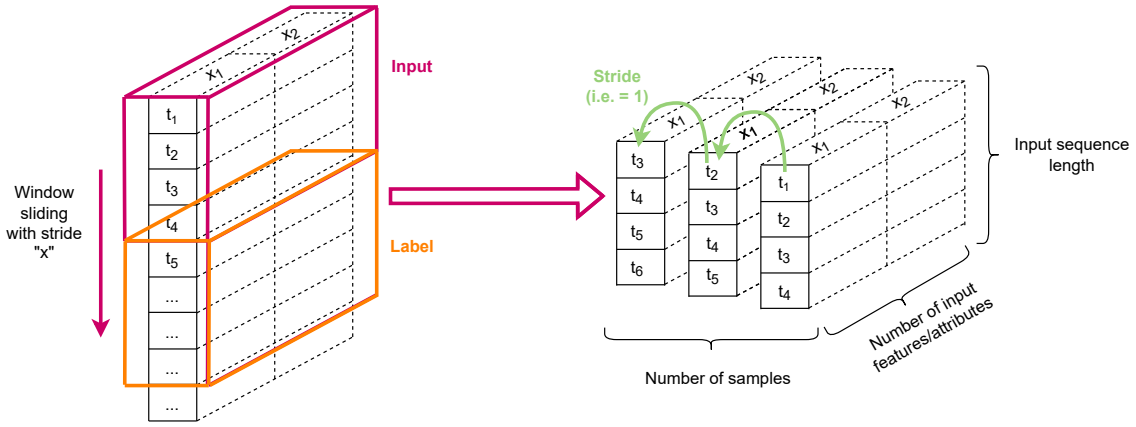


Figure 4.16: Schematic on how the data structure needs to look like such that it can be used in an LSTM RNN [36].

The left-side of the picture shows the structure of the data, a 2D-column featuring all the attributes recorded at a frequency of $100Hz$. The network requires a 3D-tensor input-output mapping, with dimensions: $[batchsize, samples, features]$. As presented in the figure, depending on the stride, i.e. how many time samples are skipped when creating the input-output mappings, one has the ability to decide how many total samples are generated. The associated formula is given in eq. (4.14).

$$Samples = floor\left(\frac{T - Window}{S}\right) + 1 \tag{4.14}$$

Where "T" equals the total amount of time samples present in the dataset, "Window" equals the sliding window length, i.e. input samples plus output samples, and "S" equals the stride. The resulting number has to be rounded down to the nearest integer number. To give an example, imagine 5 time samples and a window length of 2 with a stride of 2, the amount of input-output pairings one could make equals 2 without going out of the value range. Implementing the formula shows that indeed 2 input-output samples are created: $Samples = floor((5-2)/2) + 1 = floor(1.5) + 1 = 2$.
Applying this logic, constructing the 3D-tensor with only causal data using a sliding window is straightforward. However, as discussed in Section 4.4 also non-causal, distance-based environmental data should be represented in the input tensor. In order to cope with the causal features being time dependent and the non-causal features being distance dependent, the latter ones are digitized into "N" equal sized bins, with "N" equal to the amount of input samples. To convert the values inside each bin to a single value, the average value is calculated, except for the road curvature where the maximum road curvature in a bin is expected to be a higher quality feature than the average road curvature of that bin. An example of this process is given in fig. 4.17 where a look-ahead distance of 6 is converted to 3 equally spaced, single-value bins.

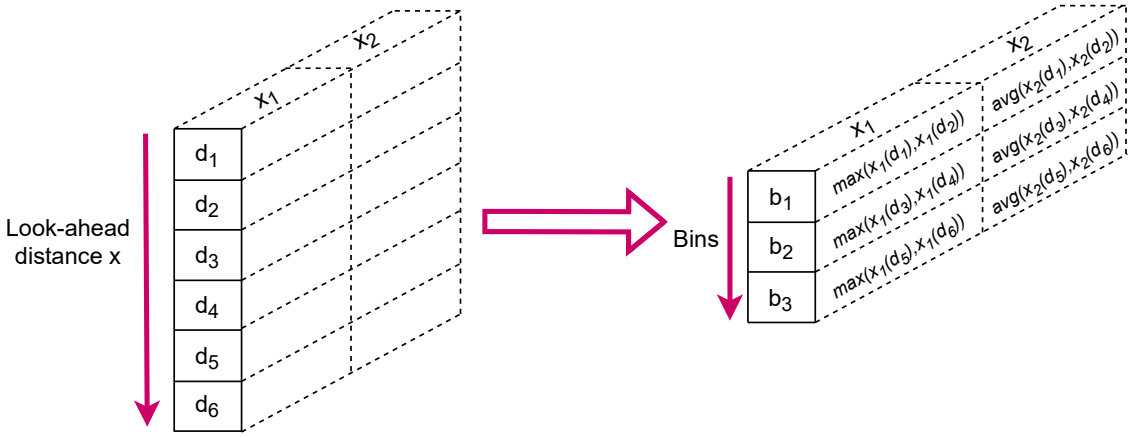


Figure 4.17: Schematic on how the look-ahead features are binned depending on the required input samples and features itself.

The resulting sequence of non-causal data points is then concatenated with the causal 3D-tensor to obtain the full 3D-tensor.

4.8. Hyperparameter Tuning

In the previous sections the global structure of the different networks were described. Also many manually adjustable parameters that affect the training performance of the model were introduced. This section serves to highlight these variables called "hyperparameters", and describe the process of finding the set of variables which ensures good model performance.
Hyperparameters can be split-up in three different categories:

1. **Structure hyperparameters**: These hyperparameters define the structure of the network. Parameters included are: amount of hidden layers, neurons per layer, neuron activation functions and for which layers regularization applies.

2. **Process hyperparameters**: These hyperparameters have a direct influence on the training pro-

cess and therefore, implicitly, an influence on the networks performance. Parameters included are: Learning rate, regularization parameters such as the dropout rate and λ in case of L1, L2-regularization, the batch size and the chosen metric for training and validation loss.

3. **Data hyperparameters**: These hyperparameters have an influence on the data, how it is structured and how the input and output sequences are devised. Parameters included are: resampling frequency, look-ahead distance, stride, scaling, way of mapping multiple values onto one single value and length of input and output sequence.

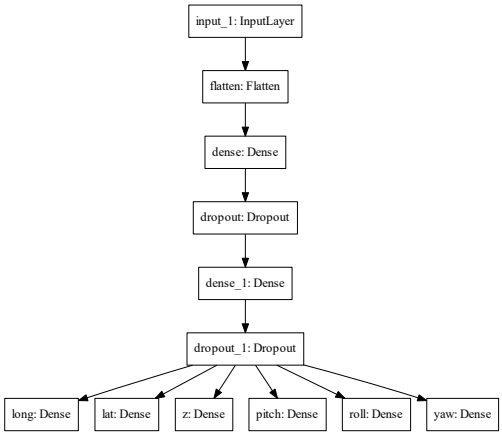
Choosing the right set of parameters is a difficult problem on its own for which many methods exist. One of the most common methods is called ”grid search”, where a search space is set-up for the different hyperparameters. Then over all the possible combinations within that search space the model is trained for a certain number of epochs, the subset of hyperparameters that gives the best model performance is selected [62]. Although this method does not scale well with the size of the search space, which is largely dependent on the amount of hyperparameters as well as the user's boundary definitions, this method was the chosen method to perform a crude hyperparameter optimization.

Only four hyperparameters were chosen for optimization, namely: the learning rate, dropout rate, batch size and amount of neurons per layer. The other hyperparameters that were not tuned, but were given a fixed value are given in table 4.4.

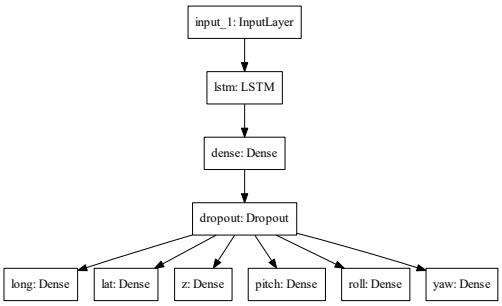
Table 4.4: Hyperparameters which are given a fixed value.

Hyperparameter	Value	Note
Structure Specific		
Number of layers MLP	2	Two hidden layers, both layers feature dropout regularization
Number of layers LSTM	1	One hidden layer of LSTM cells, the last dense output layer is required to receive an interpretable output
Number of layers Deep LSTM	3	Three hidden layer of LSTM cells, the amount of cells is independent of each other
Number of layers Enc-Dec LSTM	1	Encoder as well as decoder feature one layer of LSTM cells
MLP activation function	Relu	MLP neurons feature non-linear relu activation functions
Data Specific		
Resampling frequency	5 [Hz]	Data downsampled from 100Hz to 5Hz
Look ahead distance	100 [m]	-
Stride	4	-
Scaling	MinMax scaling	-
Road curvature mapping	Max value	-
Mapping rest of LA-features	Average	-
Length input sequence	5 [s]	5 seconds input sequence, sampled at 0.2s, means 25 input samples per feature
Length output sequence	10 [s]	*10 seconds output sequence, sampled at 0.2s means 50 output samples per feature, value chosen based on* N_p

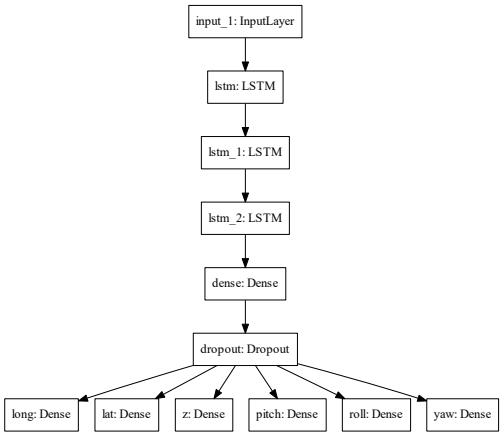
Each model features an input layer and an output layer which consists of a small dense MLP for each separate output channel with the amount of neurons equal to N_p. The dense output layers feature a linear activation function without bias and weights, i.e. $f(x) = x$. A figure of the network structures can be found in fig. 4.18. The two hidden layer MLP, found in fig. 4.18a, features a ”flatten layer” which maps the 3D-input tensor into a 2D-tensor with dimensions $[batchsize, inputseq \times \#features]$.



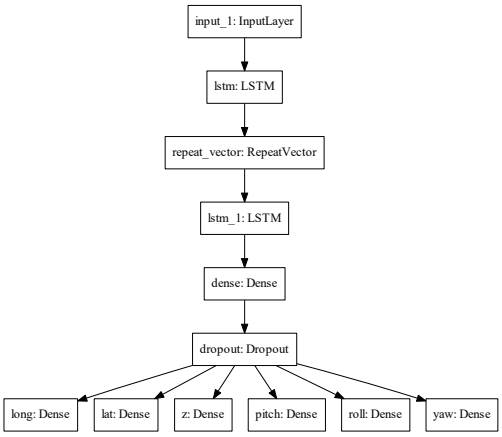
(a) Two hidden layer dense MLP network with dropout.



(b) Single layer LSTM RNN.



(c) Three layer deep LSTM RNN.



(d) Single LSTM encoder-decoder RNN.

Figure 4.18: Four different supervised neural network structures used for driver behavior regression.

Grid search was performed for the value ranges given in table 4.5. The process is as follows, first a crude search is performed for both the learning rate and the dropout rate, i.e. the value range is spanned five evenly spaced values with relatively large intervals. After which a more detailed value range is specified around the best performing value. This more detailed value range is then also spanned by five equal spaced values. For both the batch size as well as the amount of neurons, it is common to use integers equal to a power of 2, e.g. value range of [16-64] is spanned by the values [16, 32, 64]. It was assumed that the optimization of the hyperparameters is independent [62], therefore the optimization process is performed for each parameter independently. First the learning rate is optimized, then the dropout rate and the batch size are optimized, lastly the amount of neurons per layer are optimized. For all three steps, the network is trained for 15 epochs (after this point the training slows down significantly). Step 1 and 2 are only performed for the two-layer MLP and are assumed to give good cross-over performance for the other three models, only the amount of neurons/cells are assumed to be model dependent. Since the weight initialization is random (Glorot uniform distributed), the performance after 15 epochs is not static, i.e. training a neural network with the same hyperparameters multiple times will give different results. Normal conduct would be to train the network a couple of times and average out the performance, however, due to time constraints this step is not performed. As such no guarantees exist that the selected hyperparameters are truly optimal.

Table 4.5: Grid search value ranges used for optimization of selected hyperparameters.

Hyperparameter	Initial value range
Learning rate	$[5e^{-4} - 1e^{-2}]$
Dropout rate	$[0.3 - 0.7]$
Batch size	$[16 - 128]$
Neurons MLP	$[128 - 1024]$
Cells LSTM	$[64 - 512]$
Cells Deep-LSTM	$[64 - 512]$
Cells Enc-Dec LSTM	$[64 - 512]$

The results of the grid search based learning rate optimization process can be found in table 4.6. The light orange highlighted rows are the rows that showed the best average performance based on the MAE metric, around which a more accurate value range was based in the following runs. In this case the MAE metric was calculated on the unscaled test data, as such it features the units m/s^2 and rad/s for both the translational and rotational directions respectively.
Run 1 is the run with the settings as described in table 4.5. Run 2, is described as the more accurate grid search for a $lr \in [0.005 - 0.015]$, run 3 depicts the grid search run for a $lr \in [0.0003 - 0.0015]$. When comparing the error performance between run 2 and 3, the performance of run 2 was found to be slightly better, therefore it was decided to perform a last, slightly more narrow, run for a $lr \in [0.0095 - 0.015]$. In this final run, the learning rate, $lr = 0.0136$, was found to be best performing over all runs.

Table 4.6: Results of a grid search based learning rate optimization for a two hidden layer MLP network.

Run	Learning rate	Batchsize	Neurons layer 1	Neurons layer 2	Dropout rate	MAE Long acc	MAE Lat acc	MAE Z-acc	MAE Pitch rate	MAE Roll rate	MAE Yaw rate	Avg loss translational $[m/s^2]$	Avg loss rotational $[rad/s]$
1	*0.0005*	*64*	*512*	*256*	*0.5*	*0.3818*	*0.6917*	*0.0670*	*0.0154*	*0.0080*	*0.0343*	*0.3802*	*0.0192*
1	*0.002875*	*64*	*512*	*256*	*0.5*	*0.3853*	*0.6976*	*0.0671*	*0.0154*	*0.0080*	*0.0349*	*0.3833*	*0.0194*
1	*0.00525*	*64*	*512*	*256*	*0.5*	*0.4026*	*0.7111*	*0.0666*	*0.0153*	*0.0080*	*0.0368*	*0.3934*	*0.0200*
1	*0.007625*	*64*	*512*	*256*	*0.5*	*0.3924*	*0.7009*	*0.0665*	*0.0155*	*0.0079*	*0.0346*	*0.3866*	*0.0193*
1	*0.01*	*64*	*512*	*256*	*0.5*	*0.3905*	*0.6752*	*0.0653*	*0.0155*	*0.0080*	*0.0326*	*0.3770*	*0.0187*
2	*0.0050*	*64*	*512*	*256*	*0.5*	*0.3974*	*0.6953*	*0.0663*	*0.0152*	*0.0079*	*0.0341*	*0.3863*	*0.0191*
2	*0.0075*	*64*	*512*	*256*	*0.5*	*0.3927*	*0.7084*	*0.0663*	*0.0154*	*0.0079*	*0.0351*	*0.3891*	*0.0195*
2	*0.0100*	*64*	*512*	*256*	*0.5*	*0.3905*	*0.6673*	*0.0650*	*0.0153*	*0.0081*	*0.0311*	*0.3743*	*0.0182*
2	*0.0125*	*64*	*512*	*256*	*0.5*	*0.4111*	*0.7023*	*0.0666*	*0.0154*	*0.0080*	*0.0354*	*0.3933*	*0.0196*
2	*0.0150*	*64*	*512*	*256*	*0.5*	*0.4065*	*0.7049*	*0.0665*	*0.0154*	*0.0080*	*0.0365*	*0.3926*	*0.0200*
3	*0.0003*	*64*	*512*	*256*	*0.5*	*0.3803*	*0.6918*	*0.0669*	*0.0154*	*0.0079*	*0.0338*	*0.3796*	*0.0190*
3	*0.0006*	*64*	*512*	*256*	*0.5*	*0.3932*	*0.6968*	*0.0665*	*0.0154*	*0.0079*	*0.0344*	*0.3855*	*0.0192*
3	*0.0009*	*64*	*512*	*256*	*0.5*	*0.3844*	*0.7186*	*0.0670*	*0.0155*	*0.0080*	*0.0357*	*0.3900*	*0.0198*
3	*0.0012*	*64*	*512*	*256*	*0.5*	*0.3961*	*0.7054*	*0.0665*	*0.0153*	*0.0079*	*0.0350*	*0.3893*	*0.0194*
3	*0.0015*	*64*	*512*	*256*	*0.5*	*0.3911*	*0.6920*	*0.0670*	*0.0154*	*0.0079*	*0.0345*	*0.3834*	*0.0193*
4	*0.0095*	*64*	*512*	*256*	*0.5*	*0.4048*	*0.6574*	*0.0672*	*0.0154*	*0.0080*	*0.0320*	*0.3765*	*0.0185*
4	*0.010875*	*64*	*512*	*256*	*0.5*	*0.3913*	*0.7010*	*0.0662*	*0.0153*	*0.0080*	*0.0349*	*0.3862*	*0.0194*
4	*0.01225*	*64*	*512*	*256*	*0.5*	*0.3965*	*0.7021*	*0.0664*	*0.0154*	*0.0079*	*0.0354*	*0.3883*	*0.0196*
4	*0.013625*	*64*	*512*	*256*	*0.5*	*0.3848*	*0.6567*	*0.0674*	*0.0153*	*0.0080*	*0.0332*	*0.3696*	*0.0188*
4	*0.015*	*64*	*512*	*256*	*0.5*	*0.4015*	*0.6983*	*0.0666*	*0.0153*	*0.0080*	*0.0343*	*0.3888*	*0.0192*

The next step is to perform the grid search for both the batchsize as well as the dropout rate. These two are combined into a single optimization step. Looking at table 4.7 it can be seen that the best performance is always found when using a batchsize equal to 128. Concurrently, the table would suggest a dropout rate of 0.3 to be best performing, but from the previous result found in table 4.6, one can conclude that a dropout rate equal to 0.5 is also effective. Therefore, a second iteration is required, setting the batchsize at 128 and having a more elaborate search for the dropout rate in the range $[0.3 - 0.7]$ using five evenly spaced values.

Table 4.7: Results of a grid search based batchsize and dropout rate optimization for a two hidden layer MLP network.

Run	Learning rate	Batchsize	Neurons layer 1	Neurons layer 2	Dropout rate	MAE Long acc	MAE Lat acc	MAE Z-acc	MAE Pitch rate	MAE Roll rate	MAE Yaw rate	Avg loss translational m/s^2	Avg loss rotational rad/s
0	0.0136	16	512	256	0.3	0.3917	0.7274	0.0668	0.0156	0.0081	0.0350	0.3953	0.0196
1	0.0136	32	512	256	0.3	0.4026	0.6934	0.0681	0.0153	0.0080	0.0357	0.3881	0.0197
2	0.0136	64	512	256	0.3	0.3790	0.6899	0.0635	0.0154	0.0080	0.0362	0.3775	0.0198
3	0.0136	128	512	256	0.3	0.3925	0.6501	0.0639	0.0152	0.0080	0.0322	0.3689	0.0184
4	0.0136	16	512	256	0.5	0.4335	0.7339	0.0671	0.0157	0.0081	0.0355	0.4115	0.0198
5	0.0136	32	512	256	0.5	0.4052	0.7226	0.0680	0.0154	0.0080	0.0370	0.3986	0.0201
6	0.0136	64	512	256	0.5	0.4033	0.6970	0.0663	0.0153	0.0080	0.0344	0.3888	0.0192
7	0.0136	128	512	256	0.5	0.3921	0.6630	0.0662	0.0154	0.0080	0.0324	0.3738	0.0186
8	0.0136	16	512	256	0.7	0.4156	0.7385	0.0667	0.0160	0.0080	0.0404	0.4069	0.0215
9	0.0136	32	512	256	0.7	0.4017	0.6982	0.0694	0.0158	0.0082	0.0345	0.3898	0.0195
10	0.0136	64	512	256	0.7	0.4034	0.6925	0.0662	0.0154	0.0079	0.0339	0.3874	0.0191
11	0.0136	128	512	256	0.7	0.3984	0.6923	0.0665	0.0153	0.0079	0.0339	0.3857	0.0190

Table 4.8 shows the re-iteration on the dropout grid search, as one can see not very notable differences can be found between dropout rates in the range of $[0.3 - 0.5]$. The original paper [90] suggests that a dropout rate of 0.5 generally gives good performance, therefore this value is chosen for future use.

Table 4.8: Results of a grid search based dropout optimization for a two hidden layer MLP network.

Run	Learning rate	Batchsize	Neurons layer 1	Neurons layer 2	Dropout rate	MAE Long acc	MAE Lat acc	MAE Z-acc	MAE Pitch rate	MAE Roll rate	MAE Yaw rate	Avg loss translational m/s^2	Avg loss rotational rad/s
0	0.0136	128	512	256	0.3	0.3976	0.6678	0.0673	0.0152	0.0080	0.0324	0.3775	0.0185
1	0.0136	128	512	256	0.4	0.3906	0.6604	0.0649	0.0152	0.0080	0.0321	0.3720	0.0184
2	0.0136	128	512	256	0.5	0.3843	0.6719	0.0665	0.0153	0.0079	0.0325	0.3742	0.0186
3	0.0136	128	512	256	0.6	0.3943	0.7000	0.0667	0.0153	0.0080	0.0351	0.3870	0.0194
4	0.0136	128	512	256	0.7	0.3956	0.6949	0.0666	0.0152	0.0079	0.0341	0.3857	0.0191

Table 4.9 shows the grid search results for the neuron and cell density optimization. The highlighted rows present the models with the best performance. The MLP featuring 512 neurons in the first, and 256 neurons in the second layer shows the best performance as can be seen by the average losses. From this result it was decided to keep the neuron density for the second dense layer constant throughout all the other networks. The one layer LSTM with 128 and 256 LSTM cells do not show much difference in performance, at a further stage both of them will be trained for more epochs, the one showing the best results will be used. For the deeo LSTM network, a notable difference can be seen between 64 LSTM cell layers and higher cell density layers for the deep LSTM network. The encoder-decoder network seems to perform best when 128 LSTM cells are used for both the encoder and decoder model.

Table 4.9: Results of a grid search based neuron/cell density optimization for a two hidden layer MLP, an LSTM RNN, a deep LSTM RNN and a single layer encoder-decoder LSTM network.

Run	Learning rate	Batchsize	Neurons layer 1	Neurons layer 2	Dropout rate	MAE Long acc	MAE Lat acc	MAE Z-acc	MAE Pitch rate	MAE Roll rate	MAE Yaw rate	Avg loss translational m/s^2	Avg loss rotational rad/s
Dense Network	0.0136	128	256	128	0.5	0.391	0.696	0.067	0.015	0.008	0.035	0.385	0.019
	0.0136	128	256	256	0.5	0.396	0.681	0.066	0.015	0.008	0.033	0.381	0.019
	0.0136	128	256	512	0.5	0.392	0.708	0.067	0.015	0.008	0.036	0.389	0.020
	0.0136	128	512	128	0.5	0.395	0.690	0.066	0.015	0.008	0.034	0.384	0.019
	0.0136	128	512	256	0.5	0.386	0.686	0.066	0.015	0.008	0.033	0.379	0.019
	0.0136	128	512	512	0.5	0.384	0.812	0.067	0.015	0.008	0.042	0.421	0.022
	0.0136	128	1024	128	0.5	0.396	0.708	0.067	0.015	0.008	0.036	0.390	0.020
	0.0136	128	1024	256	0.5	0.391	0.699	0.066	0.015	0.008	0.035	0.385	0.019
	0.0136	128	1024	512	0.5	0.405	0.811	0.067	0.016	0.008	0.043	0.428	0.022
LSTM Network	0.0136	128	64	256	0.5	0.364	0.575	0.050	0.015	0.008	0.030	0.330	0.018
	0.0136	128	128	256	0.5	0.359	0.576	0.047	0.015	0.007	0.029	0.327	0.017
	0.0136	128	256	256	0.5	0.365	0.555	0.054	0.015	0.008	0.028	0.325	0.017
	0.0136	128	512	256	0.5	0.393	0.694	0.066	0.015	0.008	0.034	0.384	0.019
Deep LSTM Network	0.0136	128	64	256	0.5	0.386	0.571	0.068	0.015	0.008	0.026	0.341	0.016
	0.0136	128	128	256	0.5	0.396	0.703	0.066	0.015	0.008	0.036	0.388	0.020
	0.0136	128	256	256	0.5	0.392	0.696	0.066	0.015	0.008	0.034	0.385	0.019
	0.0136	128	512	256	0.5	0.395	0.698	0.067	0.015	0.008	0.035	0.387	0.019
Enc-Dec LSTM Network	0.0136	128	64	256	0.5	0.410	0.684	0.064	0.015	0.008	0.035	0.386	0.020
	0.0136	128	128	256	0.5	0.418	0.632	0.066	0.015	0.008	0.033	0.372	0.019
	0.0136	128	256	256	0.5	0.392	0.696	0.066	0.015	0.008	0.035	0.385	0.019
	0.0136	128	512	256	0.5	0.399	0.698	0.066	0.015	0.008	0.035	0.388	0.019

A final note should be made to the results presented in this section. When looking at the amount of parameters that need to be optimized, e.g. for the three layer deep LSTM network see table 4.10, one can see that the amount of optimization parameters does not scale linearly with cell density. Since

networks with more parameters take longer to train, it is possible that a bias exists towards less complex models using only 15 training epochs. These models tend to train faster, but are also more limited in their performance. In other words, these results do not necessarily indicate the true potential of each of the networks. This could be resolved by training for more epochs.

Table 4.10: Variation of optimization parameters as a function of the cell density for a three layer deep LSTM network.

Deep LSTM Network	
Cell density	**Optimization parameters**
64	*111 468*
128	*394 668*
256	*1 477 164*
512	*5 706 540*

Table 4.11 shows the results of the same grid search done for 50 epochs instead of 15 for both the deep LSTM network and encoder-decoder network. One interesting result is that the performance of the deep LSTM sees a sharp decline in performance when increasing the cell density from 128 to 256. One of the reasons for this to occur could be found in the fact that no regularization in the LSTM layer takes place, as such the model overfits on the training data and shows bad performance on the independent test data. On another interesting note, increasing the number of epochs from 15 to 50 increases the performance of both the 64 and 128 cell LSTM network significantly, with the larger network outperforming the smaller one. The difference between the two can be explained by the increased performance on the lateral acceleration prediction, $0.41 rad/s$ vs $0.46 rad/s$ MAE.
As with the deep LSTM, the encoder-decoder network seems to suffer from increasing cell density. However, in this case it occurs gradually with every density increase, and not after a certain threshold is reached. Two reasons could lay at the foundation of this effect. The first one being that this model structure seems to overfit more easily. However, this type of network structure, inherently, features less parameters than for example the deep LSTM network, 78.444 for a 64 cell and $3.607.340$ for a 512 cell enc-dec structure. So overfitting due to complexity does not necessarily seem to support this idea. Another idea might be the following, researchers [91] have found that, when using an encoder-decoder structure for phrase translation, reversing the order of the input phrase positively contributed to the prediction quality. It was argued that when the beginning of the input sentence, i.e. when reversed these are the last inputs to the encoder, lies closer to the beginning of the target sentence, i.e. in normal order these are predicted by the first outputs of the decoder, a stronger correlation between parts of the sentence is created. The reason why this occurs, is because the backwards gradient flow diminishes over the amount of cells present in the network. This phenomenon could very well be seen here as well. When one assumes that a strong correlation exists between the upcoming road conditions and one's future driving behavior, having them further apart could cause lower model performance. This theory could be tested out by reversing the order of the look-ahead features and checking whether the performance of the encoder-decoder network increases.

Table 4.11: Results of extended grid search for neuron/cell density optimization for deep and encoder-decoder LSTM with 50 training epochs.

Run	Learning Rate	Batchsize	Neurons layer 1	Neurons layer 2	Dropout rate	MAE Long acc	MAE Lat acc	MAE Z-acc	MAE Pitch rate	MAE Roll rate	MAE Yaw rate	Avg loss translational m/s^2	Avg loss rotational rad/s
Deep LSTM Network	*0.0136*	*128*	*64*	*256*	*0.5*	*0.3685*	*0.4607*	*0.0381*	*0.0155*	*0.0071*	*0.0215*	*0.2891*	*0.0147*
	0.0136	*128*	*128*	*256*	*0.5*	*0.3274*	*0.4117*	*0.0315*	*0.0152*	*0.0068*	*0.0196*	*0.2569*	*0.0139*
	0.0136	*128*	*256*	*256*	*0.5*	*0.3940*	*0.7133*	*0.0668*	*0.0153*	*0.0080*	*0.0361*	*0.3913*	*0.0198*
	0.0136	*128*	*512*	*256*	*0.5*	*0.3917*	*0.6941*	*0.0663*	*0.0154*	*0.0079*	*0.0346*	*0.3840*	*0.0193*
Enc-Dec LSTM Network	*0.0136*	*128*	*64*	*256*	*0.5*	*0.3723*	*0.4282*	*0.0349*	*0.0152*	*0.0068*	*0.0215*	*0.2785*	*0.0145*
	0.0136	*128*	*128*	*256*	*0.5*	*0.3795*	*0.5043*	*0.0449*	*0.0150*	*0.0076*	*0.0235*	*0.3096*	*0.0153*
	0.0136	*128*	*256*	*256*	*0.5*	*0.3927*	*0.6656*	*0.0605*	*0.0152*	*0.0079*	*0.0321*	*0.3729*	*0.0184*
	0.0136	*128*	*512*	*256*	*0.5*	*0.3980*	*0.6920*	*0.0666*	*0.0153*	*0.0079*	*0.0340*	*0.3855*	*0.0191*

Based on the results presented above, the hyperparameter values for the different network structures presented in table 4.12 are used for training.

Table 4.12: Table containing final hyperparameter values for a dense MLP, single layer LSTM, deep LSTM and encoder-decoder LSTM network.

	Learning rate	Batchsize	Dropout rate	Neurons layer 1	Neurons layer 2
Two hidden layer MLP network	*0.0136*	*128*	*0.5*	*512*	*256*
Single layer LSTM network	*0.0136*	*128*	*0.5*	*256*	*256*
Three layer LSTM network	*0.0136*	*128*	*0.5*	*128*	*256*
Single layer enc-dec network	*0.0136*	*128*	*0.5*	*64*	*256*

4.9. Tuned Model: Performance Analysis

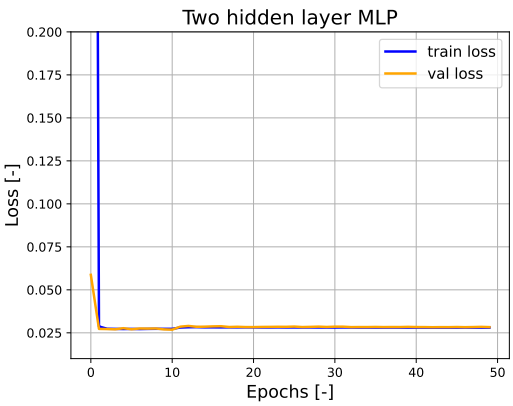
This section serves to analyze the performance of the different models. First the models are trained for a certain amount of epochs. After which the performance of the models is analyzed in the following three ways:

1. MSE and MAE computed for the unscaled data for all 6 DoFs and compared to the MSE and MAE when using a constant prediction. The result is an average error value over all predictions.
2. MSE and MAE computed for each time trace prediction in the sequence, and compared to the MSE and MAE when using a constant prediction. The result is an error over time curve.
3. Samples of the time trace predictions are plotted and analyzed to investigate model behavior.

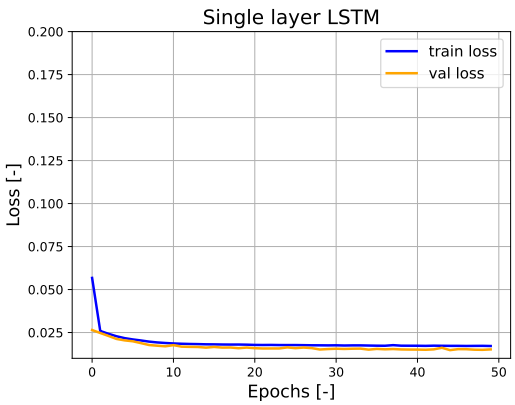
The reason why the model performance is compared to the constant prediction is because this was used in a previous study [7][26][28][31].Therefore, it is used as performance benchmark.

Learning curve

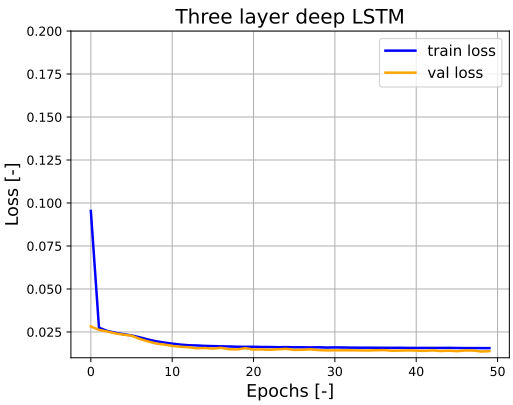
First it needs to be checked how many epochs are required, to do so each of the models is trained for 50 epochs. Depending on how quick the models are trained, 50 epochs are deemed to be sufficient. The learning curve for all four models with parameters equal to table 4.12 are given in fig. 4.19.



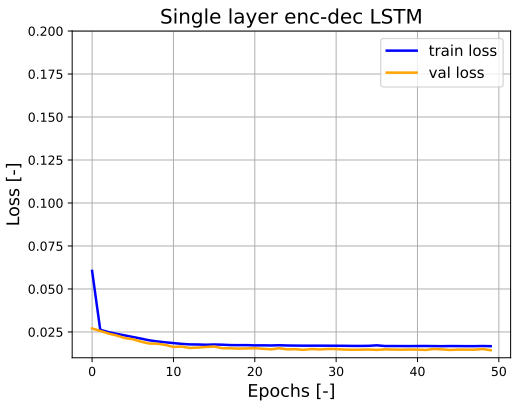
(a) Two hidden layer MLP loss curve.



(b) Single layer LSTM loss curve.



(c) Three layer deep LSTM loss curve.



(d) Single layer enc-dec LSTM loss curve.

Figure 4.19: Loss curves for a single layer MLP, single layer LSTM, three layer deep LSTM, and single layer enc-dec LSTM trained for 50 epochs.

One aspect immediately stands out in these plots, namely that the loss drops significantly after only one iteration, after which it gradually decreases or asymptotes very quickly. This effect is most visible for the MLP model, one reason could be that this occurs because the outputs of the MLP are assumed to be independent of one-another, as such it is able to learn several high-level relations, but cannot learn deep-rooted time dependencies. Therefore, the rate at which the loss decreases stagnates after the first three epochs. This could explain why this effect is less pronounced for the other three models that are able to capture time dependencies. It could also be due to the fact that the MLP uses Relu activation functions with a too high learning rate, causing dead Relu's in the network. This also explains why the RNN-based networks stop learning after several epochs as they also feature a dense layer with Relu activation. Another theory could be that this is a data issue. Because only one rural road section is used (both directions), it is possible that the model learns the majority of the input-output relation on this specific route quickly (within a few epochs). This theory could be verified by introducing a more widespread dataset containing multiple rural road segments. However, this artefact could also be a result from multiple, different aspects and not limited to the three options given above. Therefore, this effect should be investigated further at a later stage.
What can be concluded from these four loss plots is that training the models for 50 epochs is sufficient.

Model performance vs constant prediction

In table 4.13 the average MSE and MAE results of the constant prediction as well as the trained networks are presented. The loss on the unscaled prediction data is calculated for each prediction sequence of 50 samples, after which the average of the individual losses is calculated. For each of

the models, and their predicted DoFs, a percentage performance increase (percentage drop in error) compared to the constant prediction is given as well.

Table 4.13: Trained network average MSE and MAE scores per predicted DoF, as well as the percentage decrease compared to the average MSE and MAE scores of a constant prediction.

		MSE SN	MAE SN	% MSE SN	% MAE SN	MSE NS	MAE NS	% MSE NS	% MAE NS
Constant Prediction	*Long Acc*	*5.73E-01*	*4.94E-01*	-	-	*6.23E-01*	*4.72E-01*	-	-
	Lat Acc	*1.92E+00*	*9.53E-01*	-	-	*1.37E+00*	*7.81E-01*	-	-
	Z Acc	*1.10E-02*	*6.54E-02*	-	-	*8.39E-03*	*5.58E-02*	-	-
	Pitch	*1.48E-04*	*7.83E-03*	-	-	*2.62E-04*	*1.05E-02*	-	-
	Roll	*6.79E-04*	*1.78E-02*	-	-	*7.49E-04*	*1.89E-02*	-	-
	Yaw	*1.12E-02*	*6.56E-02*	-	-	*9.22E-03*	*5.65E-02*	-	-
Dense Network	*Long Acc*	*4.38E-01*	*4.30E-01*	*-23.6%*	*-13.0%*	*4.75E-01*	*4.16E-01*	*-23.8%*	*-11.9%*
	Lat Acc	*1.24E+00*	*7.87E-01*	*-35.4%*	*-17.4%*	*8.84E-01*	*6.56E-01*	*-35.5%*	*-16.0%*
	Z Acc	*6.89E-03*	*5.65E-02*	*-37.4%*	*-13.6%*	*5.81E-03*	*5.11E-02*	*-30.8%*	*-8.4%*
	Pitch	*8.09E-05*	*5.91E-03*	*-45.3%*	*-24.5%*	*1.39E-04*	*7.78E-03*	*-46.9%*	*-25.9%*
	Roll	*3.44E-04*	*1.24E-02*	*-49.3%*	*-30.3%*	*3.78E-04*	*1.33E-02*	*-49.5%*	*-29.6%*
	Yaw	*7.55E-03*	*5.29E-02*	*-32.6%*	*-19.4%*	*5.96E-03*	*4.71E-02*	*-35.4%*	*-16.6%*
LSTM Network	*Long Acc*	*2.74E-01*	*3.49E-01*	*-52.2%*	*-29.4%*	*3.08E-01*	*3.68E-01*	*-50.6%*	*-22.0%*
	Lat Acc	*4.13E-01*	*4.50E-01*	*-78.5%*	*-52.8%*	*2.79E-01*	*3.74E-01*	*-79.6%*	*-52.1%*
	Z Acc	*1.70E-03*	*2.86E-02*	*-84.5%*	*-56.3%*	*1.44E-03*	*2.59E-02*	*-82.8%*	*-53.6%*
	Pitch	*6.70E-05*	*4.89E-03*	*-54.7%*	*-37.5%*	*1.22E-04*	*6.95E-03*	*-53.4%*	*-33.8%*
	Roll	*3.39E-04*	*1.24E-02*	*-50.1%*	*-30.3%*	*3.72E-04*	*1.33E-02*	*-50.3%*	*-29.6%*
	Yaw	*2.59E-03*	*2.95E-02*	*-76.9%*	*-55.0%*	*1.92E-03*	*2.64E-02*	*-79.2%*	*-53.3%*
Deep LSTM Network	*Long Acc*	*2.28E-01*	*3.21E-01*	*-60.2%*	*-35.0%*	*3.08E-01*	*3.70E-01*	*-50.6%*	*-21.6%*
	Lat Acc	*2.94E-01*	*3.78E-01*	*-84.7%*	*-60.3%*	*2.11E-01*	*3.21E-01*	*-84.6%*	*-58.9%*
	Z Acc	*1.20E-03*	*2.41E-02*	*-89.1%*	*-63.1%*	*1.16E-03*	*2.33E-02*	*-86.2%*	*-58.2%*
	Pitch	*6.51E-05*	*4.75E-03*	*-56.0%*	*-39.3%*	*1.21E-04*	*6.86E-03*	*-53.8%*	*-34.7%*
	Roll	*3.31E-04*	*1.22E-02*	*-51.3%*	*-31.5%*	*3.64E-04*	*1.31E-02*	*-51.4%*	*-30.7%*
	Yaw	*2.01E-03*	*2.49E-02*	*-82.1%*	*-62.0%*	*1.52E-03*	*2.20E-02*	*-83.5%*	*-61.1%*
Enc-Dec LSTM Network	*Long Acc*	*2.44E-01*	*3.35E-01*	*-57.4%*	*-32.2%*	*2.95E-01*	*3.67E-01*	*-52.6%*	*-22.2%*
	Lat Acc	*3.20E-01*	*3.97E-01*	*-83.3%*	*-58.3%*	*2.24E-01*	*3.32E-01*	*-83.6%*	*-57.5%*
	Z Acc	*1.30E-03*	*2.49E-02*	*-88.2%*	*-61.9%*	*1.51E-03*	*2.52E-02*	*-82.0%*	*-54.8%*
	Pitch	*6.58E-05*	*4.81E-03*	*-55.5%*	*-38.6%*	*1.23E-04*	*6.96E-03*	*-53.1%*	*-33.7%*
	Roll	*3.35E-04*	*1.23E-02*	*-50.7%*	*-30.9%*	*3.68E-04*	*1.32E-02*	*-50.9%*	*-30.2%*
	Yaw	*2.30E-03*	*2.75E-02*	*-79.5%*	*-58.1%*	*1.66E-03*	*2.39E-02*	*-82.0%*	*-57.7%*

One important note is that, the absolute error values that are presented are the unscaled, non-normalized errors. This means that they are relative to their respective unit, meaning acceleration MAE and MSE are presented in $[m/s^2]$ and $[m^2/s^4]$ whereas the rotational rate errors are presented in $[rad/s]$ and $[rad^2/s^2]$ for MAE and MSE respectively.
When analyzing the results some interesting points can be deduced. First of all it can be observed that all trained networks provide an improvement on both the MSE as well as the MAE metric when compared to the constant prediction. The simplest network, the double layer dense network, has a drop in MSE for the test drive in SN-direction of roughly $[20\% - 40\%]$ and in MAE of roughly $[13\% - 17\%]$ for its acceleration predictions. In contrast, the RNN-based networks show a drop of roughly $[50\% - 90\%]$ for MSE and roughly $[30\% - 63\%]$ for MAE for its acceleration predictions. The best performing network is the deep LSTM network, which shows drops in error marginally better than the encoder-decoder network. It can be observed that, for each of the networks, it applies that the error drop is most significant for the acceleration in Z, then the lateral and lastly the longitudinal acceleration. When looking at the acceleration traces given in fig. 4.5 and fig. 4.6, it can be observed that the lateral acceleration features more and larger amplitude peaks ($[-7.5m/s^2 - 7.5m/s^2]$) with respect to the longitudinal acceleration ($[-7m/s^2 - 4m/s^2]$). This would mean that a constant reference would be a worse prediction in case of lateral acceleration. This observation is confirmed when comparing the MSE and MAE scores of the constant prediction for both acceleration directions. The average MSE for lateral acceleration in NS- and SN-direction ($= 1.65m^2/s^4$) is roughly 2.8 times as large as the average MSE for longitudinal acceleration ($= 0.6m^2/s^4$). The average MAE for the lateral acceleration ($= 0.87m/s^2$) is 1.8 times larger than the average MAE for longitudinal acceleration ($= 0.48m/s^2$). This difference in metric score also shows that the larger errors present in the prediction are indeed more prevalent as large predictions errors are scaled quadratically. With this in mind it is also interesting to see that all the models feature a larger decrease in MSE compared to MAE, which effectively means the model predictions feature less extremities in error compared to the constant prediction.

When looking at the performance on rotational rate, it can be seen that the drop in MSE and MAE is most significant for the yaw rate, which is expected because a car has more freedom in yaw-direction than in pitch- or roll-direction. This is confirmed when looking at the average yaw rate MSE ($= 1.02e-2 rad^2/s^2$) error of the constant prediction, which is roughly 15 times larger than the average roll rate MSE ($= 7.14e-4 rad^2/s^2$). The performance on pitch and roll rate prediction sees a drop in MSE of $\approx [45\% - 50\%]$ and a drop in MAE of $\approx [25\% - 30\%]$ for the simplest model and a larger drop of $\approx [50\% - 55\%]$ in MSE and of $\approx [30\% - 40\%]$ in MAE for the RNN structures in SN-direction. The performance on the yaw rate prediction sees a $\approx 32\%$ decrease in MSE and $\approx 20\%$ in MAE for the simplest model and a larger decrease of $\approx [77\% - 82\%]$ in MSE and $\approx [55\% - 62\%]$ in MAE for the RNN structures in SN-direction. As is the case for the accelerations, the deep LSTM also seems to perform best for rotational rate prediction.
As presented in section 4.3, the data consisted of 24 drives in SN- and 13 drives in NS-direction. Due to the bias towards the SN-direction it would have been expected for the models to perform better in SN-direction. When looking at the longitudinal acceleration prediction this is indeed the case for each of the RNN structured networks, where the absolute MSE and MAE scores are roughly $[12\% - 35\%]$ and $[5\% - 15\%]$ lower for the LSTM and Deep LSTM networks in SN-direction. However, the models in NS-direction outperform the ones in SN-direction in both lateral acceleration as well as yaw rate prediction. This can be explained by the looking at the error scores of the constant prediction. The MSE is $\approx 40\%$ lower in NS-direction than in SN-direction. One explanation for this could be that, when driving in NS-direction, in the majority of corners the car is located on the outer lane with smaller radius of curvature, therefore decreasing lateral acceleration. Another explanation could be that the participants simply drove slower through corners reducing lateral acceleration. This artefact should be investigated further to completely understand the reasons why this is exactly happening.

In order to give more insight into the values presented in the table above, error trace figures for both longitudinal and lateral acceleration prediction are given in fig. 4.20 and fig. 4.21. For the error plots of the other DoF, consult fig. A.14, fig. A.15, fig. A.16 and fig. A.17. In these figures the MSE and MAE scores for each prediction method are plotted for each sample in the SN and NS test drive dataset. Consulting these plots gives more insight into the explanations tailored to table 4.13. For example, when looking at the MAE scores for the SN drive in fig. 4.20, it can be seen that the RNN-based networks generally perform better than the constant prediction or the simpler dense network prediction. However, when looking at the MSE scores for the SN drive, one can clearly deduce that relatively high error peaks are more prevalent in case of the constant prediction. This does follow the argument that high prediction errors are less present for model predictions. This effect is even more clear when looking at the lateral acceleration error plots given in fig. 4.21. However, in the case of lateral acceleration prediction the RNN-based networks also decreased the MAE more significantly when compared to the dense network and constant prediction. This effect can clearly be seen in fig. 4.21.
In the next section temporal prediction traces will be given to understand how this looks like in terms of prediction.

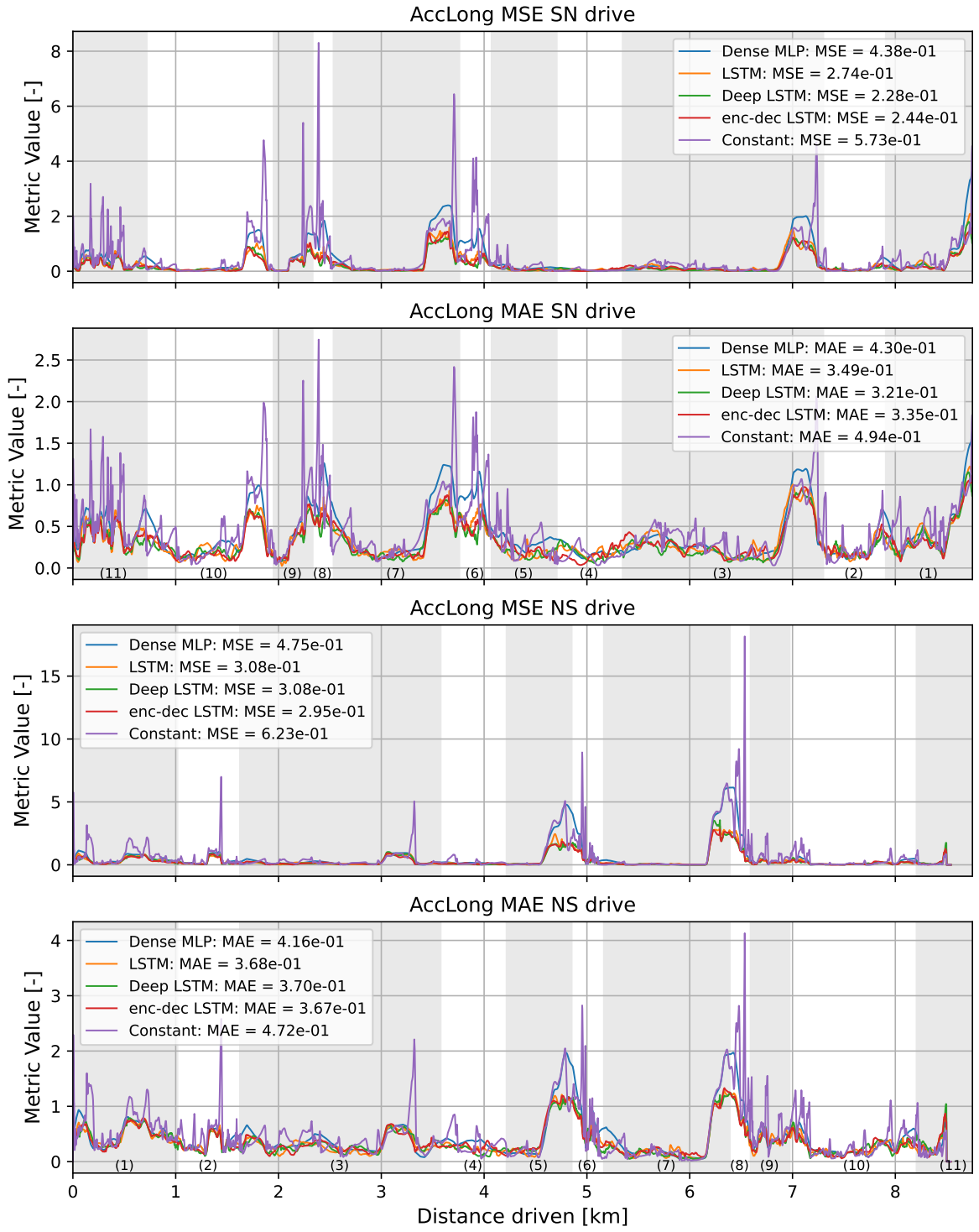


Figure 4.20: MSE and MAE trace for vehicle longitudinal acceleration prediction for both NS- and SN-direction using a trained Dense MLP, single layer LSTM, Deep LSTM and Enc-Dec LSTM network as well as the MSE and MAE trace of a constant prediction.

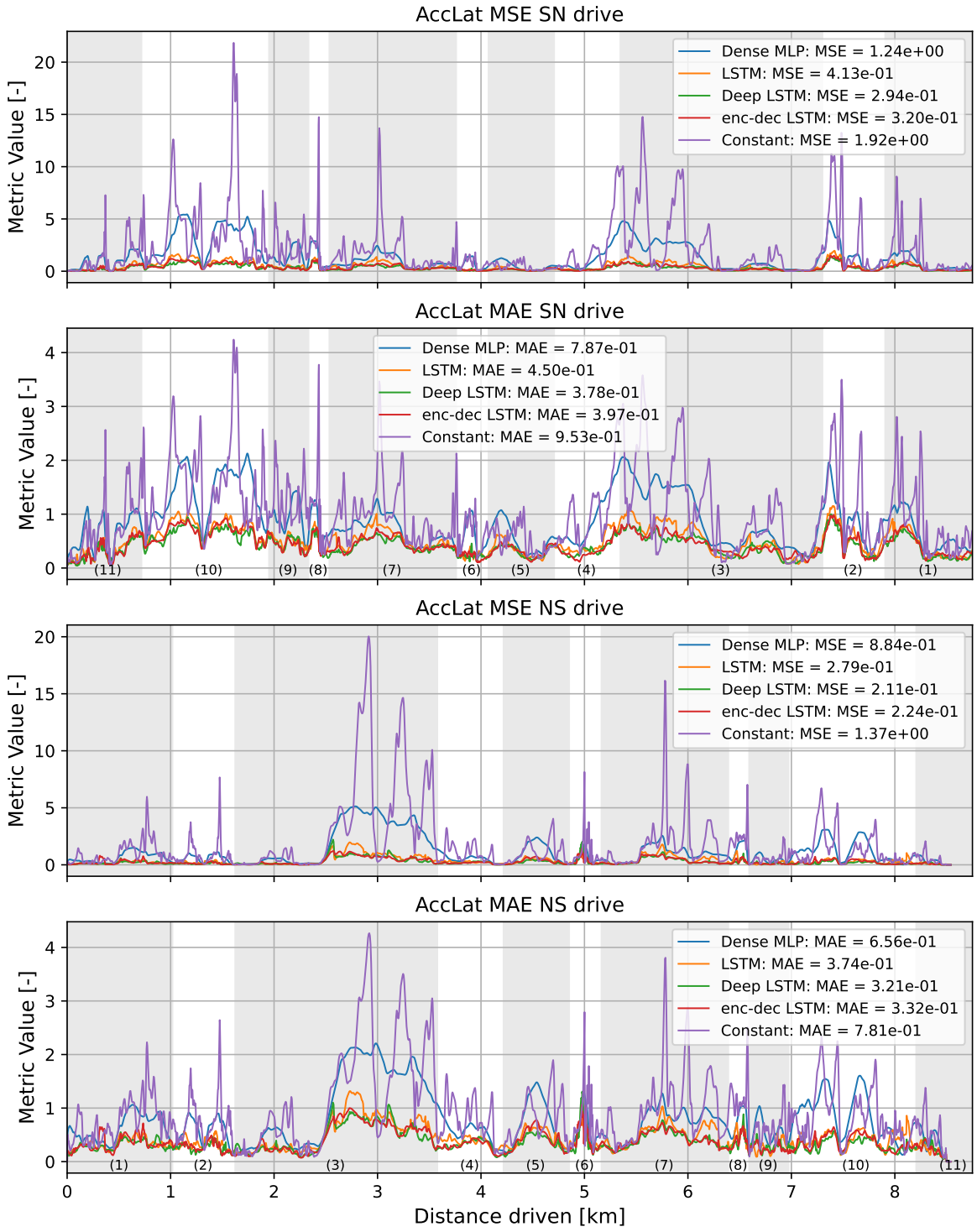


Figure 4.21: MSE and MAE trace for vehicle lateral acceleration prediction for both NS- and SN-direction using a trained Dense MLP, single layer LSTM, Deep LSTM and Enc-Dec LSTM network as well as the MSE and MAE trace of a constant prediction.

Time trace prediction analysis

In fig. 4.22 and fig. 4.23 time trace predictions are shown for the Dense MLP, LSTM, Deep LSTM and Enc-Dec LSTM networks. In these plots also the benchmark constant as well as the oracle (target) prediction is given. What can be observed is that the Dense network seem to have converged to a constant longitudinal acceleration prediction that lies around the value of ”0”. When analyzing further it was found that the network output is nearly constant for all DoF, i.e., model prediction is largely input independent. After investigating the weight and bias values of the dense layers it was found that almost all of the weights were negative. Combining this with the fact that the input to the Relu activation function is always positive or zero, i.e. input features are mapped to [0-1] and Relu outputs are also positive per definition, it means that most output neurons feature dead Relus. Thus, the network prediction is almost completely a bias-based prediction. As discussed earlier, this can occur when the learning rate is too large which causes too big weight and bias updates. This also explains the behavior that was observed in the loss plots in fig. 4.19.
It seems that the RNN-based networks, LSTM, Deep LSTM and Enc-Dec LSTM do not suffer from this behavior as much and do not feature a full dead Relu output layer. However, as seen in fig. 4.19 the learning does stagnate quickly, which denotes that dead Relu’s are present and do affect performance. Although the RNN-based networks somewhat follow the target, it can be seen that many of them are discontinuous and feature large jumps between the input and predicted sequence. It is worth investigating if the performance of the models does increase when lowering the learning rate or changing the Relu to the previously discussed Selu activation function, which should eliminate the risk of non-active neurons. Similar plots for the other predicted DoFs can be found in fig. A.18, fig. A.19, fig. A.20 and fig. A.21.

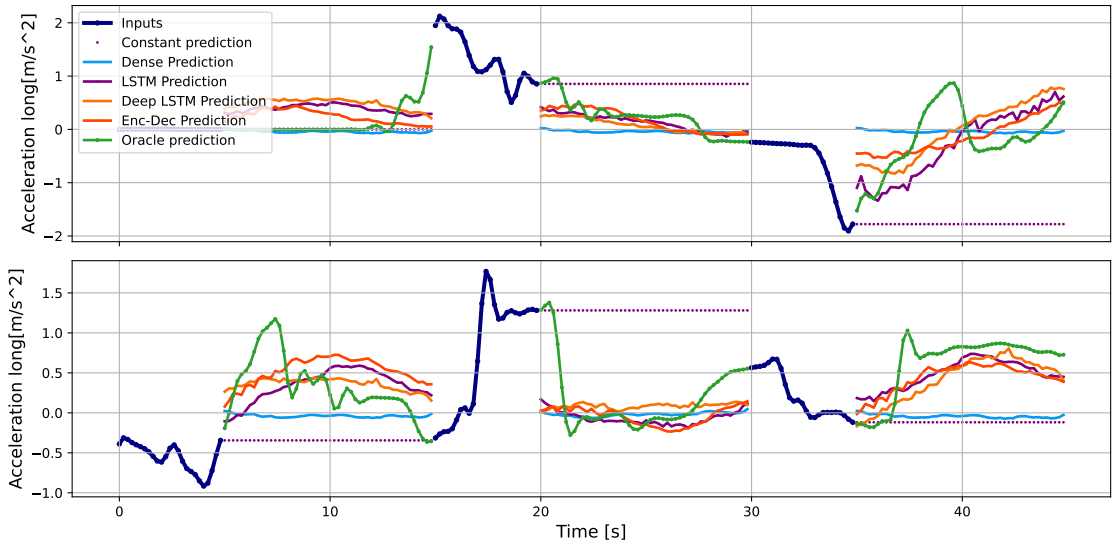


Figure 4.22: Longitudinal acceleration prediction traces using a trained MLP, LSTM, Deep LSTM, Enc-Dec network as well as a constant prediction trace.

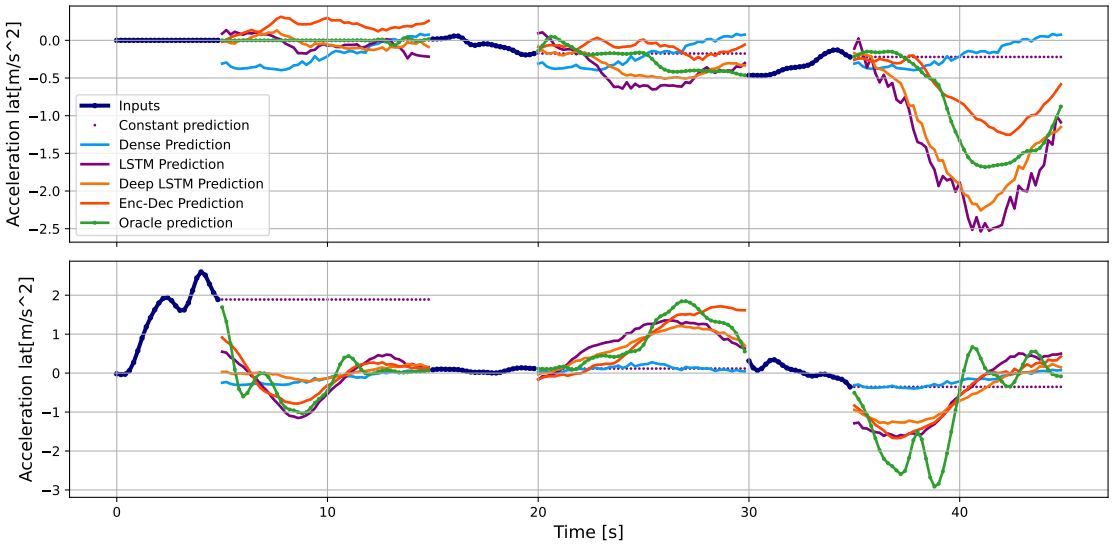


Figure 4.23: Lateral acceleration prediction traces using a trained MLP, LSTM, Deep LSTM, Enc-Dec network as well as a constant prediction trace.

4.10. Discussion and Conclusions

In the previous chapter the use of supervised data-driven approaches to perform driver reference generation for use in MPC MCA was investigated. First of all the data was reviewed and it was found that several signals, present in the data-set, had to be preprocessed by applying an offet or a lowpass filter. Included signals were the steering wheel angle, road curvature, and acceleration in longitudinal, lateral and vertical direction. Based on the MPC prerequisites it was argued that the network should output a predicted sequence including the translational accelerations and rotational rates in all DoFs. The set of input features was determined based on features found in literature and from the direct link with the to-be-predicted outputs. It was argued that a driver also uses upcoming environmental features to adapt his/her driving style accordingly. The upcoming speedlimit, road curvature, road width and road elevation were added to the input sequence by introducing a look-ahead distance. Since the input sequence is time and not distance dependent, the look-ahead features were digitized to the amount of input samples and mapped to a single value using either the bin-average or bin-maximum. Before setting-up the models, the data was down-sampled to reduce the number of output samples and decrease model complexity. The data was scaled using min-max scaling, where it was argued that differences in value ranges of input features is not desirable, an effect prevalent when using z-score normalization. Four separate models were investigated, a two hidden layer MLP, a single layer LSTM RNN, a three layer Deep LSTM RNN and an Encoder-Decoder LSTM RNN. Each model features dropout regularization on their dense layers. They also feature an output layer for each DoF consisting of a dense network with the amount of neurons equal to the output samples. These dense output layers exist pure out of functionality to split-up the output features and are the only layers that feature a linear activation function, i.e. $f(x) = x$ without any trainable parameters. The rest of the dense layers feature Relu activation functions with trainable weights and biases. Machine learning models feature many hyperparameters that can be categorized into structure, process or data hyperparameters. Only the learning rate, batch size, dropout rate and the amount of neurons/cells in each layer were optimized. The best performing hyperparameter values are presented in table 4.12. Fully trained networks were compared to the benchmark constant prediction and MSE and MAE metric improvements were computed for each model. Networks were trained for 50 epochs, after 50 training iterations it was found that the Deep LSTM network marginally outperformed the Encoder-Decoder LSTM network on both the MSE and MAE metrics for the test drive in both NS- as well as SN-direction. With the latter marginally outperforming the single layer LSTM RNN. The performance gains between the RNN models were found to be more prevalent for longitudinal and lateral acceleration as well as yaw rate prediction. The two hidden layer dense network was found to be the worst performing model. When investigating the temporal prediction traces it was found that the dense network output was largely independent of the

inputs caused by dead Relu's in the output layer of the network, an effect common when using too large learning rates [40]. It was argued that this can also negatively affect the other RNN-based network structures as they also feature a dense layer with Relu activation functions.
In this chapter many aspects of data-driven models were investigated. However, some aspects require more attention. A list of future recommendations is given below.

1. First of all, feature selection is completely absent for the networks presented in this chapter. Applying L1-regularization to the models could reduce the input feature dimension as it applies a penalty to the sum of weights, effectively reducing input feature dimensionality [65]. It should be investigated if this is even necessary because the input dimensionality is not extremely large, i.e. 20 input features. Also L1-regularization does not give more insight into which features are more important than others when certain predictions are made.

2. Another point of attention could be changing the outputs, now the assumption was made that the output of the network should be equal to the MPC reference DoFs. However, it could also be possible to make vehicle input predictions. These predicted inputs could then be used to drive the same virtual vehicle used in the simulator.

3. Another point of interest is reversing the order of the look-ahead features for the Encoder-Decoder LSTM network. In another study [91] it was found that this creates stronger connections between the tails of the input and predicted output of the model. In this case it can be argued that the first few meters of look-ahead features have a stronger correlation with short-term driver dynamics prediction.

4. More time can be spent on performing the grid search. Instead of training one model for each hyperparameter setting, multiple models need to be trained for more epochs with their performance averaged. This gives a better insight into which set of parameters actually work best. Next to this, also other parameters could be included into the hyperparameter optimization, such as the data down-sampling frequency, the stride, the length of the input sequence as well as the network structure itself. Also different regularization techniques could be investigated such as L1-, L2-regularization. At this moment no regularization is applied to the LSTM cells at all. LSTM regularization is left untouched at this moment in time but could very well result in better model performance.

5. For each of the models the Selu activation function should be applied. This eliminates the risk of introducing non-active neurons in the models, elongating the learning process, which effectively increases model performance. Altering this parameter also means that hyperparameter optimization should be performed again for each of the parameters.

6. Lastly, the effect of the prediction on motion cueing quality should be investigated. Four different models are presented that are able to predict the next 10 seconds of vehicle dynamics. However, the effect of the prediction quality on motion cueing quality is not known.

5 Project Planning

In Chapter 2 the advantages and disadvantages of the industry benchmark CWA were presented [85]. It was found that one of its main disadvantages is the inclusion of false cues due to parameter tuning as well as the exclusion of explicit simulator limits. In recent years a new and promising real-time motion cueing algorithm was devised, called ”model predictive control motion cueing algorithm” [22]. Experiments have shown that subjective as well as objective motion cueing quality increases when compared to the CWA [26]. Also, a strong correlation with a longer and more precise reference trajectory, and subjective motion cueing quality was found. Therefore, finding ways to provide such accurate reference was the subject of Chapter 4. In Chapter 4 data-driven, supervised prediction methods were discussed. Four different models were investigated, a two hidden layer MLP, a single LSTM RNN, a three layer Deep LSTM and an Encoder-Decoder LSTM RNN. It was found that these methods are capable of performing 10s future prediction for all 6 DoF's.
This chapter will serve as a proposal for future projects, based on the discussion provided at the end of each of the previous chapters. Keeping the goal of the thesis in mind, Section 5.1 discusses the specific projects on MPC as well as the data-driven methods. In Section 5.2 three different hypotheses on motion cueing quality will be stated, in order to validate the hypotheses an experiment will be performed. The experiment will briefly be discussed in Section 5.3.

5.1. Further Investigation

Although many concepts have been investigated in the previous chapters, certain aspects need to be investigated further or algorithms need to be expanded in the following way. Since the goal of the thesis is to understand how the prediction quality influences motion cueing quality, an experiment needs to be performed on at least a 6DoF simulator. To this extent it is paramount that the MPC algorithm in Chapter 3 is expanded to 6 DoF's. Translation in Z-direction as well as rotation around the Z-axis, i.e. yaw, need to be added. Another aspect that needs to be included is the possibility to reduce the order of the Hessian matrix H. This will reduce computational complexity and will make it possible to run fast simulations with a longer prediction horizon. For this, a move blocking strategy needs to be implemented [16]. Lastly, state weighting in the cost function needs to be implemented. This gives the possibility of making a trade-off between translational and rotational motion.
As discussed in Chapter 4, several methods for improving the models were presented. First and foremost the problem of dead Relu's needs to be taken care off. When fixed, the hyperparameters need to be tuned again. With this also the effect of using a different data scaling method will be investigated. Lastly, the effect of reversing the order of the look-ahead features used in the Encoder-Decoder network will be investigated. Then the performance needs to be rechecked, after which it shall be decided whether any other steps is required.
When the models have been analyzed further, the MPC algorithm and the different prediction methods will be combined. Offline simulations will be done with a constant 1.5s future prediction as found in [26], the 10s future network predictions as well as an oracle 10s future prediction. The best performing prediction model - MPC combination will be used in the experiment. This will be decided by performing an analysis on RMSE on cueing errors as well as an analysis on the amount of false cues as performed

in [28].

5.2. Research Questions and Hypotheses

Based on the topics discussed previously the following research question is presented.

"How does subjective and objective motion cueing quality, using model predictive control, alter when the output of more accurate data-driven supervised models, that predict driver behavior, are used as reference trajectory."

It should be noted that this research question is preceded by two other research questions that need to be answered first.

"How does prediction quality improve when using data-driven supervised machine learning methods compared to the benchmark constant prediction."

and

"What is the effect of using MPC with constant, a more accurate model, and perfect oracle prediction on simulator motion."

The first of the two sub-questions has partially been answered in Section 4.9, however, this analysis needs to be expanded in the future with the reworked networks. The latter sub-question needs to be answered in follow-up investigations. The main research question shall be validated, only when the two sub-questions are answered, by performing experiments where participants will experience four different conditions:

1. Condition 1: Data from prerecorded drive used to compute offline simulator commands using CWA MCA industry benchmark.
2. Condition 2: Offline motion simulator commands generated by MPC MCA using a constant 1.5s prediction as used in [26].
3. Condition 3: Offline motion simulator commands generated by MPC MCA using a 10s trained, NN model prediction.
4. Condition 4: Offline motion simulator commands generated by MPC MCA using a 10s oracle prediction.

All four conditions will be set-up using the dataset that was utilized throughout the investigations provided in this thesis report. The hypotheses concerning the four conditions are defined as follows.

- Perceived motion cueing quality of MPC with constant prediction is marginally better than the quality of CWA cueing, also found by Ellensohn [28]. This will be the anchor measurement.
- Perception of motion cueing quality of MPC with an accurate prediction model is better than MPC with constant prediction.
- Perception of motion cueing quality of MPC with oracle prediction model is better than MPC with an accurate prediction model.

5.3. Experiment Design

5.3.1. Rating Method

Cleij et al. [19] proposes a continuous rating (CR) method where participants rate the perceived motion incongruence, or so called PMI which results in a motion incongruence rating or MIR. Participants drive passively in a dynamic driving simulator, i.e. not operating the steering wheel or gas/brake pedals. This way, participants are able to fully concentrate on rating the PMI. The MIR scale ranges from 0 to 10, 0 meaning no perceived motion mismatch, whereas a 10 indicates the largest perceived motion mismatch in the experiment. Participants will be trained on some prerecorded test drives prior to the experiment to understand what these scores entail. As discussed in [19], the training set will include regions of low as

well as regions of large cueing error with a specific set of cueing errors the participant may experience (e.g. scaling error or sign error). Performing this step is found to improve the overall rating quality and consistency [19][54][28]. The biggest advantage in using this rating approach is the high temporal resolution it produces. The acquired results can be used in the validation to find correlation between physical (measurable mismatches) and perceived mismatches. However, some disadvantages to this method also exist as discussed by Kolff et al.[54].

- One made assumption is that any non-zero ratings occur due to perceived motion mismatches. This means that performing such a rating task in a real-vehicle should result in perfect (zero) rating at all times. Since participants are asked to specifically rate these mismatches, they are more likely to actively focus on the rating when compared to driving a real car. This will most likely result in more non-zero ratings even if no mismatch exists.
- Another assumption is that the CR by the participants is an accurate representation of actual perceived motion. However, humans exhibit an inherent time delay, which is corrected for in the models by Cleij et al. [19]. Humans also can exhibit the tendency to anticipate on previous experienced mismatches in previous drives, meaning the perception of a past maneuver might affect the rating at the current time.

Even though these flaws exist, their influence can be somewhat limited (not completely omitted) by careful instruction to the participants. Contrary to [19], where each trial is subdivided into segments, but similar to [26] each participant will give a post-hoc rating (PR) at the end of each trial (not segmented). The PR is used to validate the results of the CR [19]. The rating scale of the PR and CR are the same. The CR results from a passive drive provides high resolution and high quality motion cueing quality scores. Therefore, this method will be used for the experiment.

5.3.2. General Experiment Set-up

The set-up of the experiments closely follows the set-up used by Cleij et al. [19] and Kolff et al. [54]. Experiments will take place at the simulation headquarters of BMW in Munich on an, at this time undisclosed, available motion simulator. Participants will passively experience a drive on the route found in fig. 4.2. To create the offline drives that participants will experience, part of the dataset mentioned in Section 5.2 will be used. This means one of the drives present in the test-set, discussed in Section 4.4, will be used as reference drive. For the drive in the test-set, the prediction performance of all four different NN networks will be analyzed. The model that shows the best performance will be used as reference generation model for the MPC MCA used in condition 3. To anchor the measurement and validate the results, offline generated simulator setpoints are calculated. For condition 1, a BMW tuned CWA is used. For condition 2, 3 and 4, the prediction sequences (i.e. constant, NN model and oracle) will be integrated with a BMW in-house developed MPC controller into BMW simulator control software.

5.3.3. Participant Group

BMW will give full access to the simulator hardware and software during a six week time period. The first three to four weeks are used to test and verify the experiment set-up. Then, in the remaining two to three weeks, the experiment will take place occupying around 40-60 participants, which averages around 4 participants per day. Participants are gathered through an online BMW portal which usually fills up within the hour. The participants will experience 4 drives randomly ordered: motion cueing by CWA, offline MPC with constant prediction, offline MPC with the prediction model and offline MPC with oracle prediction. The experiment features the driving scenario as independent variable: CWA, MPC MCA with constant, model, and oracle prediction. The dependent variables are the MRI and PR.

5.3.4. Scientific Purpose

To finalize this thesis report, this short paragraph is committed to the discussion on the scientific relevance of this research.
In this report many questions were asked that ultimately lead to the presentation of the hypotheses made in section 5.2. This thesis committed itself to provide answers to some of these questions. Further research, by performing the proposed experiments, tries to provide supporting evidence to conclude the hypotheses. Multiple goals, some of which already attained, were the foundation of this research.

The first goal was to understand if it is possible to model driver dynamic behavior, based on a simple set of measurable environmental as well as personal features, using complex Neural Network model types. If possible, to which degree these models can provide accurate predictions. These two questions have already been partly answered, however, as discussed more time should be invested to remove some small bugs present in the models. The results of this investigation can be used by scientists in other fields that research specific dynamic driver behavior using Neural Network models as well.
The goal of performing the experiments is to gather information that can lead to answers in different areas. The main goal being to understand the impact on motion cueing quality of using a complex reference prediction model compared to a constant reference. To give a better insight to the experiment results, a perfect prediction reference is also used. This is a situation which is unattainable, yet desired. The resulting conclusion from these experiments can provide future scientists a lead to follow up on: is it worth it to invest a lot of time into the matter of developing time consuming and complex prediction models if it yields no significant benefit? The other goal is to investigate to which degree an increase in prediction quality has an effect on the control sequence, mainly the first control input, calculated by the MPC algorithm. More specifically, does this increase in prediction quality effectively change the discrete time sequence of computed first control inputs applied to the system and enhance the objective/subjective motion cueing quality? Answering these questions are expected to increase scientific understanding on the subject of MPC MCA further.

Bibliography

[1] *ISO 8855:2011 Road vehicles — Vehicle dynamics and road-holding ability — Vocabulary*, 2011-12.

[2] Andrei Aksjonov, Pavel Nedoma, Valery Vodovozov, Eduard Petlenkov, and Martin Herrmann. A novel driver performance model based on machine learning. *15th IFAC Symposium on Control in Transportation Systems CTS 2018*, 51(9):267–272, 2018. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2018.07.044. URL https://www.sciencedirect.com/science/article/pii/S2405896318307675.

[3] D. Allerton. *Principles of Flight Simulation*. AIAA education series. Wiley, 2009. ISBN 9781600867033.

[4] Afshine Amidi and Shervine Amidi. Recurrent neural networks cheatsheet. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks, 2019.

[5] Mauro Baseggio, Alessandro Beghi, Mattia Bruschetta, Fabio Maran, and Diego Minen. An MPC approach to the design of motion cueing algorithms for driving simulators. In *Proceedings of the 2011 14th International IEEE Conference on Intelligent Transportation Systems*, Washington DC, USA, 10 2011. doi: 10.1109/ITSC.2011.6083053.

[6] Alessandro Beghi, Mattia Bruschetta, and Fabio Maran. A real time implementation of MPC based Motion Cueing strategy for driving simulators. In *Proceedings of the 51st IEEE Conference on Decision and Control, Maui, Hawaii*, December 2012. doi: 10.1109/CDC.2012.6426119.

[7] Alessandro Beghi, Mattia Bruschetta, Fabio Maran, and Diego Minen. A Model-based Motion Cueing Strategy for Compact Driving Simulation Platforms. In *Proceedings of the Driving Simulation Conference 2012*, Paris, France, 09 2012.

[8] Martin Behrendt. Receding horizon optimal mpc. https://commons.wikimedia.org/w/index.php?curid=7963069.

[9] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[10] Christian Bielaczek. Die auswirkung der aktiven fahrerbeeinflussung auf die fahrsicherheit. *ATZ - Automobiltechnische Zeitschrift*, 101(9):714–724, 1999. ISSN 2192-8800. doi: 10.1007/BF03224277. URL https://doi.org/10.1007/BF03224277.

[11] BMW. BMW group sets new standards for driving simulation. https://www.press.bmwgroup.com/global/article/detail/T0320021EN/, 11 2020.

[12] Waad Bouaguel, Ghazi Mufti, and Mohamed Limam. On the effect of search strategies on wrapper feature selection in credit scoring. pages 218–223, 05 2013. ISBN 978-1-4673-5547-6. doi: 10.1109/CoDIT.2013.6689547.

[13] Danko Brezak, Tomislav Bacek, Dubravko Majetic, Josip Kasac, and Branko Novakovic. A comparison of feed-forward and recurrent neural networks in time series forecasting. In *2012 IEEE Conference on Computational Intelligence for Financial Engineering and Economics (CIFEr)*, pages 1–6, 2012. doi: 10.1109/CIFEr.2012.6327793.

[14] Mattia Bruschetta, Fabio Maran, Carlo Cenedese, Alessandro Beghi, and Diego Minen. An mpc-based motion cueing implementation with time-varying prediction and drivers skills characterization. In Andras Kemeny, Frédéric Merienne, Florent Colombet, and Stéphane Espié, editors, *Proceedings of the Driving Simulation Conference 2016 Europe*, pages 11–20, Paris, France, 2016. Driving Simulation Association.

[15] Mattia Bruschetta, Carlo Cenedese, and Alessandro Beghi. A real-time, mpc-based motion cueing algorithm with look-ahead and driver characterization. *Transportation Research Part F: Traffic Psychology and Behaviour*, 61:38–52, 2019. ISSN 1369-8478. doi: https://doi.org/10.1016/j.trf.2017.04.023. URL https://www.sciencedirect.com/science/article/pii/S1369847816306593. Special TRF issue: Driving simulation.

[16] Raphael Cagienard, Pascal Grieder, Eric Kerrigan, and Manfred Morari. Move blocking strategies in receding horizon control. *Journal of Process Control*, 17(6):563–570, 07 2007. ISSN 0959-1524. doi: 10.1016/j.jprocont.2007.01.001.

[17] François Chollet et al. Keras. https://keras.io, 2015.

[18] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. 12 2014.

[19] Diane Cleij, Joost Venrooij, Paolo Pretto, Daan Marinus Pool, Max Mulder, and Heinrich H. Bülthoff. Continuous subjective rating of perceived motion incongruence during driving simulation. *IEEE Transactions on Human-Machine Systems*, 48(1):1–13, 09 2017. doi: 10.1109/THMS.2017.2717884.

[20] Bjorn Conrad, J. G. Douvillier, and Stanley F. Schmidt. Washout Circuit Design for Multi-Degrees-Of-Freedom Moving Base Simulators. In *Proceedings of the AIAA Visual and Motion Simulation Conference, Palo Alto (CA)*, number AIAA-1973-929, 1973.

[21] B.J. Correia Grácio. *The Effects of Specific Force on Self-Motion Perception in a Simulation Environment*. PhD thesis, Delft University of Technology, 11 2013.

[22] M. Dagdelen, G. Reymond, Andras Kemeny, M. Bordier, and N. Maïzi. MPC Based Motion Cueing Algorithm: Development and Application to the ULTIMATE Driving Simulator. In *Proceedings of the Driving Simulation Conference 2004 Europe*, pages 221–233, Paris, France, 2004.

[23] Daimler. The new driving simulator: Fast response and a photorealistic environment. https://media.daimler.com/marsMediaSite/en/instance/ko/The-new-driving-simulator-Fast-response-and-a-photorealistic-environment.xhtml?oid=9362177, 07 2011.

[24] Frank M. Drop, Mario Olivari, Mikhail Katliar, and Heinrich H. Bülthoff. Model predictive motion cueing: Online prediction and washout tuning. In Andras Kemeny, Florent Colombet, Frédéric Merienne, and Stéphane Espié, editors, *Proceedings of the 17th Driving Simulation Conference 2018 Europe VR*, pages 71–78, Antibes, France, 2018. Driving Simulation Association. ISBN 978-2-85782-734-4.

[25] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011.

[26] Felix Ellensohn, Dimitar Hristakiev, Markus Schwienbacher, Joost Venrooij, and Daniel Rixen. Evaluation of an optimization based motion cueing algorithm suitable for online application. In Andras Kemeny, Florent Colombet, Frédéric Merienne, and Stéphane Espié, editors, *Proceedings of the Driving Simulation Conference 2019 Europe VR*, pages 93–100, Strasbourg, France, 2019. Driving Simulation Association. ISBN 978-2-85782-749-8.

[27] Felix J. Ellensohn. *Urban Motion Cueing Algorithms – Trajectory Optimization for Driving Simulators*. PhD thesis, University of Technology München, dec 2019.

[28] Felix J. Ellensohn. *Urban Motion Cueing Algorithms – Trajectory Optimization for Driving Simulators*. PhD thesis, University of Technology München, dec 2019.

[29] Atila Ertas and Jesse C. Jones. *The Engineering Design Process*. Wiley and Sons, 2nd edition, November 1996.

[30] Zhou Fang and Andras Kemeny. Explicit mpc motion cueing algorithm for real-time driving simulator. In *Proceedings of The 7th International Power Electronics and Motion Control Conference*, volume 2, pages 874–878, 2012. doi: 10.1109/IPEMC.2012.6258965.

[31] Zhou Fang and Andras Kemeny. An efficient model predictive control-based motion cueing algorithm for the driving simulator. *SIMULATION*, 92, 10 2016. doi: 10.1177/0037549716667835.

[32] Zhou Fang, M. Tsushima, E. Kitahara, N. Machida, D. Wautier, and Andras Kemeny. Motion cueing algorithm for high performance driving simulator using yaw table. *IFAC-PapersOnLine*, 50(1):15965–15970, 07 2017. doi: 10.1016/j.ifacol.2017.08.1750.

[33] H.J. Ferreau, A. Potschka, and C. Kirches. qpOASES webpage. http://www.qpOASES.org/, 2007–2017.

[34] H.J. Ferreau, C. Kirches, A. Potschka, H.G. Bock, and M. Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

[35] Christodoulos A. Floudas and V. Visweswaran. Quadratic optimization, 1995.

[36] Mohammad Fneish. Keras lstm diagram. https://github.com/MohammadFneish7/Keras_LSTM_Diagram, 2019.

[37] Carlos E. García, David M. Prett, and Manfred Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989. ISSN 0005-1098. doi: https://doi.org/10.1016/0005-1098(89)90002-2. URL https://www.sciencedirect.com/science/article/pii/0005109889900022.

[38] Tortora Gerard J. *Principles of anatomy 'I&' physiology /*. Wiley, 2014. ISBN 1118345002.

[39] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13-15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/glorot10a.html.

[40] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[41] Peter R. Grant and Lloyd D. Reid. Motion Washout Filter Tuning: Rules and Requirements. *Journal of Aircraft*, 34(2):145–151, 1997. doi: 10.2514/2.2158.

[42] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*, volume 385. 01 2012. ISBN 978-3-642-24796-5. doi: 10.1007/978-3-642-24797-2.

[43] Alex Graves. Generating sequences with recurrent neural networks. 08 2013.

[44] Eric Groen, Mario Clari, and Ruud Hosman. *Psychophysical thresholds associated with the simulation of linear acceleration*. doi: 10.2514/6.2000-4294. URL https://arc.aiaa.org/doi/abs/10.2514/6.2000-4294.

[45] Eric L. Groen and W Bles. How to use body tilt for the simulation of linear self motion. *Journal of vestibular research : equilibrium 'I&' orientation*, 14(5):375–85, December 2004.

[46] Eric L. Groen, Mario S. V. Valenti Clari, and Ruud J. A. W. Hosman. Evaluation of Perceived Motion during a Simulated Takeoff Run. *Journal of Aircraft*, 38(4):600–606, July/August 2001. doi: 10.2514/2.2827.

[47] Mark Hall. Correlation-based feature selection for machine learning. *Department of Computer Science*, 19, 06 2000.

[48] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

[49] Ashesh Jain, Hema Koppula, Bharad Raghavan, Shane Soh, and Ashutosh Saxena. Car that knows before you do: Anticipating maneuvers via learning temporal driving models. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 3182–3190, Santiago, Chile, 12 2015. doi: 10.1109/ICCV.2015.364.

[50] M Katliar. *Optimal control of motion simulators*. PhD thesis, Albert-Ludwigs-Universität, Freiburg i.Br., Germany, 2020.

[51] Mikhail Katliar. *Optimal control of motion simulators*. PhD thesis, University of Freiburg, Georges-Köhler-Allee 102, DE-79110 Freiburg, dec 2020.

[52] Diederik P. Kingma and Jimmy Ba. Adam, A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations (ICLR)*, San Diego, California, USA, 2015.

[53] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, Long Beach, California, USA, 2017. ISBN 9781510860964. doi: 10.5555/3294771.3294864.

[54] Maurice Kolff, Joost Venrooij, Markus Schwienbacher, Daan M. Pool, and Max Mulder. Quality comparison of motion cueing algorithms for urban driving simulations. *Actes (IFSTTAR)*, pages 141–148, 2021. ISSN 2115-418X. 20th Driving Simulation and Virtual Reality Conference and Exhibition, DSC 2021 EUROPE ; Conference date: 14-09-2021 Through 17-09-2021.

[55] Sotiris Kotsiantis, Dimitris Kanellopoulos, and P. Pintelas. Data preprocessing for supervised learning. *International Journal of Computer Science*, 1(12):111–117, 01 2006.

[56] Jan Kukačka, Vladimir Golkov, and Daniel Cremers. Regularization for deep learning: A taxonomy, 2018. URL https://openreview.net/forum?id=SkHkeixAW.

[57] B. Kumari and Tripti Swarnkar. Filter versus wrapper feature subset selection in large dimensionality micro array: A review. *International Journal of Computer Science and Information Technologies*, 2:1048–1053, 01 2011.

[58] Ohbyung Kwon and Jae Mun Sim. Effects of data set features on the performances of classification algorithms. *Expert Systems with Applications*, 40(5):1847–1857, 2013. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2012.09.017. URL https://www.sciencedirect.com/science/article/pii/S0957417412010718.

[59] Wook Hyun Kwon and Soo Hee Han. *Receding Horizon Control*. Advanced Textbooks in Control and Signal Processing. Springer-Verlag London, 1 edition, 2005. doi: 10.1007/b136204.

[60] Alexander Lamprecht, Tim Emmert, Dennis Steffen, and Knut Graichen. Learning-based driver prediction for mpc-based motion cueing algorithms. In Andras Kemeny, Jean-Rémy Chardonnet, and Florent Colombet, editors, *Proceedings of the Driving Simulation Conference 2021 Europe VR*, pages 133–140, Munich, Germany, 2021. Driving Simulation Association.

[61] Mark Lau, S Yue, Keck Ling, and Jan Maciejowski. A comparison of interior point and active set methods for fpga implementation of model predictive control. *Proc. European Control Conference*, 03 2015.

[62] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: A big comparison for nas. 12 2019.

[63] Jacob Mattingley, Yang Wang, and Stephen Boyd. Code generation for receding horizon control. In *Part of 2010 IEEE Multi-Conference on Systems and Control*, September 2010.

[64] J.A. Mulder, W.H.J.J. van Staveren, J.C. van der Vaart, E. de Weerdt, C.C. de Visser, A.C. in 't Veld, and E. Mooij. *FLight Dynamics*. Lecture Notes AE3202. Faculty of Aerospace Engineering, 2013.

[65] Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 78, New York, NY, USA, 2004. Association for Computing Machinery. ISBN 1581138385. doi: 10.1145/1015330.1015435. URL https://doi.org/10.1145/1015330.1015435.

[66] Council of the EU. New cars in eu will need advanced safety systems from 2022. https://www.consilium.europa.eu/en/press/press-releases/2019/11/08/safer-cars-in-the-eu/, 11 2019.

[67] A.V. Oppenheim and A.S. Willsky. *Signals and Systems: Pearson New International Edition*. Always learning. Pearson Education Limited, 2013. ISBN 9781292025902.

[68] Charles Clark Ormsby. Model of human dynamic orientation. 1974.

[69] Gabriele Pannocchia, James B. Rawlings, and Stephen J. Wright. Fast, large-scale model predictive control by partial enumeration. *Automatica*, 43(5):852–860, 2007. ISSN 0005-1098. doi: https://doi.org/10.1016/j.automatica.2006.10.019. URL https://www.sciencedirect.com/science/article/pii/S0005109806004511.

[70] Arben Parduzi, Joost Venrooij, and Stefanie Marker. The effect of head-mounted displays on the behavioural validity of driving simulators. In Andras Kemeny, Jean-Rémy Chardonnet, and Florent Colombet, editors, *Proceedings of the Driving Simulation Conference 2020 Europe VR*, pages 125–132, Antibes, France, 2020. Driving Simulation Association.

[71] Joost Ploeg, Diane Cleij, Daan Marinus Pool, Max Mulder, and Bülthoff H.H. Sensitivity analysis of an mpc-based motion cueing algorithm for a curve driving scenario. In *Proceedings of the 19th Driving Simulation Conference 2020*, Antibes, France, 09 2020.

[72] Aaron Pressman. Waymo reaches 20 million miles of autonomous driving. https://fortune.com/2020/01/07/googles-waymo-reaches-20-million-miles-of-autonomous-driving/, 01 2020.

[73] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(98)00116-6. URL https://www.sciencedirect.com/science/article/pii/S0893608098001166.

[74] C. V. Rao, S. J. Wright, and J. B. Rawlings. Application of interior-point methods to model predictive control. *J. Optim. Theory Appl.*, 99(3):723–757, December 1998. ISSN 0022-3239.

[75] Carl Edward Rasmussen and Zoubin Ghahramani. Occam's razor. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, page 276–282, Cambridge, MA, USA, 2000. MIT Press.

[76] J.B. Rawlings, D.Q. Mayne, and M.M. Diehl. *Model Predictive Control: Theory, Computation, and Design 2nd Edition*. Nob Hill Publishing, 2017. ISBN 9780975937754.

[77] Lloyd D. Reid and Meyer A. Nahon. Flight Simulation Motion-Base Drive Algorithms. Part 1: Developing and Testing the Equations. Technical Report UTIAS 296, University of Toronto, Institute for Aerospace Studies, December 1985.

[78] Lloyd D. Reid and Meyer A. Nahon. Flight Simulation Motion-Base Drive Algorithms. Part 2: Selecting the System Parameters. Technical Report UTIAS 307, University of Toronto, Institute for Aerospace Studies, May 1986.

[79] Renault. "roads", a new renault driving simulator for autonomous vehicle. https://www.renaultgroup.com/en/news-on-air/news/roads-a-new-renault-driving-simulator-for-autonomous-vehicle/, 09 2017.

[80] Richard Romano, Ehsan Sadraei, and Gustav Markkula. Rapid tuning of the classical motion cueing algorithm. In Andras Kemeny, Florent Colombet, Frédéric Merienne, and Stéphane Espié, editors, *Proceedings of the Driving Simulation Conference 2017 Europe VR*, pages 89–92, Stuttgart, Germany, 2017. Driving Simulation Association.

[81] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. , 323(6088):533–536, October 1986. doi: 10.1038/323533a0.

[82] David E. Rumelhart, Geoffrey E Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 01 1986.

[83] Murat Sazli. A brief review of feed-forward neural networks. *Communications Faculty Of Science University of Ankara*, 50(1):11–17, 01 2006. doi: 10.1501/0003168.

[84] Robin Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview, 11 2019.

[85] Stanley F. Schmidt and Bjorn Conrad. Motion Drive Signals for Piloted Flight Simulators. Technical Report NASA CR-1601, National Aeronautics and Space Administration, Ames Research Center, 1970.

[86] Bernhard Schöner, Hans-Peterand Morys. *Dynamic Driving Simulators*, pages 177–198. Springer International Publishing, Cham, 2016. ISBN 9783319123523. doi: 10.1007/978-3-319-12352-3_9.

[87] Ivan Selesnick and Charles Burrus. Generalized digital butterworth filter design. *IEEE Transactions on Signal Processing*, 46, 05 1998. doi: 10.1109/78.678493.

[88] Amir Shahzad and Paul J. Goulart. A new hot-start interior-point method for model predictive control*. *IFAC Proceedings Volumes*, 44(1):2470–2475, 2011. ISSN 1474-6670. doi: https://doi.org/10.3182/20110828-6-IT-1002.02817. URL https://www.sciencedirect.com/science/article/pii/S1474667016439844. 18th IFAC World Congress.

[89] J. Sola and Joaquin Sevilla. Importance of input data normalization for the application of neural networks to complex industrial problems. *Nuclear Science, IEEE Transactions on*, 44(3):1464 – 1468, 07 1997. doi: 10.1109/23.589532.

[90] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 06 2014.

[91] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, volume 2, page 3104–3112, Montreal, Canada, 2014.

[92] Noelia Sánchez-Maroño, Amparo Alonso-Betanzos, and María Tombilla-Sanromán. Filter methods for feature selection – a comparative study. pages 178–187, 12 2007. ISBN 978-3-540-77225-5. doi: 10.1007/978-3-540-77226-2_19.

[93] Zaiyong Tang and Paul Fishwick. Feedforward neural nets as models for time series forecasting. *INFORMS Journal on Computing*, 5(4):374–385, 11 1993. doi: 10.1287/ijoc.5.4.374.

[94] Orit Taubman-Ben-Ari and Vera Skvirsky. The multidimensional driving style inventory a decade later: Review of the literature and re-evaluation of the scale. *Accident Analysis and Prevention*, 93:179–188, 8 2016. ISSN 00014575. doi: 10.1016/j.aap.2016.04.038.

[95] Robert J. Telban and Frank M. Cardullo. Motion Cueing Algorithm Development: Human-Centered Linear and Nonlinear Approaches. Technical Report NASA CR-2005-213747, National Aeronautics and Space Administration, Langley Research Center, Hampton, Virginia 23681-2199, May 2005.

[96] Inc. The MathWorks. Matlab 2018b.

[97] Toyota. Toyota develops world-class driving simulator: Real-as-possible environment to aid development of active safety technology. https://global.toyota/en/detail/277491, 11 2007.

[98] S. J. Tupling and Michael R. Pierrynowski. Use of cardan angles to locate rigid bodies in three-dimensional space. *Medical and Biological Engineering and Computing*, 25:527–532, 2006.

[99] Aleksander Väljamäe, Pontus Larsson, Daniel Västfjäll, and Mendel Kleiner. Travelling without moving: Auditory scene cues for translational self-motion. 2005.

[100] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[101] Liuping Wang. *Model Predictive Control Systems Design and Implementation Using MATLAB*. Advances in Industrial Control. Springer, 2009. ISBN 9781848823303.

[102] Yang Wang and Stephen Boyd. Fast model predictive control using online optimization. *IFAC Proceedings Volumes*, 41(2):6974–6979, 2008. ISSN 1474-6670. doi: https://doi.org/10.3182/20080706-5-KR-1001.01182. URL https://www.sciencedirect.com/science/article/pii/S1474667016400662. 17th IFAC World Congress.

[103] William Warren. Self-motion: Visual perception and visual control. *Perception of Space and Motion*, 01 1995.

[104] E. Alper Yildirim and Stephen Wright. Warm-start strategies in interior-point methods for linear programming. *SIAM Journal on Optimization*, 12:782–810, 01 2002. doi: 10.1137/S1052623400369235.

[105] Laurence R. Young and Jacob L. Meiry. A revised dynamic otolith model. *Aerospace medicine*, 39 6:606–8, 1968.

[106] Greg L. Zacharias. Motion cue models for pilot-vehicle analysis. 1978.

III

Appendices

A Appendix Preliminary Thesis

A.1. Chapter 2

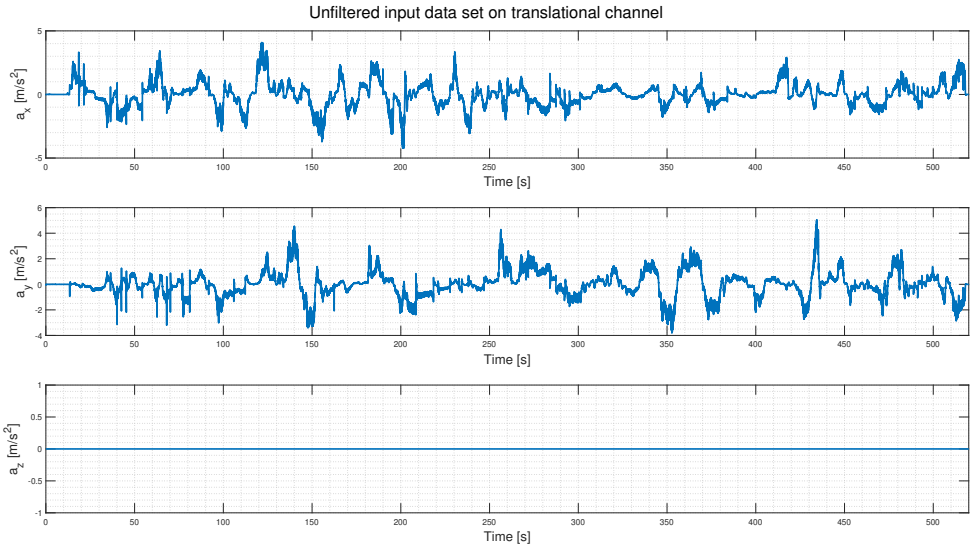


Figure A.1: Full translational vehicle data from BMW test run.

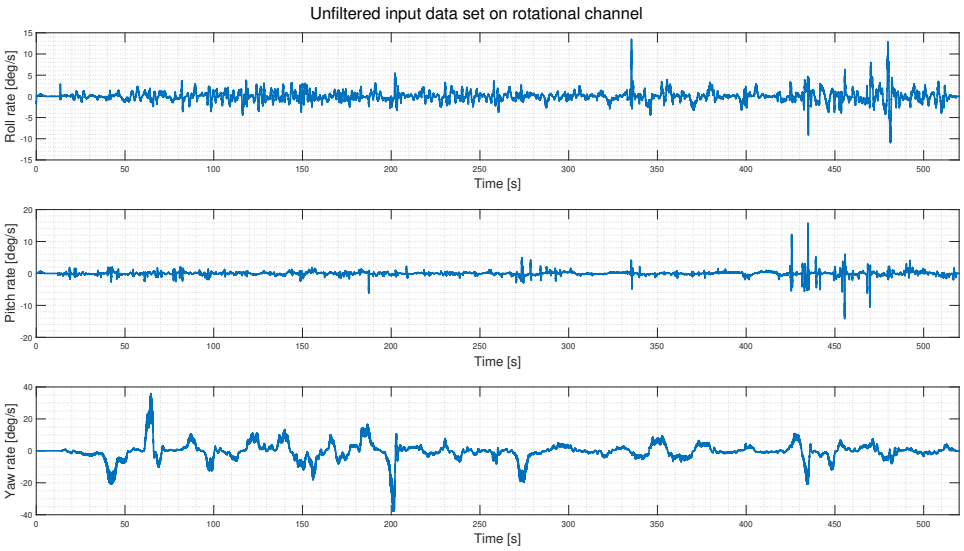


Figure A.2: Full rotational vehicle data from BMW test run.

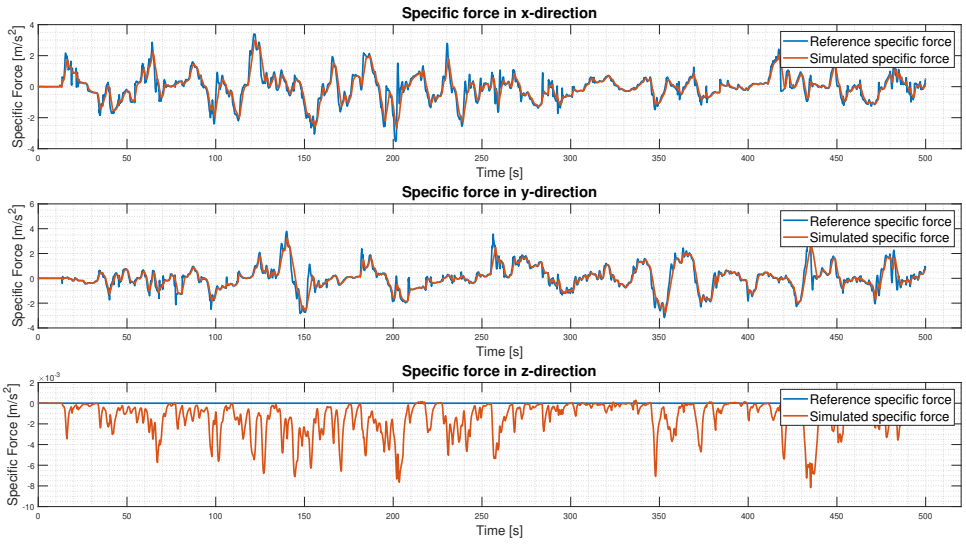


Figure A.3: CWA reference vs simulated specific forces in body coordinate reference frame using BMW test drive as reference.

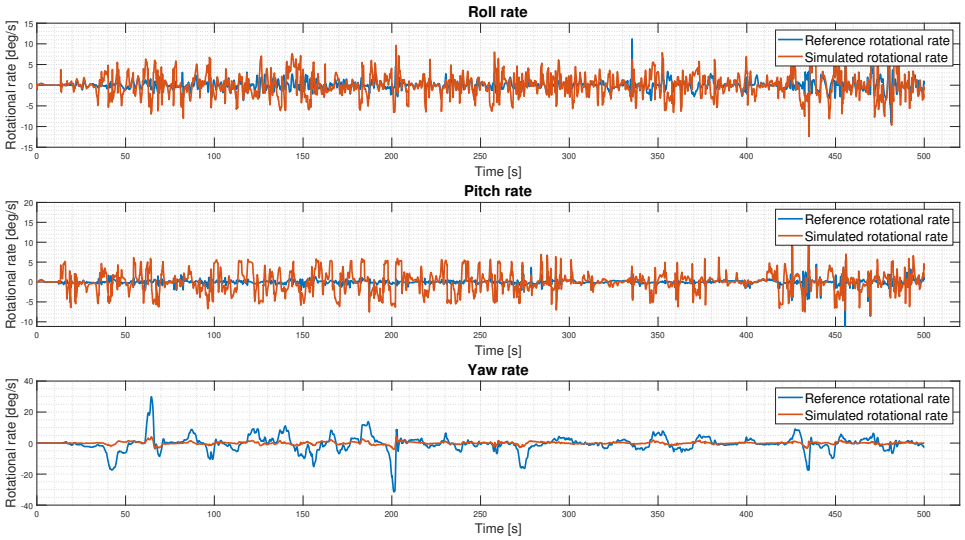


Figure A.4: CWA reference vs simulated rotational rates in body coordinate reference frame using BMW test drive as reference.

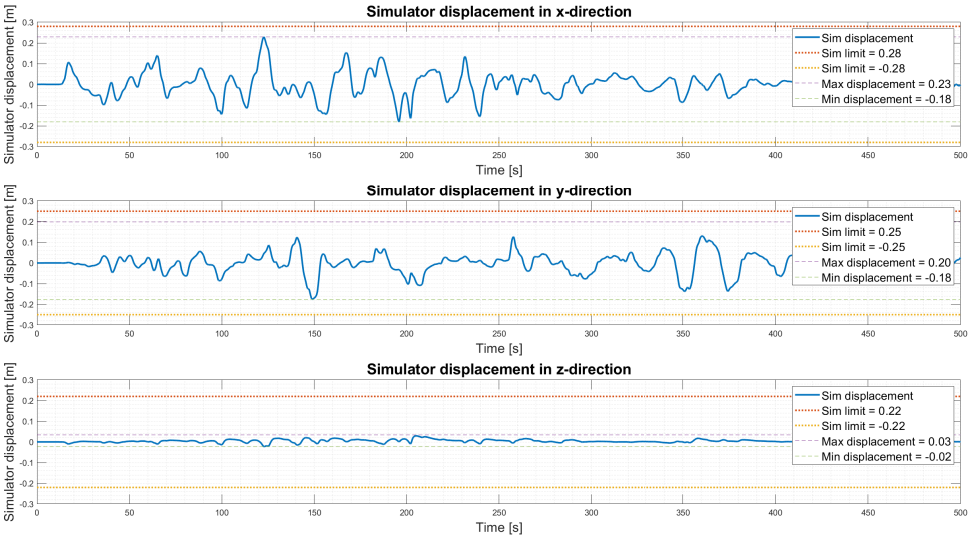


Figure A.5: CWA translational simulator displacement using BMW test drive as reference.

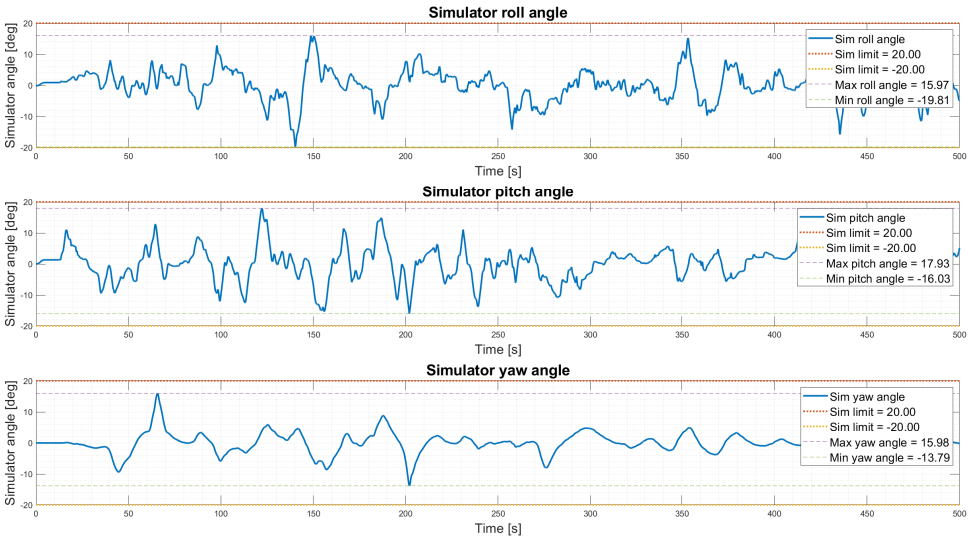


Figure A.6: CWA rotational simulator angles using BMW test drive as reference.

A.2. Chapter 3

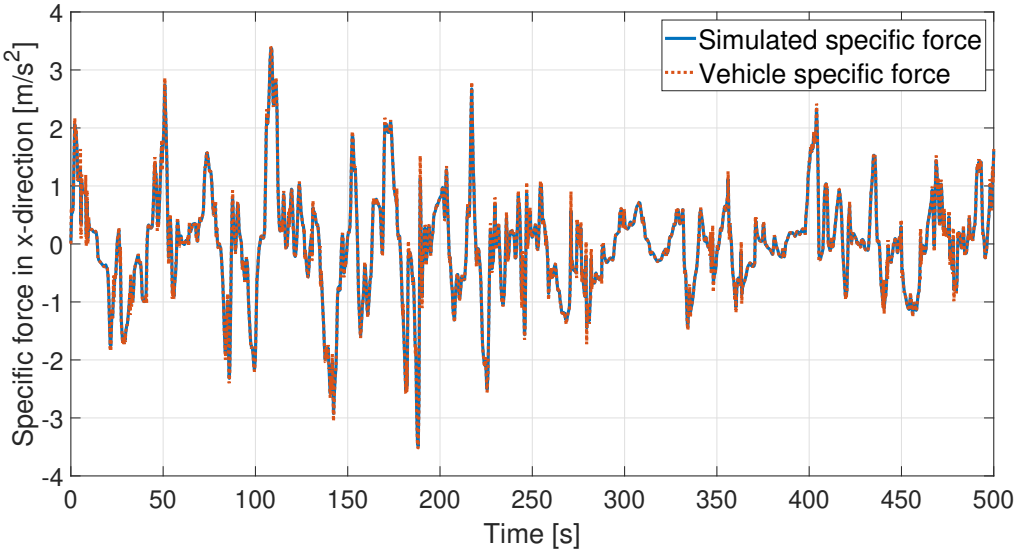


Figure A.7: Specific force in x-direction for 500s simulation using MPC with double integrator model.

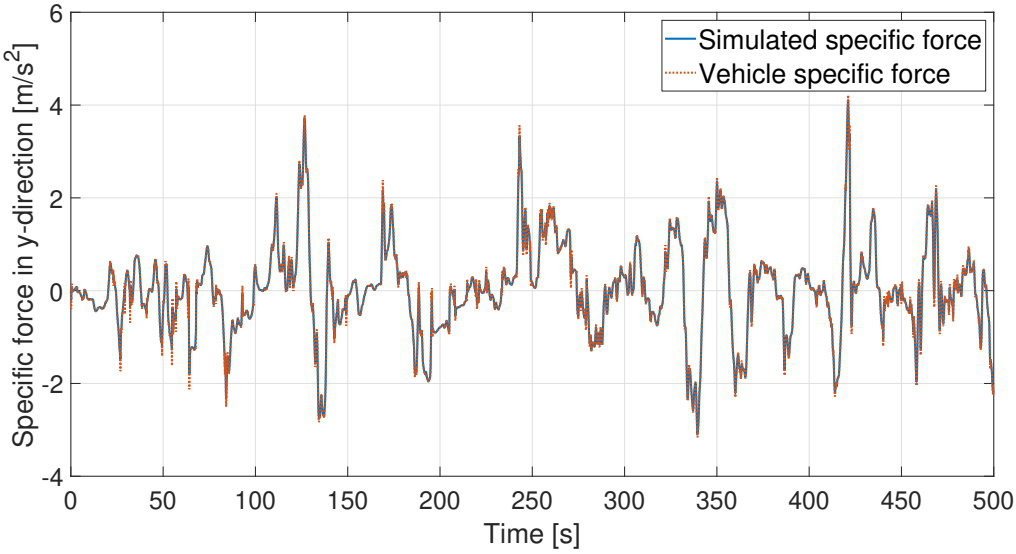


Figure A.8: Specific force in y-direction for 500s simulation using MPC with double integrator model.

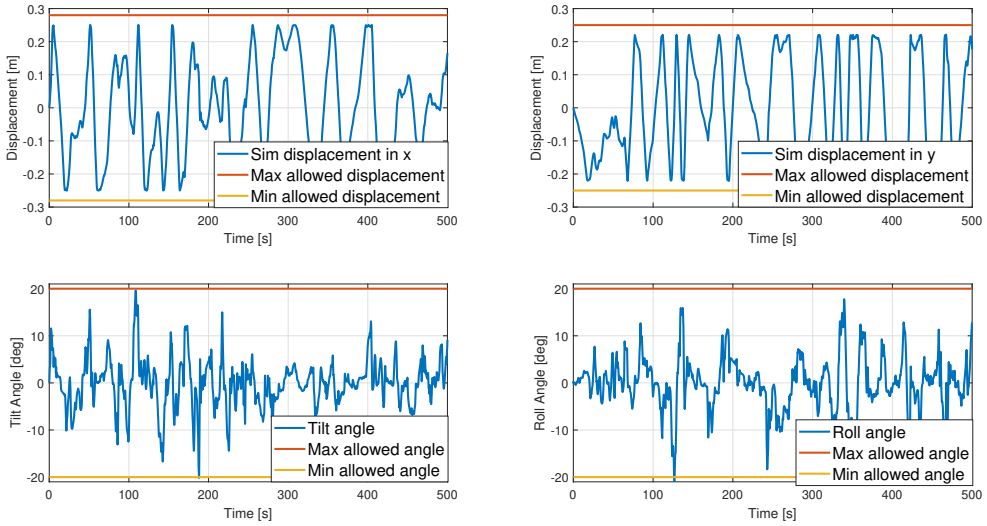


Figure A.9: Displacement in x- and y-direction, pitch and roll angle for 500s simulation using MPC with double integrator model.

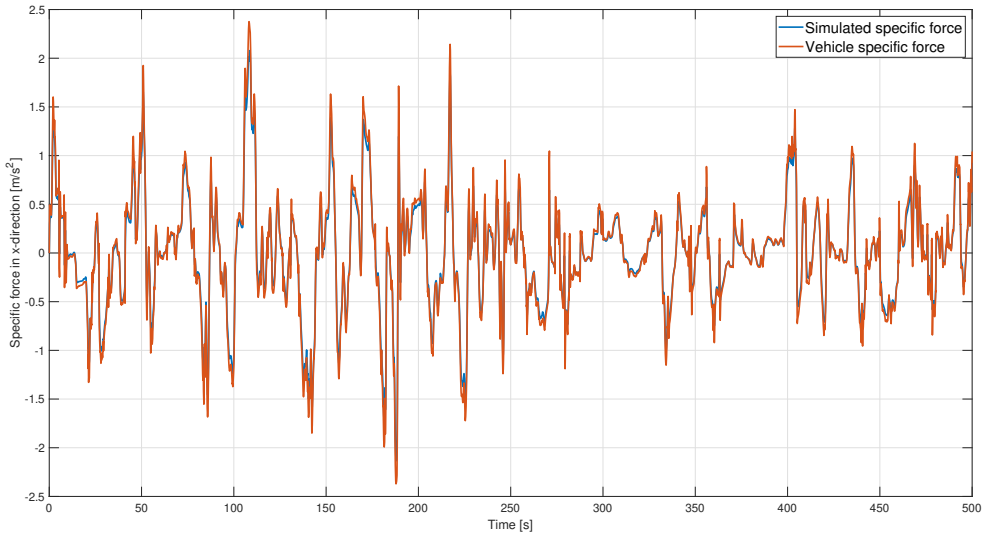


Figure A.10: Specific force in x-direction for 500s simulation using MPC with integrated vestibular model.

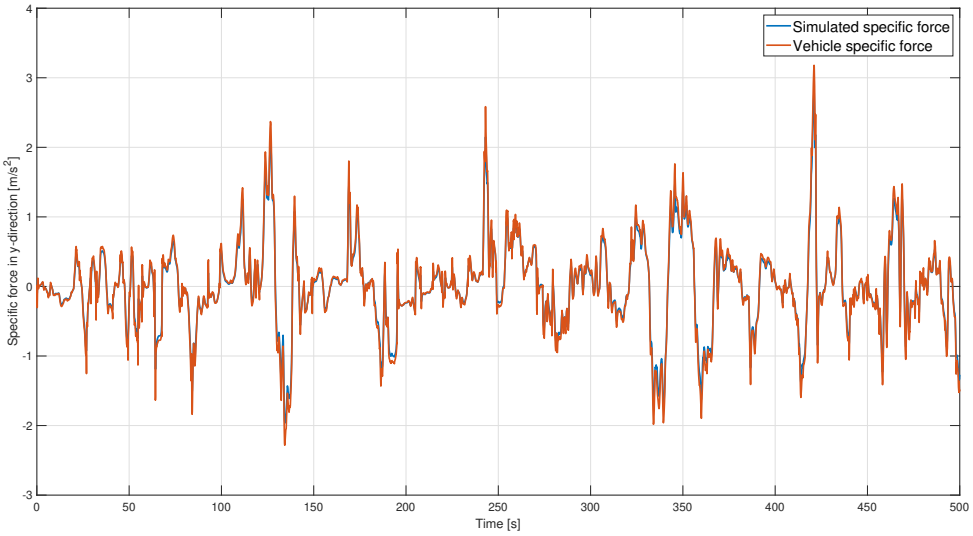


Figure A.11: Specific force in y-direction for 500s simulation using MPC with integrated vestibular model.

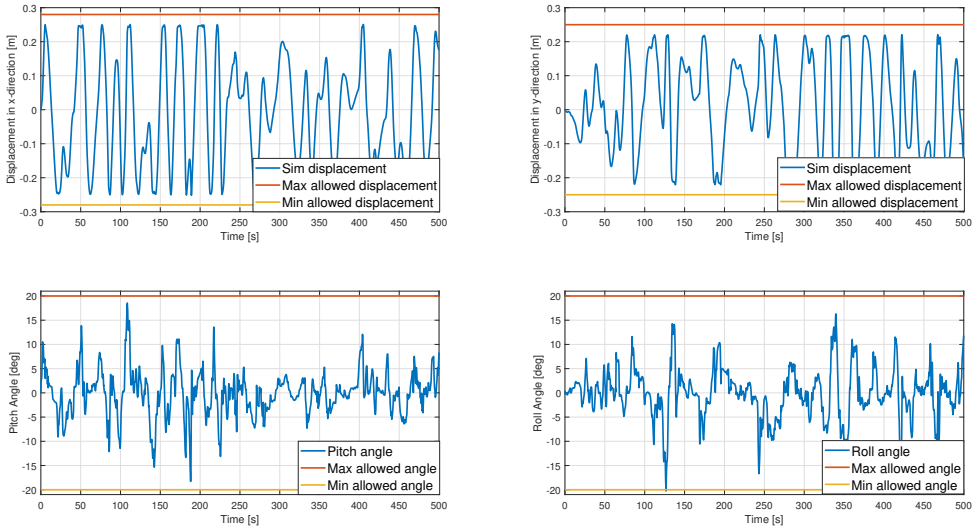


Figure A.12: Displacement in x- and y-direction, pitch and roll angle for 500s simulation using MPC with integrated vestibular model.

A.3. Chapter 4

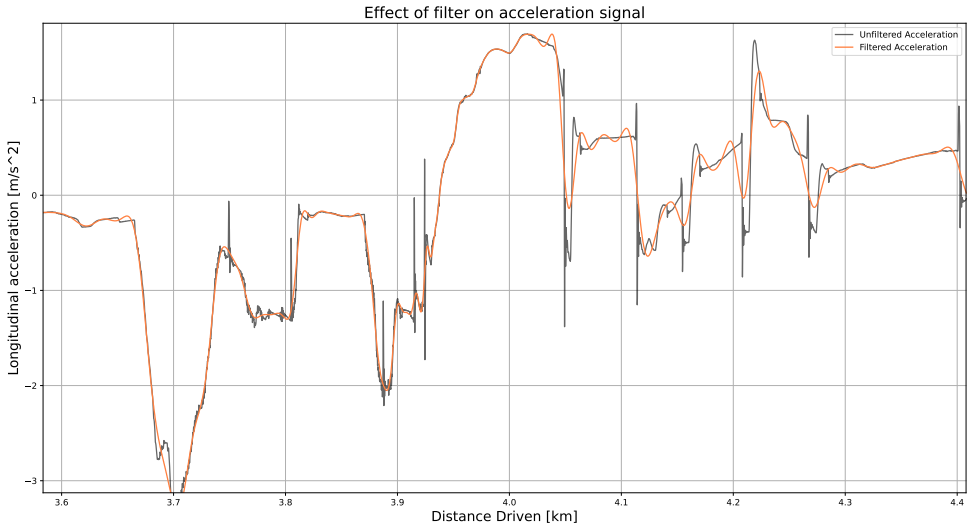


Figure A.13: Effect of 1Hz 5th-order lowpass butterworth filter on longitudinal acceleration.

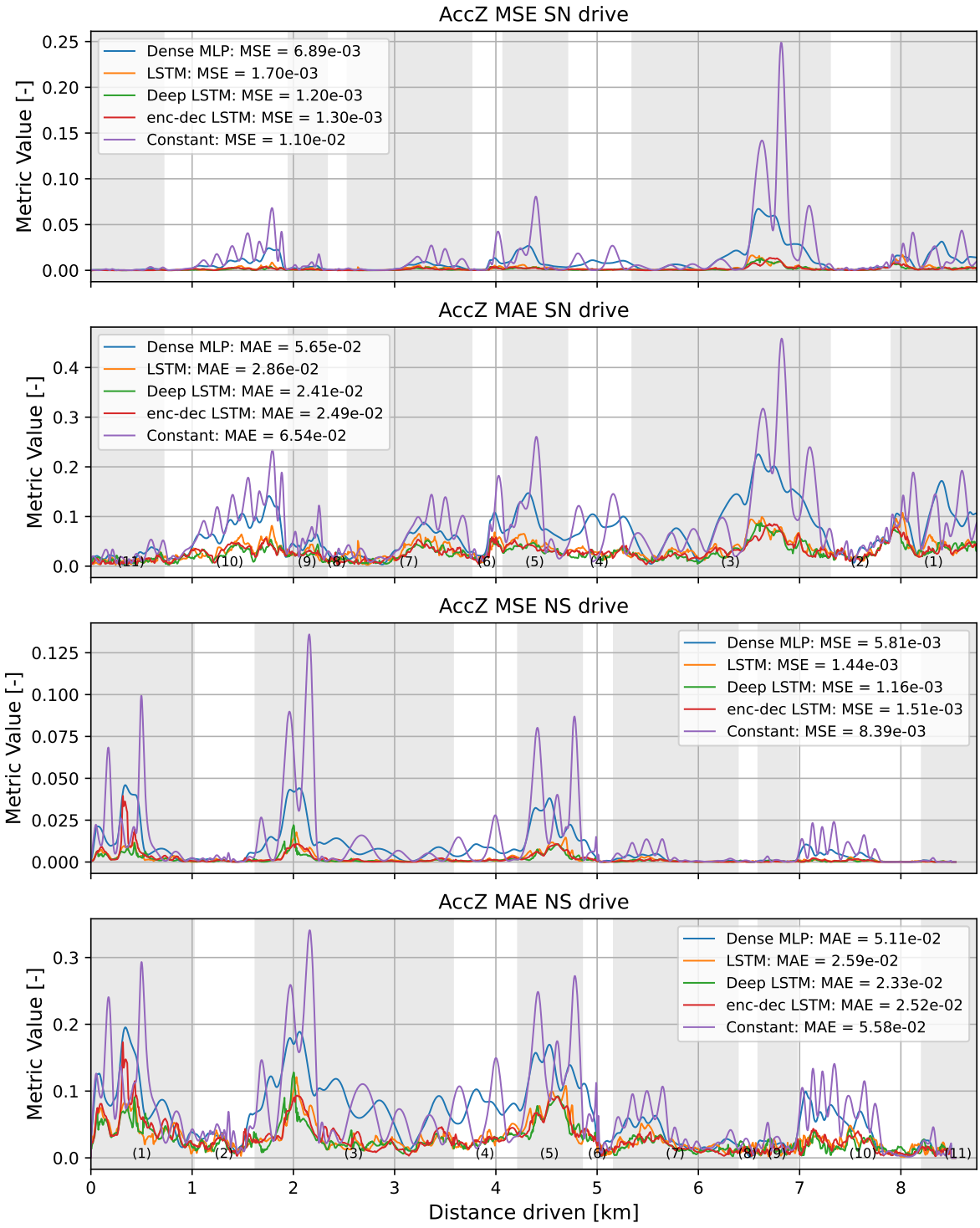


Figure A.14: MSE and MAE trace for vehicle vertical acceleration prediction for both NS- and SN-direction using a trained Dense MLP, single layer LSTM, Deep LSTM and Enc-Dec LSTM network as well as the MSE and MAE trace of a constant prediction.

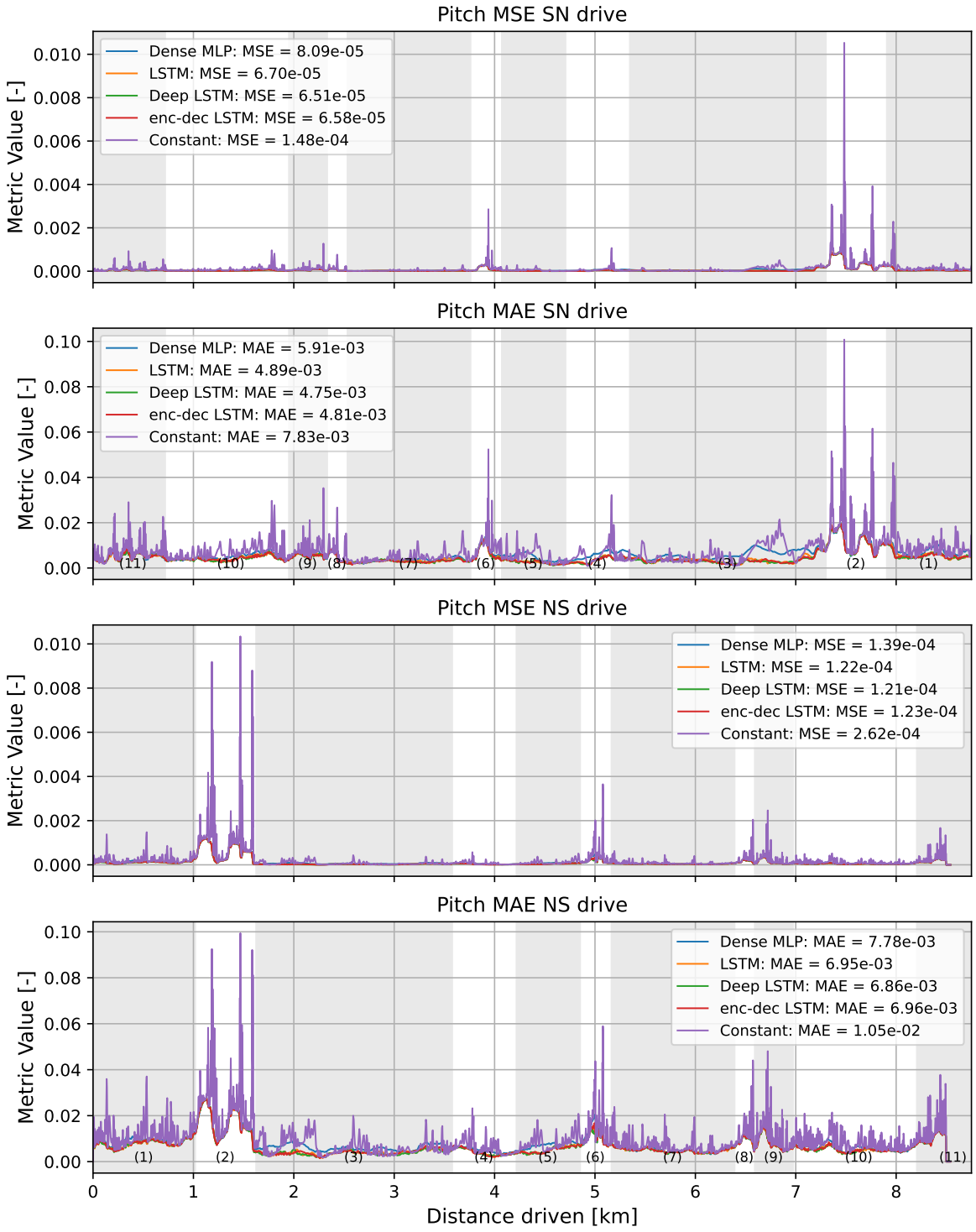


Figure A.15: MSE and MAE trace for vehicle pitch rate prediction for both NS- and SN-direction using a trained Dense MLP, single layer LSTM, Deep LSTM and Enc-Dec LSTM network as well as the MSE and MAE trace of a constant prediction.

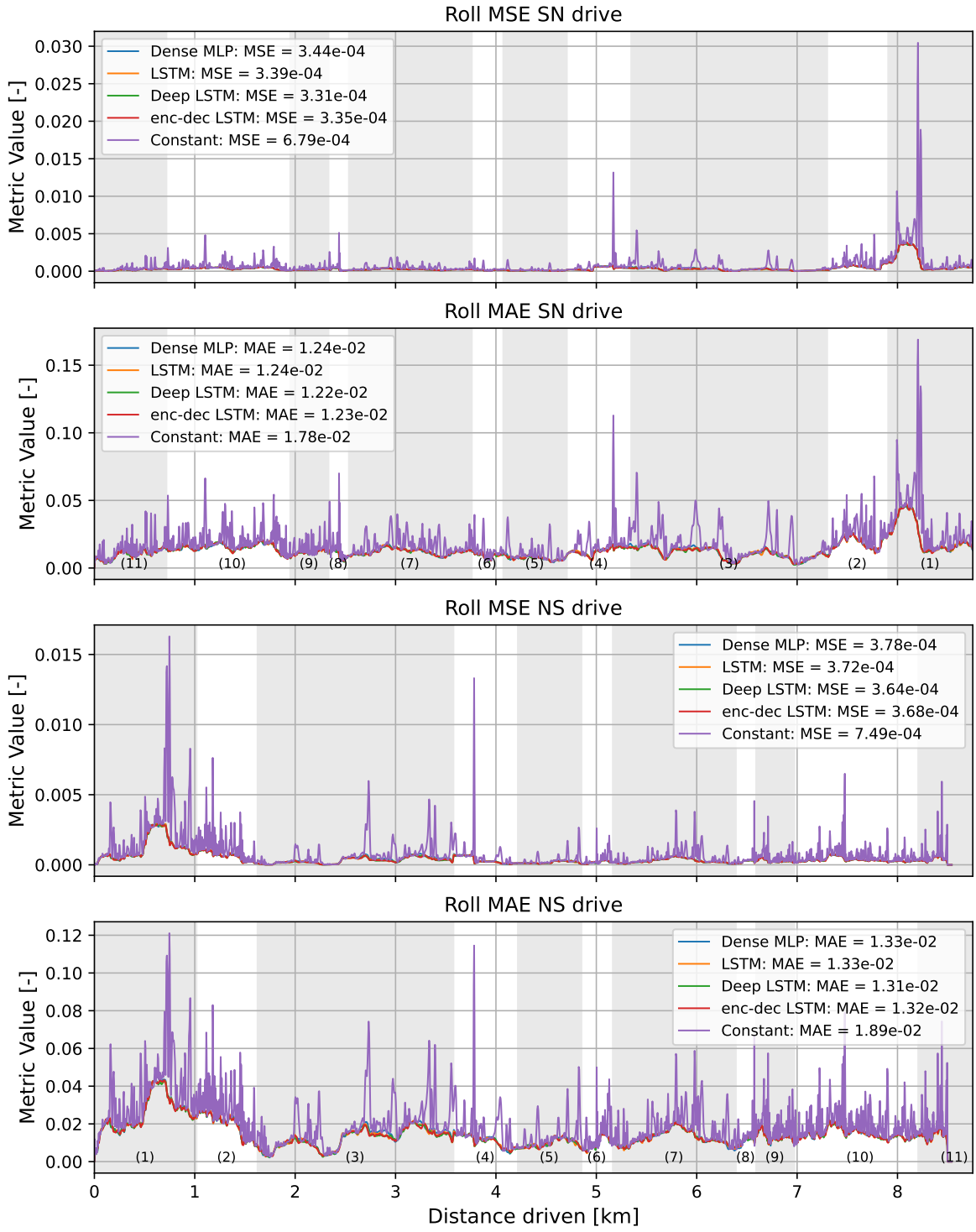


Figure A.16: MSE and MAE trace for vehicle roll rate prediction for both NS- and SN-direction using a trained Dense MLP, single layer LSTM, Deep LSTM and Enc-Dec LSTM network as well as the MSE and MAE trace of a constant prediction.

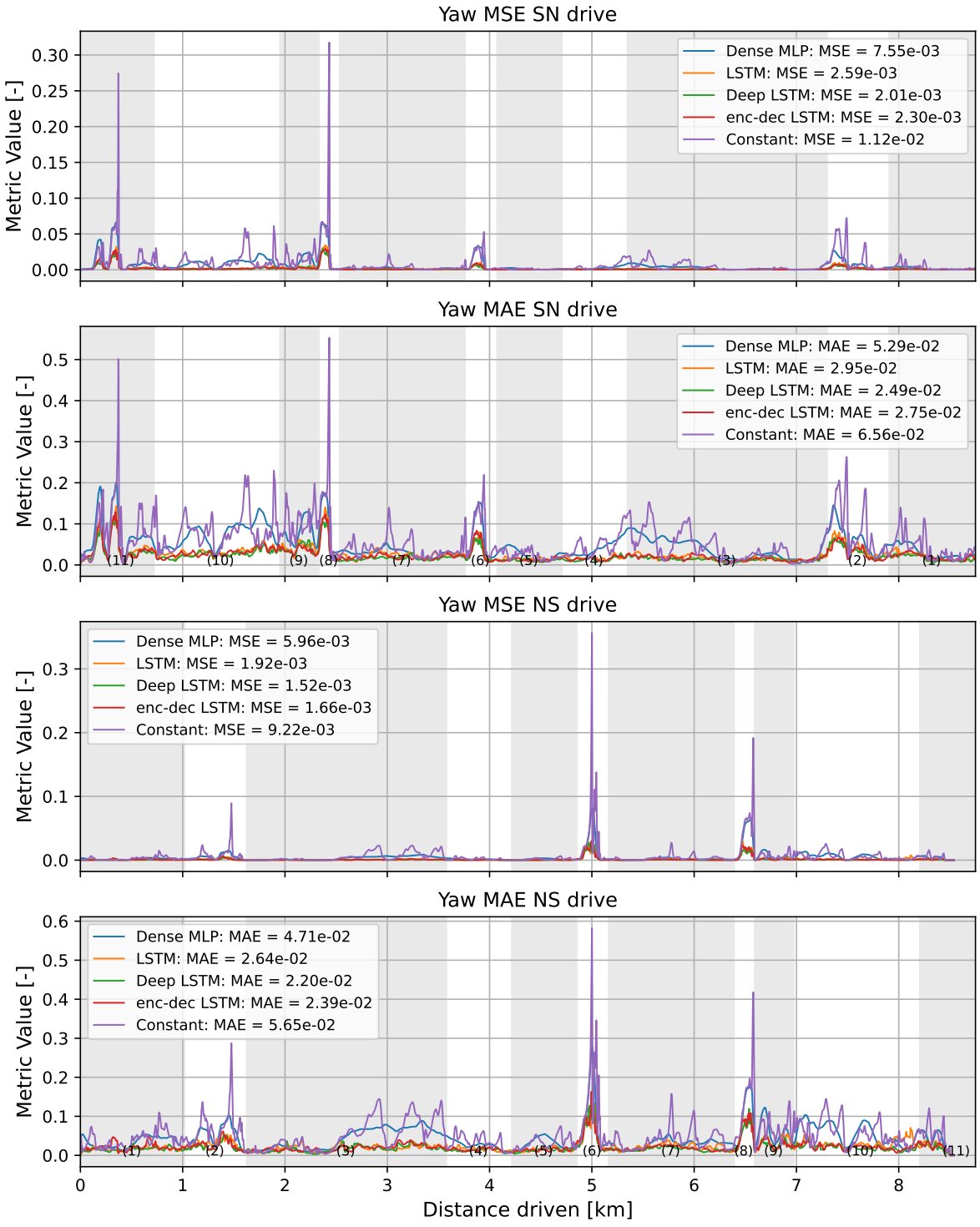


Figure A.17: MSE and MAE trace for vehicle yaw rate prediction for both NS- and SN-direction using a trained Dense MLP, single layer LSTM, Deep LSTM and Enc-Dec LSTM network as well as the MSE and MAE trace of a constant prediction.

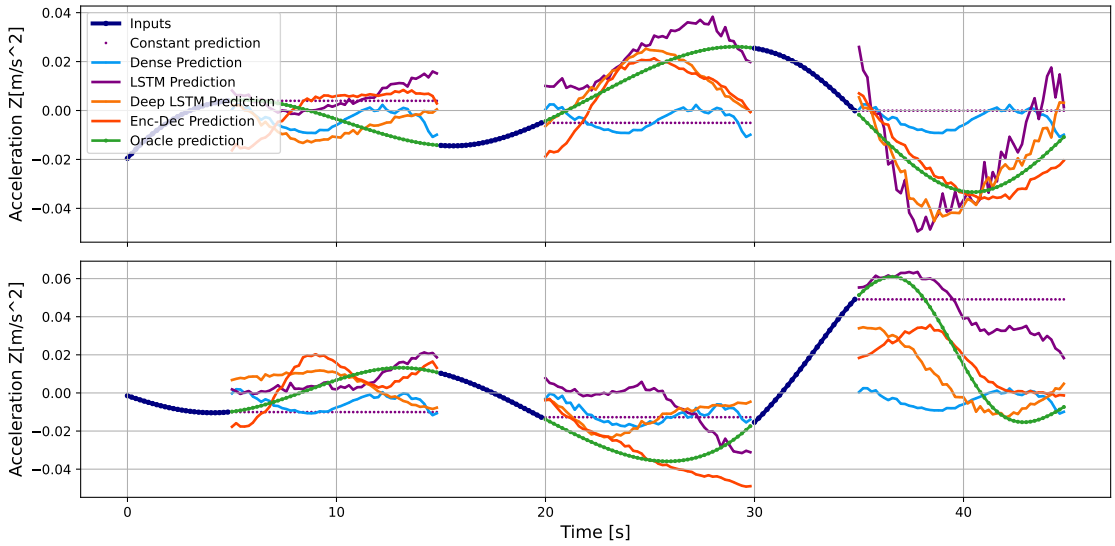


Figure A.18: Vertical acceleration prediction traces using a trained MLP, LSTM, Deep LSTM, Enc-Dec network as well as a constant prediction trace.

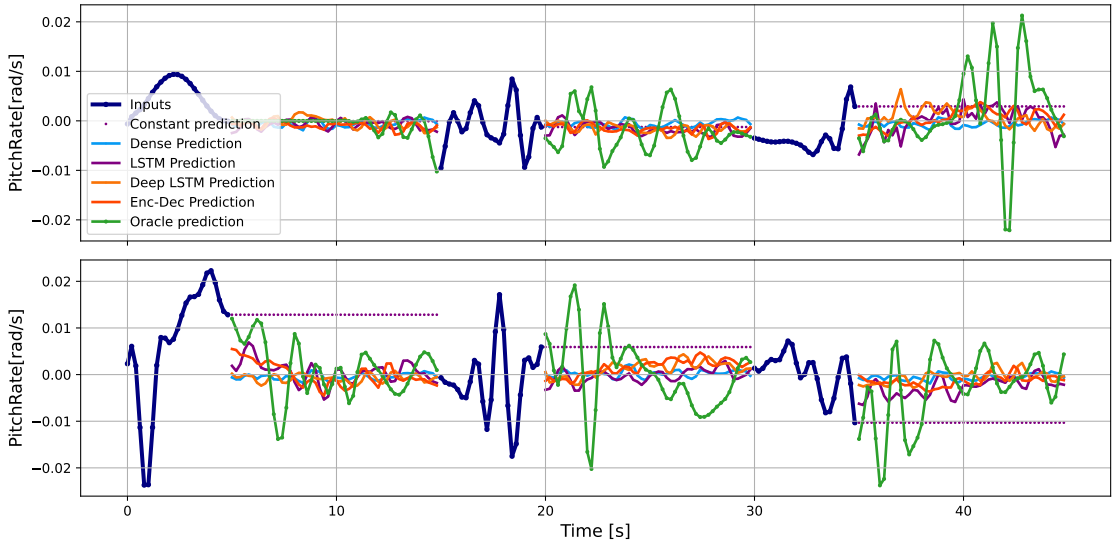


Figure A.19: Pitch rate prediction traces using a trained MLP, LSTM, Deep LSTM, Enc-Dec network as well as a constant prediction trace.

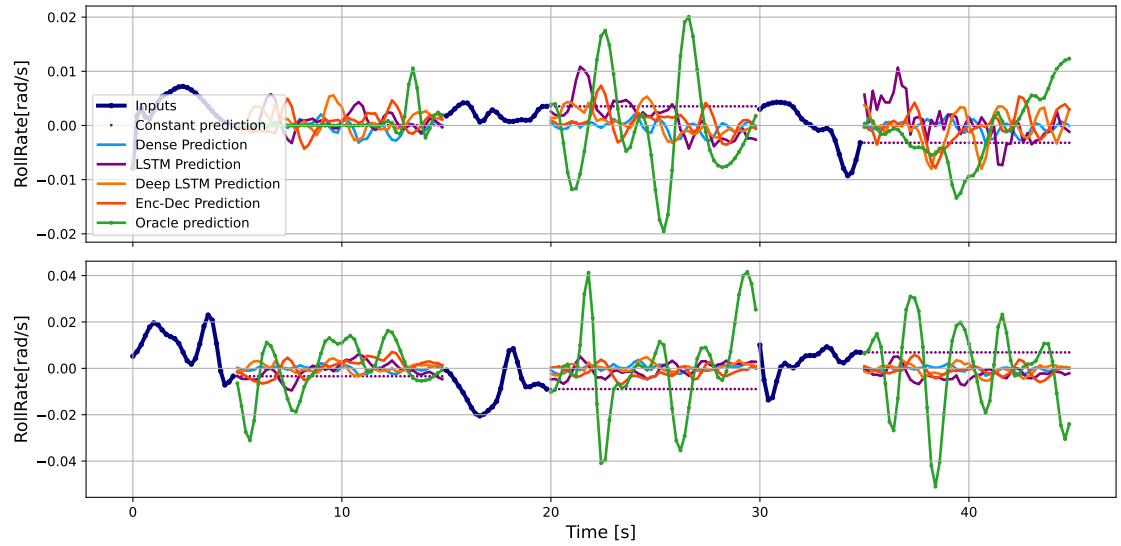


Figure A.20: Roll rate prediction traces using a trained MLP, LSTM, Deep LSTM, Enc-Dec network as well as a constant prediction trace.

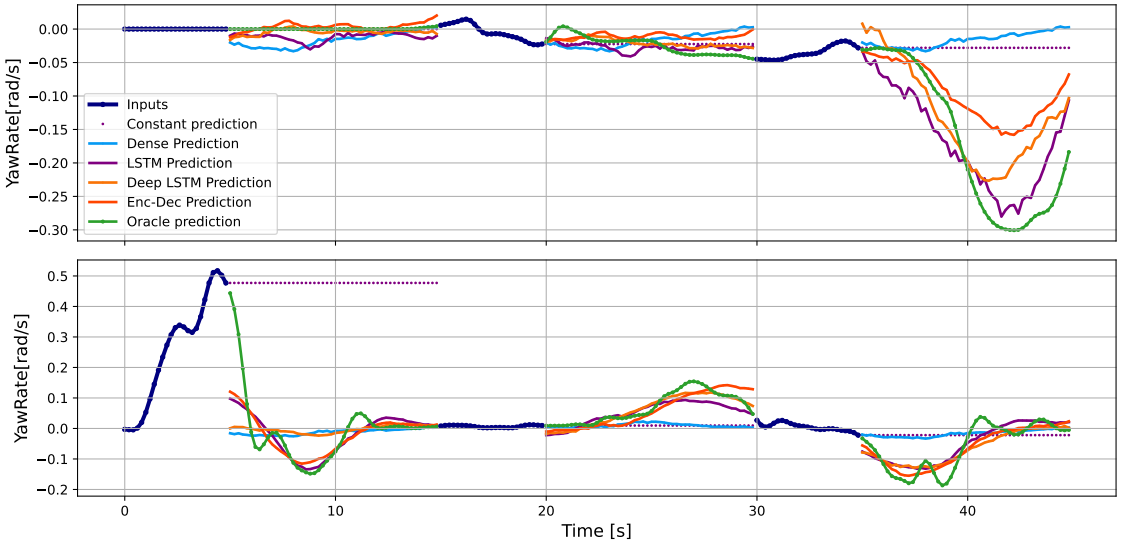


Figure A.21: Yaw rate prediction traces using a trained MLP, LSTM, Deep LSTM, Enc-Dec network as well as a constant prediction trace.

B

Appendix Paper

B.1. NS vs. SN Direction Additional Kinematic Vehicle Analysis

Figure B.1 and Figure B.2 show the kinematic profiles averaged over all drives in both NS and SN direction, respectively. In these figures, the effect of different sections on the average driver behavior can be investigated.

The highest acceleration peaks occur near regions where a change in speedlimit happens, i.e., when going from road segment *a* to *b*. However, two road segments stand-out to this '*rule*'. Road segment 6 features a round-about which has to be taken at moderated speed. As can be seen in both NS and SN direction, first, a strong deceleration is applied when approaching the round-about, shortly followed by an acceleration peak when exiting the round-about.

Road segment 8 is a transition segment. In NS direction, the speedlimit changes from $100km/h$, to $70kmg/h$, to $50km/h$ over a short distance, followed by two sharp turns. Therefore, in Figure B.1, a relatively long lasting deceleration maneuver can be seen, after which a higher acceleration peak occurs when past the last turn. Vice versa, in SN direction the speedlimit increases over the same short distance. Therefore, in Figure B.2 the opposite behavior can be seen, first a strong deceleration to account for the succession of sharp turns, followed by a strong acceleration to reach the speedlimit again.

Slow but sharp corners result in the requirement of a high yaw rate, and a peak in lateral acceleration when a small radius of curvature is driven at moderate speed. This behavior can be seen for both NS and SN direction in road segment 6 and 8, where a round-about and two sharp turns are present, respectively. This results in distinct yaw rate peaks in both segments. Still, a difference between NS and SN direction can be found when looking at Figure B.1 and Figure B.2, where the maximum average yaw peak in NS direction on road segment 6, is significantly higher during round-about entry (first two peaks), and similar to each other on exit (3rd peak). This is only logical as a participant driving in NS direction has to take the 3rd exit (in SN direction the 1st exit), which results in a yaw rate peak at entry, followed by a longer lasting and opposite yaw rate when driving around the round-about, and finally again an opposite yaw rate towards the exit. In SN direction, traversing the round-about is as if one short corner, with continuous road curvature has to be followed, resulting in only one peak in yaw rate. At high speed, but moderate radius of curve turns, lateral acceleration is more dominant, and only a more mild vehicle yaw rate is demanded. As can be seen, these type of corners dominate the driven route.

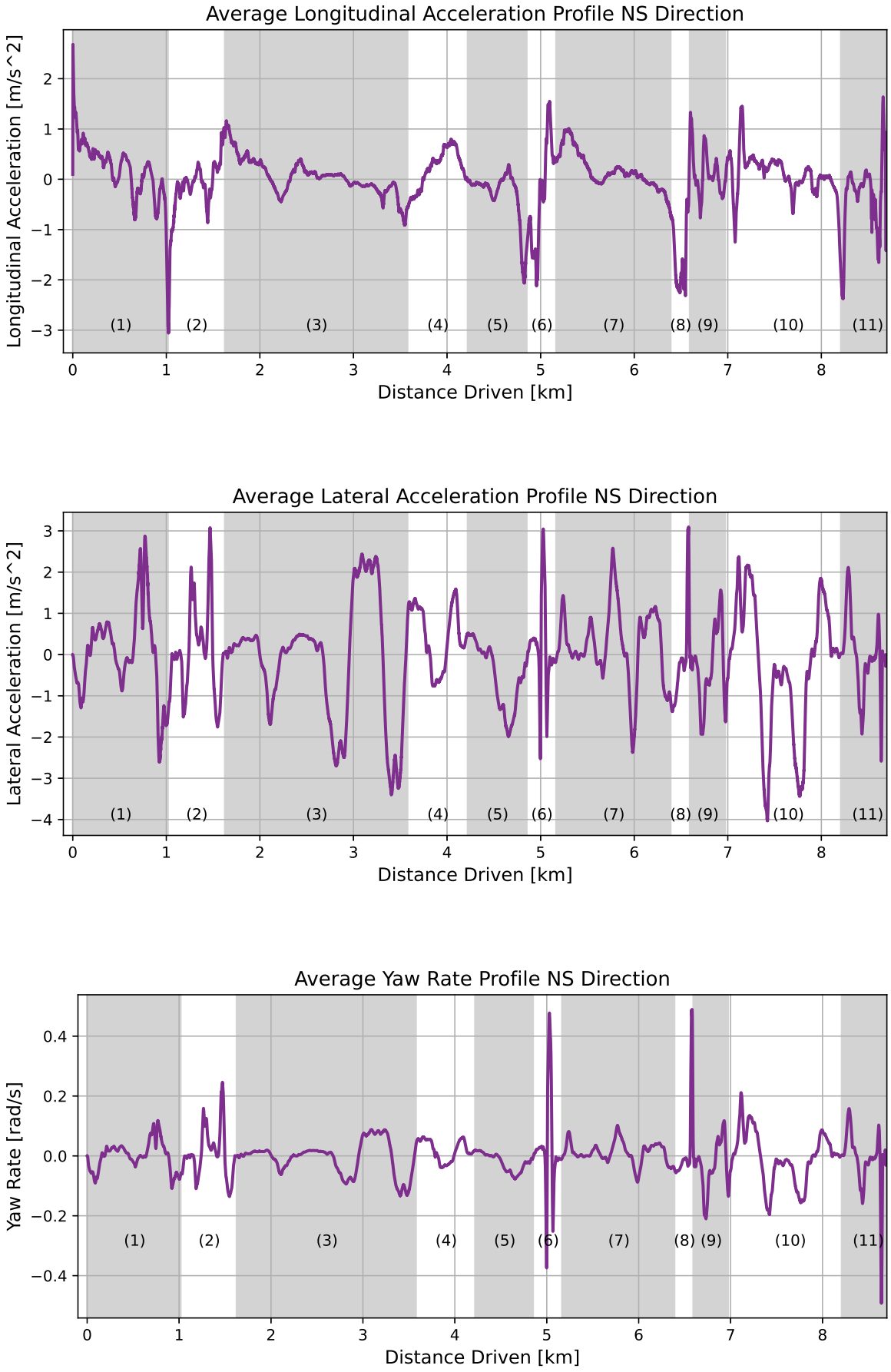


Figure B.1: The average kinematic profiles for all NS Drives

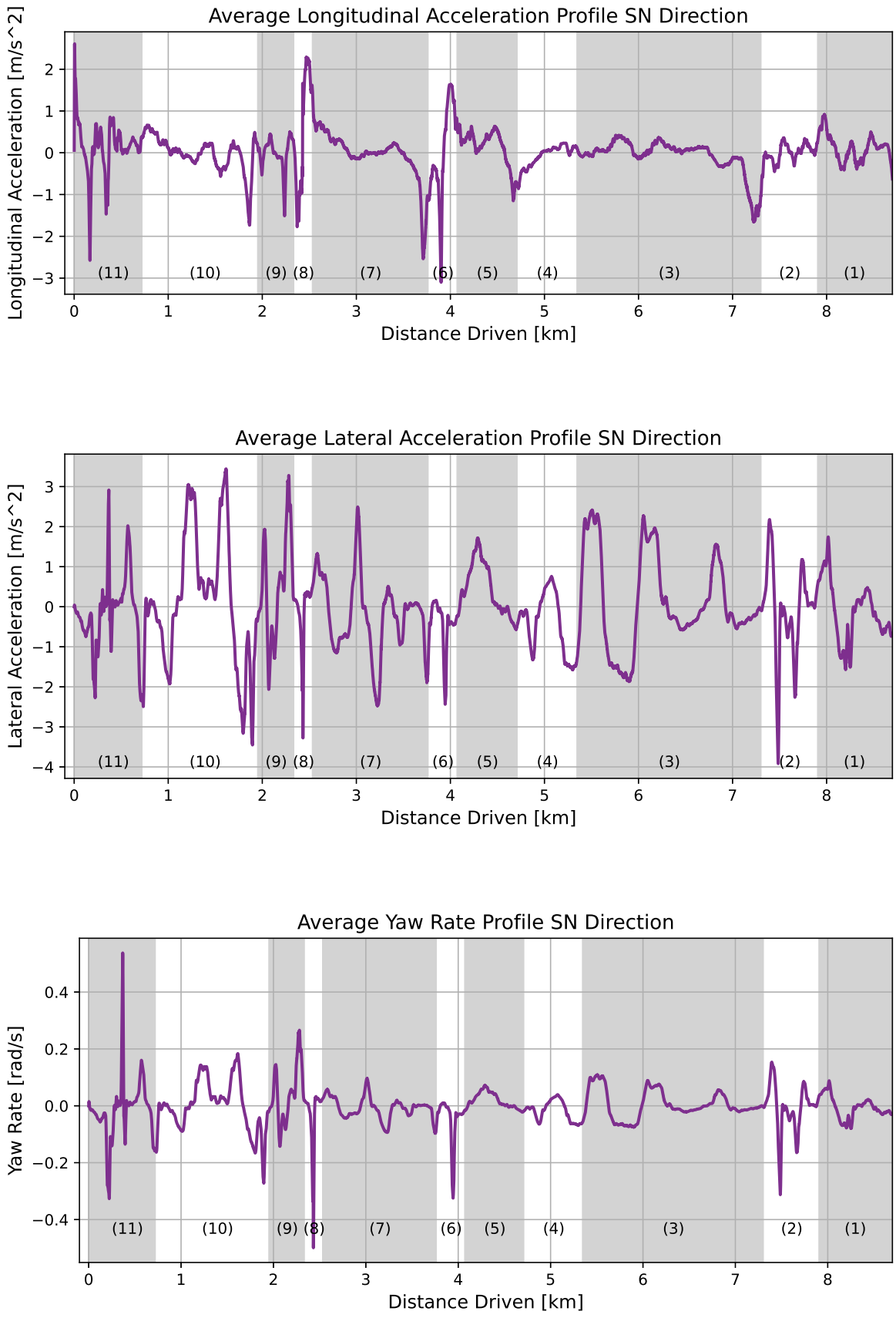


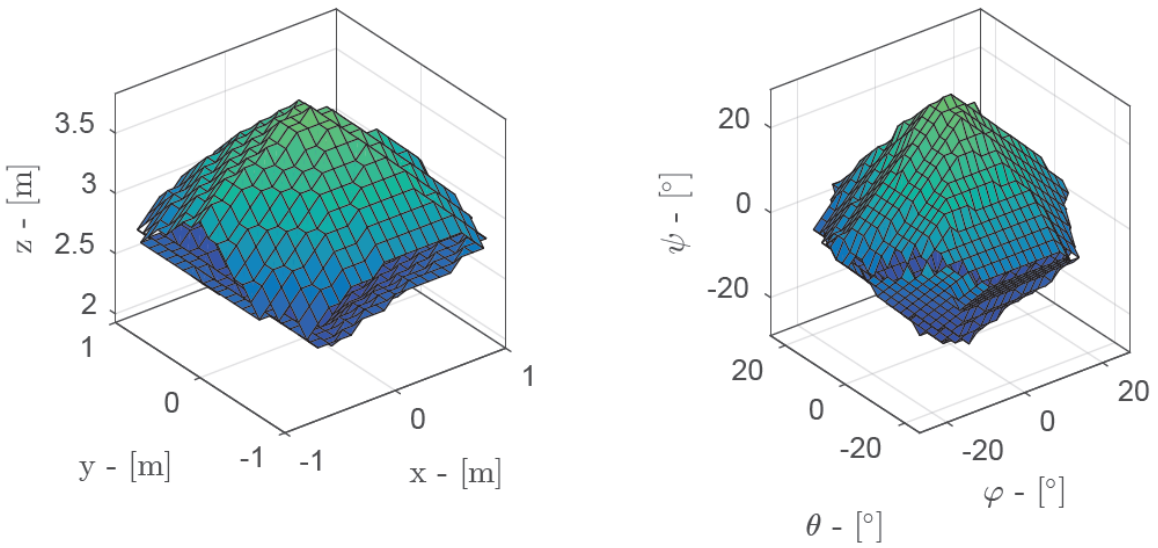
Figure B.2: The average kinematic profiles for all SN Drives

B.2. Nonlinear Workspace Domain

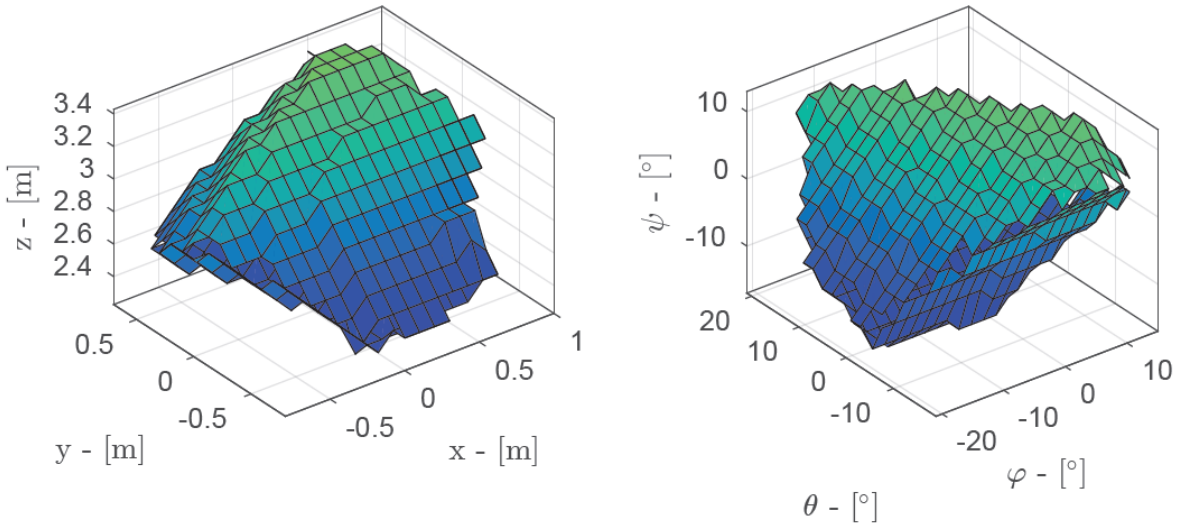
The presented illustrations in this section are retrieved from the PhD dissertation by Ellensohn[27]. Figure B.3 shows the available workspace in translational and rotational direction of the simulator when situated at the neutral position in Figure B.3a, and when diverted from the neutral position in Figure B.3b. The figures show how the polyhedron volume of reachable workspace changes nonlinearly when the simulator changes its relative position. Since the simulator is moved by its actuators, this behavior relates directly with the available actuator stroke length.

From these figures it becomes clear how the constant angle assumption over the prediction horizon N_p can affect the MPC computed output. When the simulator is positioned in its neutral state and excursions are required from the simulator, the MPC algorithm calculates the control input while adhering to the constraints imposed by the assumed-constant polyhedron in Figure B.3a. However, as can be seen in Figure B.3b, by tilting the simulator 10° around x-, y-, or z-direction, this polyhedron has already changed significantly, i.e., the actual physical constraints have been adjusted. In the MPC implementation defined in this report, this effect is ignored.

An MPC that incorporates these nonlinear constraints, has the ability to always adhere to the changing polyhedron constraints by updating the transformation matrices over the prediction horizon N_p for each time step. However, this would require a different nonlinear solver that is able to run real-time, which is a challenging problem.



(a) Available workspace when the simulator is in its neutral position,i.e., **Left:** $x, y, z = 0$ and **Right:** $\phi, \theta, \psi = 0°$.

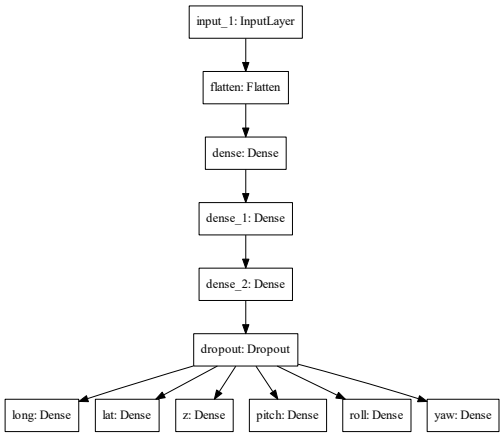


(b) Available workspace when the simulator is not in its neutral position,i.e., **Left:** $x, y = 0.5m$ and $z = 0$ and **Right:** $\phi, \theta, \psi = 10°$.

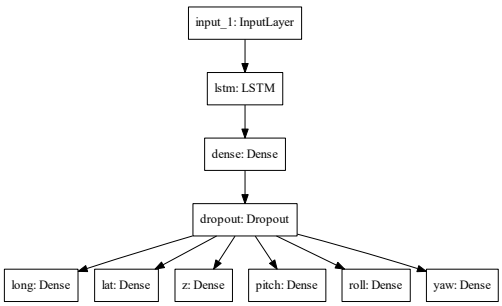
Figure B.3: Translational and rotational available workspace of a simulator in x,y,z and ϕ, θ, and ψ. Retrieved from [27].

B.3. Neural Network Model Structures

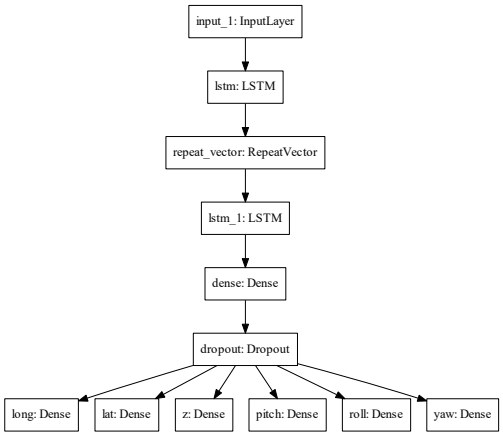
Figure B.4 shows the three other network structures mentioned in the paper. As discussed, the final three layers are equal for each of the networks.



(a) Three-layer Dense MLP.



(b) Single-layer LSTM RNN.



(c) Single-layer Encoder-Decoder LSTM RNN.

Figure B.4: The three other data-driven neural network structures, used to perform future vehicle state prediction.

B.4. MIR: W_1 and W_2 Analysis

As explained in the paper found in Part I, two different weight settings were used in the presented MPC MCA simulations. Figure B.5 shows the estimated MIR signals for a test drive in SN direction, highlighting one of the peaks occurring at $460s$. In this figure ratings for six different simulations are presented. For both W_1 and W_2: constant, Deep LSTM (denoted by 'LSTM'), and oracle MPC. Although, the cost on state deviations decreased by 20%, only a marginal decrease could be found in the estimated MIR signal. An explanation is given in the following.

Figure B.6 illustrates the difference in perceived specific force in both x- and y-direction for the same peak that was highlighted in Figure B.5. From these figures one can see that the difference between the two MPC outputs is indeed very small. Because the cost function features five different cost terms, adjusting one might not result in the expected change in output. An example of the cost value over time is given in the preliminary thesis in Figure 3.3. From these figures, and based on findings in [71], it was argued that not only the individual weight cost value, but also the ratio between weight term values and the absolute cost are important when an optimal control input is computed. If for example the assumption is made that the terminal cost is limiting, i.e., the simulator needs to be at the neutral point at the end of the prediction horizon N_p, reducing the state weighting by 20%, i.e., an absolute value decrease of 0.08 in p_x (see Table V in part I), is only a relative decrease of 0.032% with the terminal cost term. This means that the effect of the absolute neutral push of the state weighting decreases only by 0.032%, instead of the expected 20%. In this case the exact cost value distribution is not known, in future work it would be an interesting metric to keep track of by means of, e.g., a stacked area chart.

This discussion highlights, that even though it is claimed that the tuning of weight terms is more understandable than the tuning process of the washout MCA [22, 60], tuning of MPC parameters can still be very challenging in practice and is not as intuitive as one might think. Next to this, to understand the significance of results, it is advised to validate the results on more test drives which enables the use of statistical significance tests.

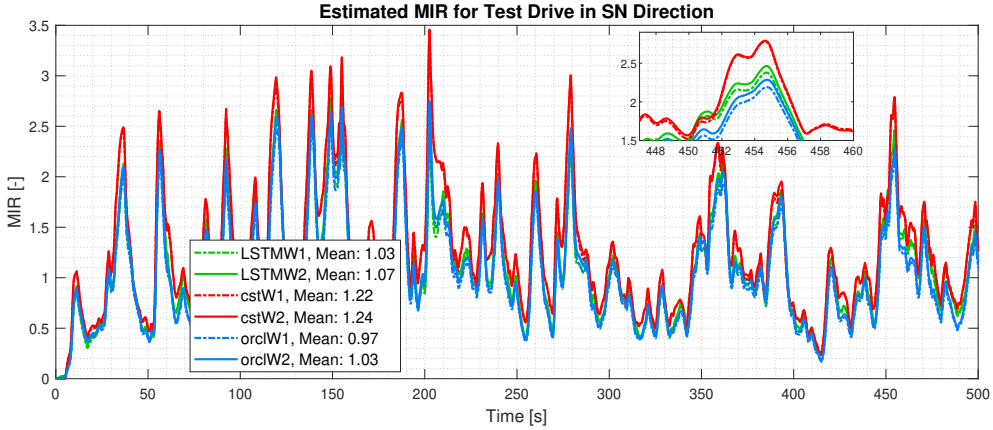


Figure B.5: Full rotational vehicle data from BMW test run.

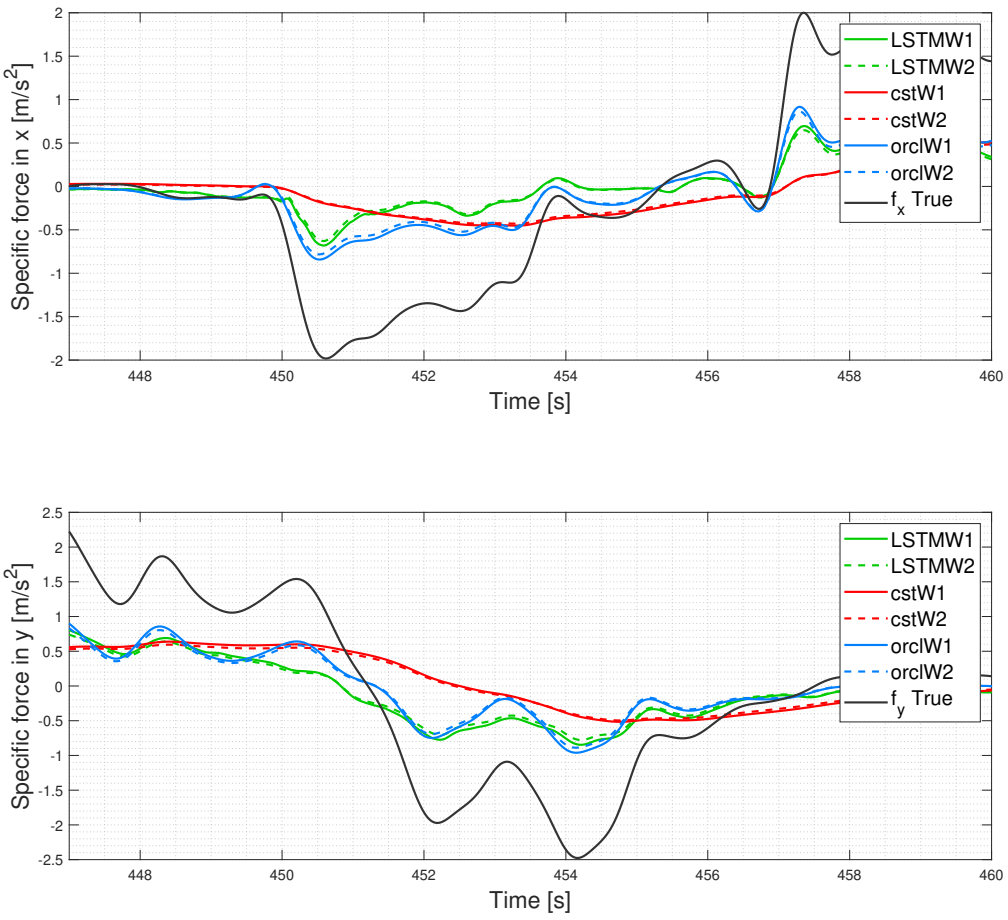


Figure B.6: Perceived specific force vs. true reference in x- and y-direction for three different MPC reference settings, each simulated using W_1 and W_2.